# Real-Time Verification for Distributed Cyber-Physical Systems

**Hoang-Dung Tran**
University of Nebraska-Lincoln, Lincoln, Nebraska, USA

**Luan Viet Nguyen**
University of Dayton, Dayton, Ohio, USA

**Patrick Musau**
Vanderbilt University, Nashville, Tennessee, USA

**Weiming Xiang**
Augusta University, Nashville, Tennessee, USA

**Taylor T. Johnson**
Vanderbilt University, Nashville, Tennessee, USA

## Abstract

Safety-critical distributed cyber-physical systems (CPSs) have been found in a wide range of applications. Notably, they have displayed a great deal of utility in intelligent transportation, where autonomous vehicles communicate and cooperate with each other via a high-speed communication network. Such systems require an ability to identify maneuvers in real-time that cause dangerous circumstances and ensure the implementation always meets safety-critical requirements. In this paper, we propose a real-time decentralized reachability approach for safety verification of a distributed multi-agent CPS with the underlying assumption that all agents are time-synchronized with a low degree of error. In the proposed approach, each agent periodically computes its local reachable set and exchanges this reachable set with the other agents with the goal of verifying the system safety. Our method, implemented in Java, takes advantages of the timing information and the reachable set information that are available in the exchanged messages to reason about the safety of the whole system in a decentralized manner. Any particular agent can also perform local safety verification tasks based on their local clocks by analyzing the messages it receives. We applied the proposed method to verify, in real-time, the safety properties of a group of quadcopters performing a distributed search mission.

**2012 ACM Subject Classification** Computing methodologies → Distributed computing methodologies
**Keywords and Phrases** Verification, Reachability Analysis, Distributed Cyber-Physical Systems
**Digital Object Identifier** 10.4230/LITES.8.2.7
**Related Version** *Previous Version*: https://doi.org/10.1007/978-3-030-21759-4_15 [31]
**Supplementary Material** *Software (Source Code)*: https://github.com/verivital/rtreach
**Funding** The material presented in this paper is based upon work supported by the Air Force Office of Scientific Research (AFOSR) through contract number FA9550-22-1-0019 and the Defense Advanced Research Projects Agency (DARPA) through contract number FA8750-18-C-0089. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFOSR or DARPA.
**Received** 2020-10-22 **Accepted** 2022-01-28 **Published** 2022-12-07
**Editor** Alessandro Abate, Uli Fahrenberg, and Martin Fränzle
**Special Issue** Special Issue on Distributed Hybrid Systems



## 1 Introduction

The emergence of 5G technology has inspired a massive wave of the research and development in science and technology in the era of IoT where the communication between computing devices has become significantly faster with lower latency and power consumption. The power of this modern communication technology influences and benefits all aspects of Cyber-Physical Systems (CPSs) such as smart grids, smart homes, intelligent transportation and smart cities. In particular, the study of autonomous vehicles has become an increasingly popular research field in both academic and industrial transportation applications. Automotive crashes pose significant financial and life-threatening risks, and there is an urgent need for advanced and scalable methods that can efficiently verify a distributed system of autonomous vehicles.

Over the last two decades, although many methods have been developed to conduct reachability analysis and safety verification of CPS, such as the approaches proposed in [1, 4, 5, 7, 13, 14, 17, 20, 23, 30, 32], applying these techniques to *real-time distributed* CPS remains a big challenge. This is due to the fact that, 1) all existing techniques have intensive computation costs and are usually too slow to be used in a real-time manner and, 2) these techniques target the safety verification of a *single* CPS, and therefore they naturally cannot be applied efficiently to a *distributed* CPS where clock mismatches and communication between agents (i.e., individual systems) are essential concerns. Since the future autonomous vehicles systems will work distributively involving effective communication between each agent, there is an urgent need for an approach that can provide formal guarantees of the safety of distributed CPS in real-time. More importantly, the safety information should be defined based on the *agents local clocks* to allow these agents to perform "intelligent actions" to escape from the upcoming dangerous circumstances. For example, if an agent A knows based on its local clock that it will collide with an agent B in the next 5 seconds, it should perform an action such as stopping or quickly finding a safe path to avoid the collision.

In this paper[1], we propose a *decentralized real-time reachability* approach for safety verification of a distributed CPS with multiple agents. We are particularly interested in two types of safety properties. The first one is a *local safety property* which specifies the local constraints of the agent operation. For example, each agent is only allowed to move within a specific region, does not hit any obstacles, and its velocity needs to be limited to a specific range. This type of property does not require the information of other agents and can be verified locally at run-time. The second safety property is a *global property* defined on the states of multiple agents. Particularly, we consider a peer-to-peer collision free property and a generalized property where we want to verify if all agents satisfy a set of linear constraints (on the states of all agents) defining the property, e.g., two agents do not go into the same region at the same time.

Our decentralized real-time reachability approach works as follows. Each agent *locally* and *periodically computes* the local reachable set of states from the current local time to the next $T$ seconds, and then *encodes* and *broadcasts* its reachable set information to the others via a communication network. When the agent receives a reachable set message, it immediately *decodes* the message to read the reachable set information of the sender, and then performs *peer-to-peer collision checking* based on its current state and the reachable set of the sender. Verifying a generalized global property involving the states of $N$ agents is done at the time an agent receives all needed reachable sets from other agents. Additionally, the local safety property of the agent is verified simultaneously with the reachable set computation process at run-time. The proposed verification approach is based on an underlying assumption that is, all agents are time-synchronized to some level of accuracy. This assumption is reasonable as it can be achieved by using existing

[1] This paper is an extension of [31].

time synchronization protocols such as the Network Time Protocol (NTP). Our approach has successfully verified in real-time the local safety properties and collision occurrences for a group of quadcopters conducting a search mission.

The rest of the paper is organized as follows. Section 2 presents briefly the distributed CPS modeling and its verification problems. Section 3 gives the detail of real-time reachability for single agent and how to use it for real-time local safety verification. Section 4 addresses the utilization reachable set messages for checking peer-to-peer collision. Section 5 investigates the global safety verification problem. Section 6 presents the implementation and evaluation of our approach via a distributed search application using quadcopters.

## 2 Problem Formulation

In this paper, we consider a distributed CPS with $N$ agents that can communicate with each other via an asynchronous communication channel.

### Communication Model

The communication between agents is implemented by the *actions* of sending and receiving messages over an asynchronous communication channel. We formally model this communication model as a single automaton, Channel, which stores the set of in-flight messages that have been sent, but are yet to be delivered. When an agent sends a message $m$, it invokes a *send(m)* action. This action adds $m$ to the *in-flight* set. At any arbitrary time, the Channel chooses a message in the in-flight set to either delivers it to its recipient or removes it from the set. All messages are assumed to be unique and each message contains its sender and recipient identities. Let $M$ be the set of all possible messages used in communication between agents. The sending and receiving messages by agent $i$ are denoted by $M_{i,*}$ and $M_{*,i}$, respectively.

### Agent Model

The $i^{th}$ agent is modeled as a hybrid automaton [16, 27] defined by the tuple $\langle \mathcal{A}_i = V_i, A_i, \mathcal{D}_i, \mathcal{T}_i \rangle$, where:

**a)** $V_i$ is a set of variables consisting of the following:
- **i)** a set of continuous variables $X_i$ including a special variable $clk_i$ which records the agent's *local time*, and
- **ii)** a set of discrete variables $Y_i$ including the special variable $msghist_i$ that records all sent and received messages. A valuation $\mathbf{v}_i$ is a function that associates each $v_i \in V_i$ to a value in its type. We write $val(V_i)$ for the set of all possible valuations of $V_i$. We abuse the notion of $\mathbf{v}_i$ to denote a state of $\mathcal{A}_i$, which is a valuation of all variables in $V_i$.

The set $Q_i \triangleq val(V_i)$ is called the set of *states*.

**b)** $A_i$ is a set of *actions* consisting of the following subsets:
- **i)** a set $\{send_i(m) \mid m \in M_{i,*}\}$ of send actions (i.e., output actions),
- **ii)** a set $\{receive_i(m) \mid m \in M_{*,i}\}$ of receive actions (i.e., input actions), and
- **iii)** a set $H_i$ of other, ordinary actions.

**c)** $\mathcal{D}_i \subseteq val(V_i) \times A_i \times val(V_i)$ is called the set of *transitions*. For a transition $(\mathbf{v}_i, a_i, \mathbf{v}'_i) \in \mathcal{D}_i$, we write $\mathbf{v}_i \xrightarrow{a_i} \mathbf{v}'_i$ in short.
- **i)** If $a_i = send_i(m)$ or $receive_i(m)$, then all the components of $\mathbf{v}_i$ and $\mathbf{v}'_i$ are identical except that $m$ is added to $msghist$ in $\mathbf{v}'_i$. That is, the agent's other states remain the same on message sends and receives. Furthermore, for every state $\mathbf{v}_i$ and every receive action $a_i$, there must exist a $\mathbf{v}'_i$ such that $\mathbf{v}_i \xrightarrow{a_i} \mathbf{v}'_i$, i.e., the automaton must have well-defined behavior for receiving any message in any state.
- **ii)** If $a_i \in H_i$, then $\mathbf{v}_i.msghist = \mathbf{v}'_i.msghist$.

**d)** $\mathcal{T}_i$ is a collection of trajectories for $X_i$. Each trajectory of $X_i$ is a function mapping an interval of time $[0, t], t \geq 0$ to $val(V_i)$, following a flow rate that specifies how a real variable $x_i \in X_i$ evolving over time. We denote the *duration* of a trajectory as $\tau_{dur}$, which is the right end-point of the interval $t$.

### Agent Semantics

The *behavior* of each agent can be defined based on the concept of an *execution* which is a particular run of the agent. Given an initial state $\mathbf{v}_i^0$, an *execution* $\alpha_i$ of an agent $A_i$ is a sequence of states starting from $\mathbf{v}_i^0$, defined as $\alpha_i = \mathbf{v}_i^0, \mathbf{v}_i^1, \ldots$, and for each index $j$ in the sequence, the state update from $\mathbf{v}_i^j$ to $\mathbf{v}_i^{j+1}$ is either a transition or trajectory. A state $\mathbf{v}_i^j$ is *reachable* if there exists an executing that ends in $\mathbf{v}_i^j$. We denote $\mathsf{Reach}(A_i)$ as the reachable set of agent $A_i$.

### System Model

The formal model of the complete system, denoted as $\mathsf{System}$, is a network of hybrid automata that is obtained by parallel composing the agent's models and the communication channel. Formally, we can write, $\mathsf{System} \triangleq \mathcal{A}_1 \| \ldots \mathcal{A}_N \| \mathsf{Channel}$. Informally, the agent $\mathcal{A}_i$ and the communication channel $\mathsf{Channel}$ are synchronized through sending and receiving actions. When the agent $A_i$ sends a message $m \in M_{i,j}$ to the agent $A_j$, it triggers the $send_i(m)$ action. At the same time, this action is synchronized in the $\mathsf{Channel}$ automaton by putting the message $m$ in the *in-flight* set. After that, the $\mathsf{Channel}$ will trigger (non-deterministically) the $receive_j(m)$ action. This action is synchronized in the agent $A_j$ by putting the message $m$ into the message history $msghist_j$.

In this paper, we investigate three real-time safety verification problems for distributed cyber-physical systems as defined in the following.

▶ Problem 1 (Local safety verification in real-time). The real-time local safety verification problem is to compute online the reachable set $\mathsf{Reach}(A_i)$ of the agent and verify if it violates the local safety property, i.e., checking $\mathsf{Reach}(A_i) \cap \mathcal{U}_i = \emptyset?$, where $\mathcal{U}_i \triangleq \{x_i | C_i x_i \leq d_i, x_i \in X_i\}$ is the unsafe set of the agent.

▶ Problem 2 (Decentralized real-time collision verification). The decentralized real-time collision verification problem is to reason in real-time whether an agent $A_i$ will collide with other agents from its current local time $t_c^i$ to the *computable, safe time instance in the future* $T_{safe}$ based on

**i)** the *clock mismatches*, and

**ii)** the *exchanging reachable set messages* between agents.

Formally, we require that $\forall\ t_c^i \leq t \leq T_{safe}, d_{ij}(t) \geq l$, where $d_{ij}(t)$ is the distance between agents $A_i$ and $A_j$ at the time $t$ of the agent $A_i$ local clock, and $l$ is the allowable safe distance between agents.

▶ Problem 3 (Decentralized real-time global safety verification). The decentralized real-time global safety verification problem is to construct online (at each agent) the reachable set of all agents $globalReach$ and verify if it violates the global safety property, i.e., checking $globalReach \cap \mathcal{U} = \emptyset$, where $\mathcal{U} \triangleq Cx \leq d$, $x = [x_1^T, \ldots, x_N^T]^T, x_i \in X_i$, is the unsafe set of the whole system.

## 3 Real-Time Local Safety Verification

The first important step in our approach is, each agent $A_i$ computes forwardly its reachable set of states from the current local time $t^i$ to the next $(t^i + T)$ seconds which is defined by $\mathcal{R}_i[t^i, t^i + T]$. Since there are many variables used in the agent modeling that are irrelevant in safety verification, we only need to compute the reachable set of state that is related to the agent's physical dynamics

(so called as *motion dynamics*) which is defined by a nonlinear ODE $\dot{x}_i = f(x_i, u_i)$, where $x_i \in \mathbb{R}^n$ is state vector and $u_i \in \mathbb{R}^m$ is the control input vector. The agent can switch from one mode to the another mode via discrete transitions, and in each mode, the control law may be different. When the agent computes its reachable set, the only information it needs are its current set of states $x_i(t^i)$ and the current control input $u_i(t^i)$. It should be clarified that although the control law may be different among modes, the control signal $u_i$ is updated with the same control period $T_c^i$. Consequently, $u_i$ is a constant vector in each control period.

Assuming that the agent's current time is $t_j^i = j \times T_c$, using its local sensors and GPS, we have the current state of the agent $x_i$. Note that the local sensors and the provided GPS can only provide the information of interest to some accuracy, therefore the actual state of the agent is in a set $x_i \in I_i$. The control signal $u_i$ is computed based on the state $x_i$ and a reference signal, e.g., a set point denoting where the agent needs to go to, and then computed control signal is applied to the actuator to control the motion of the agent. From the current set of states $I_i$ and the control signal $u_i$, we can compute the forward reachable set of the agent for the next $t_j^i + T$ seconds. This reachable set computation needs to be completed after an amount of time $T_{runtime}^i < T_c^i$ because if $T_{runtime}^i \geq T_c^i$, a new $u_i$ will be updated. The control period $T_c^i$ is chosen based on the agent's motion dynamics, and thus to control an agent with fast dynamics, the control period $T_c^i$ needs to be sufficiently small. This is the source of the requirement that the allowable run-time for reachable set computation be small.

To compute the reachable set of an agent in real-time, we use the well-known face-lifting method [6, 9] and a *hyper-rectangle* to represent the reachable set. This method is useful for short-time reachability analysis of real-time systems. It allows users to define an allowable run-time $T_{runtime}^i$, and has no dynamic data structures, recursion, and does not depend on complex external libraries as in other reachability analysis methods. More importantly, the accuracy of the reachable set computation can be iteratively improved based on the *remaining allowable run-time*.

Algorithm 3.1 describes the real-time reachability analysis for one agent. The Algorithm works as follows. The time period $[t^i, t^i + T]$ is divided by $M$ steps. The reach time step is defined by $h_i = T/M$. Using the reach time step and the current set $I_i$, the face-lifting method performs a *single-face-lifting operation*. The results of this step are a new reachable set and a *remaining reach time* $T_{remainReachTime}^i < T$. This step is iteratively called until the reachable set for the whole time period of interest $[t^i, t^i + T]$ is constructed completely, i.e., the remaining reach time is equal to zero. Interestingly, with the reach time step size $h_i$ defined above, the face-lifting algorithm may be finished quickly after an amount of time which is smaller than the allowable run-time $T_{runtime}^i$ specified by user, i.e., there is still an amount of time called remaining run time $T_{remainRunTime}^i < T_{runtime}^i$ that is available for us to recall the face-lifting algorithm with a smaller reach time step size, for example, we can recall the face-lifting algorithm with a new reach time step $h_i/2$. By doing this, the conservativeness of the reachable set can be iteratively improved. The core step of face-lifting method is the single-face-lifting operation. We refer the readers to [6] for further detail. As mentioned earlier, the local safety property of each agent can be verified at run-time simultaneously with the reachable set computation process. Precisely, let $\mathcal{U}_i \triangleq C_i x_i \leq d_i$ be the unsafe region of the $i^{th}$ agent, the agent is said to be safe from $t^i$ to $t^i + t \leq t^i + T$ if $\mathcal{R}_i[t^i, t^i + t] \cap \mathcal{U}_i = \emptyset$. Since the reachable set $\mathcal{R}_i[t^i, t^i + t]$ is given by the face-lifting method at run-time, the local safety verification problem for each agent can be solved at run-time. Since Algorithm 3.1 computes an over-approximation of the reachable set of each agent in a short time interval, it guarantees the soundness of the result as described in the following lemma.

▶ **Lemma 1** ([6, 9]). *The real-time reachability analysis algorithm is sound, i.e., the computed reachable set contains all possible trajectories of agent $A_i$ from $t^i$ to $t^i + T$.*

**Algorithm 3.1** Real-time reachability analysis for agent $A_i$.

**Input**: $I_i$, $u_i$, $t^i$, $T$, $h_i$, $T^i_{runtime}$, $\mathcal{U}_i$
**Output**: $\mathcal{R}_i[t^i, t^i + T]$, $safe = true$ or $safe = uncertain$

**procedure** INITIALIZATION
  $step = h_i$ % Reach time step
  $T_1^i = T^i_{runtime}$ % Remaining run-time
**procedure** REACHABILITY ANALYSIS
  **while** $(T_1^i > 0)$ **do**
    $\mathcal{CR} = I_i$ % Current reachable set
    $safe = true$
    $T_2^i = T$ % Remaining reach time
    **while** $T_2^i > 0$ **do**
      % Do Single Face Lifting
      $\mathcal{R}, T' = SFL(\mathcal{CR}, step, T_2^i, u_i)$
      $\mathcal{CR} = \mathcal{R}$ % Update reach set
      $T_2^i = T'$ % Update remaining reach time
      **if** $(\mathcal{CR} \cap \mathcal{U}_i \neq \emptyset)$ **then**: $safe = uncertain$
      $\mathcal{R}_i[t^i, t^i + T] = \mathcal{CR}$
    % *Update remaining runtime*
    $T_1^i = T_1^i - (A_i.currentTime() - t^i)$
    **if** $T_1^i \leq 0$ **then** break
    **else**
      $step = h_i/2$ % Reduce reach time step
  **return** $\mathcal{R}_i[t^i, t^i + T] = \mathcal{CR}, safe$

## 4 Decentralized Real-Time Collision Verification

Our collision verification scheme is performed based on the exchanged reachable set messages between agents. For every control period $T_c$, each agent executes the real-time reachability analysis algorithm to check if it is locally safe and to obtain its current reachable set with respect to its current control input. When the current reachable set is available, the agent encodes the reachable set in a message and then broadcasts this message to its cooperative agents and listens to the upcoming messages sent from these agents. When a reachable set message arrives, the agent immediately decodes the message to construct the current reachable set of the sender and then performs peer-to-peer collision detection. The process of computing, encoding, transferring, decoding of the reachable set along with collision checking is illustrated in Figure 1 based on the agent's local clock.

Let $t^i_{rs}$, $t^i_e$, $t^i_{tf}$, $t^i_d$, and $t^i_c$ respectively be the instants at which we compute, encode, transfer, decode the reachable set and do collision checking on the agent $A_i$. Note that these time instants are based on the agent $A_i$'s local clock. The actual run-times are defined as follows.

$\tau^i_{rs} = t^i_e - t^i_{rs}$, % *reachable set computation time*,
$\tau^i_e = t^i_{tf} - t^i_e$, % *encoding time*,
$\tau^i_{tf} \approx t^j_d - t^i_{tf}$, % *transferring time*,
$\tau^i_d = t^i_c - t^i_d$, % *decoding time*.

Note that we do not know the exact transfer time $\tau^i_{tf}$ since it depends on two different local time clocks. The above transfer time formula describes its approximate value when neglecting the mismatch between the two local clocks. The actual reachable set computation time is close to the

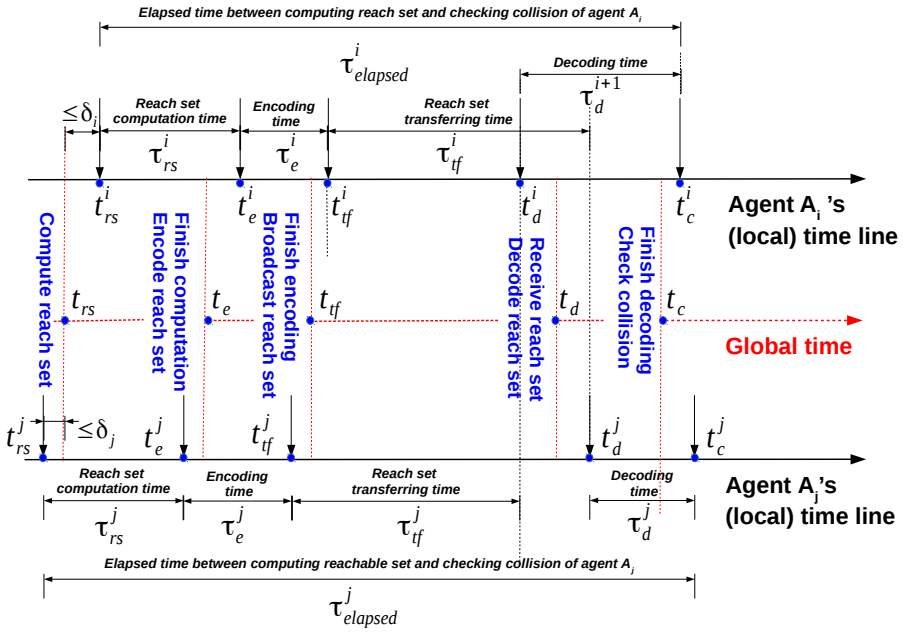


**Figure 1** Timeline for reachable set computing, encoding, transferring, decoding and collision checking. The timeline for these core steps in verification is plotted in parallel with the virtual global time under the assumption that the agent's clock is synchronized with the global time within an error between $-\delta_*$ and $\delta_*$.

allowable run-time chosen by user, i.e., $\tau^i_{rs} \approx T^i_{runtime}$. We will see later that the encoding time and decoding time are fairly small in comparison with the transferring time, i.e., $\tau^i_e \approx \tau^i_d \ll \tau^i_{tf}$. All of these run-times provide useful information for selecting an appropriate control period $T_c$ for an agent. However, for collision checking purposes, we only need to consider the time instants that an agent starts computing reachable set $t^i_{rs}$ and collision checking $t^i_c$.

A reachable set message contains three pieces of information: the reachable set which is a list of intervals, the time period (based on the local clock) in which this reachable set is valid, i.e., the start time $t^i_{rs}$ and the end time $t^i_{rs} + T$ and the time instant that this message is sent. Based on the timing information of the reachable set and the time-synchronization errors, an agent can examine whether or not a received reachable set contains information about the future behavior of the sent agent which is useful for collision checking. The usefulness of the reachable sets used in collision checking is defined as follows.

▶ **Definition 2** (Useful reachable sets)**.** Let $\delta_i$ and $\delta_j$ respectively be the time-synchronization errors of agent $A_i$ and $A_j$ in comparison with the *virtual global time* $t$, i.e, $t - \delta_i \le t^i \le t + \delta_i$ and $t - \delta_j \le t^j \le t + \delta_j$, where $t^i$ and $t^j$ are current local times of $A_i$ and $A_j$ respectively. The reachable sets $\mathcal{R}_i[t^i_{rs}, t^i_{rs} + T]$ and $\mathcal{R}_j[t^j_{rs}, t^j_{rs} + T]$ of the agent $A_j$ that are available at the agent $A_i$ at time $t^i_c$ are *useful* for collision checking between $A_i$ and $A_j$ if:

$$\begin{aligned} &t^i_c < t^j_{rs} + T - \delta_i - \delta_j, \\ &t^i_c < t^i_{rs} + T. \end{aligned} \tag{1}$$

Assume that we are at a time instant where the agent $A_i$ checks if a collision occurs. This means that the current local time is $t^i_c$. Note that agent $A_i$ and $A_j$ are synchronized to the global time with errors $\delta_i$ and $\delta_j$ respectively. The reachable set $\mathcal{R}_j[t^j_{rs}, t^j_{rs} + T]$ is useful if it contains information about the *future behavior* of agent $A_j$ under the view of the agent $A_i$ based on its

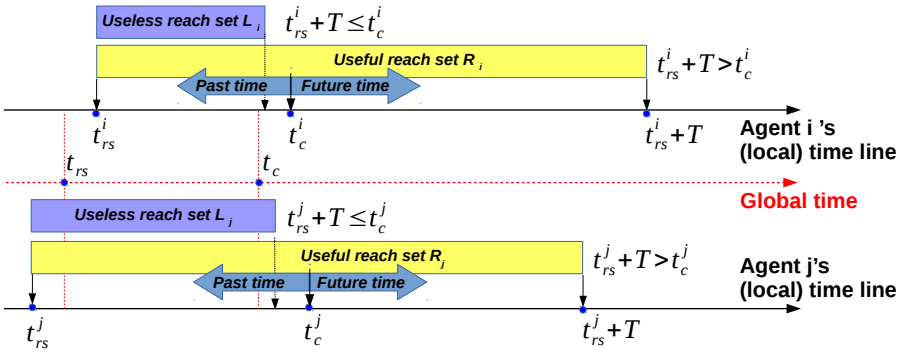


**Figure 2** Useful reachable sets. An exchanged reachable set is useful for real-time verification if and only if it contains the estimation of all possible trajectories of an agent in a time period in the future.

**Algorithm 4.2** Decentralized Real-Time Collision Verification at Agent $A_i$.

**Input**: $l$, % safe distance between agents
**Output**: $collision, T_{safe}$ % collision flag and safe time interval in the future

**procedure** PEER-TO-PEER COLLISION DETECTION
  **if** new message $\mathcal{R}_j[t^j_{rs}, t^j_{rs}+T]$ arrive **then**
    decode message
    $t^i_c = A_i.current\_time()$ % current time
    $t^i_{rs} = \mathcal{R}_i.t^i_{rs}$ % current reachable set start time
    **if** $t^i_c < t^j_{rs} + T - \delta_i - \delta_j$ and $t^i_c < t^i_{rs} + T$ **then** % check usefulness
      compute possible minimum distance $d_{min}$ between two agents
      **if** $d_{min} > l$ **then**
        Collision = false
        $T_{safe} = min(t^j_{rs} + T - \delta_i - \delta_j, t^i_{rs} + T)$
      **else**
        Collision = uncertain, $T_{safe} = [\,]$
      store the message

local clock. This can be guaranteed if we have: $t^j_{rs} + T \geq t^i_{rs} - \delta_j + T > t^i_c + \delta_i$. Additionally, the current reachablet set of agent $A_i$ contains information about its future behavior if $t^i_c < t^i_{rs} + T$ as depicted in Figure 2. We can see that if $t^i_c > t^j_{rs} + T + \delta_i + \delta_j$, then the reachable set of $A_j$ contains a past information, and thus it is useless for collision checking. One interesting case is when $t^j_{rs} + T - \delta_i - \delta_j < t^i_c < t^j_{rs} + T + \delta_i + \delta_j$. In this case, we do not know whether the received reachable set is useful or not.

▶ Remark. We note that the proposed approach does not rely on the concept of Lamport's happens-before relation [22] to compute the local reachable set of each agent. If the agent could not receive reachable messages from others until a requested time-stamp expires, it still calculates the local reachable set based on its current state and the state information of other agents in the messages it received previously. In other words, our method does not require the reachable set of each agent to be computed corresponding to the ordering of the events (sending or receiving a message) in the system, but only relies on the local clock period and the time-synchronization errors between agents. Such implementation ensures that the computation process can be accomplished in real-time, and is not affected by the message transmission delay.

The peer-to-peer collision checking procedure depicted in Algorithm 4.2 works as follows: when a new reachable set message arrives, the receiving agent decodes the message and checks the usefulness of the received reachable set and its current reachable set. Then, the agent combines

its current reachable set and the received reachable set to compute the minimum possible distance between two agents. If the distance is larger than an allowable threshold $l$, there is no collision between two agents in some known time interval in the future, i.e., $T_{safe}$.

▶ **Lemma 3.** *The decentralized real-time collision verification algorithm is sound.*

**Proof.** From Lemma 1, we know that the received reachable set $\mathcal{R}_j[t^j_{rs}, t^j_{rs} + T]$ contains all possible trajectories of the agent $A_j$ from $t^j_{rs}$ to $t^j_{rs} + T$. Also, the current reachable set of the agent $A_i$, $\mathcal{R}_i[t^i_{rs}, t^i_{rs} + T]$, contains all possible trajectories of the agent from $t^i_{rs}$ to $t^i_{rs} + T$. If those reachable sets are useful, then they contains all possible trajectories of two agents from $t^i_c$ to sometime $T_{safe} = min(t^j_{rs} + T - \delta_i - \delta_j, t^i_{rs} + T)$ in the future based on the agent $A_i$ clock. Therefore, the minimum distance $d_{min}$ between two agents computed from two reachable sets is the smallest distance among all possible distances in the time interval $[t^i_c, T_{safe}]$. Consequently, the collision free guarantee is sound in the time interval $[t^i_c, T_{safe}]$. ◀

We have studied how to use exchanged reachable sets to do peer-to-peer collision detection. Next, we consider how to verify online the global behavior of a distributed CPS in decentralized manner.

## 5 Decentralized Real-Time Global Safety Verification

▶ **Definition 4** (Globally useful reachable set.)**.** Consider a distributed CPS with $N$ agents with time synchronization errors $\delta_i, i = 1, 2, \ldots, N$, a globally useful reachable set of the whole system under the view of agent $A_i$ based on its current local time clock $t^i_c$ is defined below:

$$globalReach = \bigwedge_{i=1}^{N} \mathcal{R}_i[t^i_{rs}, t^i_{rs} + T] \wedge \mathcal{T},$$
$$\mathcal{T} \triangleq (t^i_c \leq t \leq T + min\{t^i_{rs} - \delta_i - \delta_j\}, j \neq i, 1 \leq j \leq N). \quad (2)$$

For any time $t$ such that $t^i_c \leq t \leq T + min\{t^i_{rs} - \delta_i - \delta_j\}$ for $\forall\ 1 \leq j \leq N, i \neq j$, we have $\mathcal{R}_i(t) \subseteq \mathcal{R}_i[t^i_{rs}, t^i_{rs} + T], \forall i$. In other words, *globalReach* contains all possible trajectories of all agents from the current local time $t^i_c$ of agent $A_i$ to the future time defined by $T + min\{t^i_{rs} - \delta_i - \delta_j\}, j \neq i, 1 \leq j \leq N$. The globally useful reachable set is a collection of all useful reachable sets (defined in the previous section) received and decoded at an agent $A_i$ under its current local clock $t^i_c$. The inner intersection determines that at the time that all reachable sets have been received and decoded at the agent $A_i$, i.e., $t^i_c$, only a portion of each received reachable set $\mathcal{R}_j[t^j_{rs}, t^j_{rs} + T]$ between $[t^i_c, T + min\{t^i_{rs} - \delta_i - \delta_j]\}$ is useful for checking collision.

It should be noted that to construct a global reachable set, an agent needs to wait for all messages arrive and then decodes all these messages. This process may have an expensive computation cost, especially when the number of agents increases. Since this global reachable set is only valid in an interval of time, the amount of time that is available for verify the global property may be small and not enough for the agent to perform the global safety verification. Having additional hardware for handling in parallel the processes of receiving/decoding messages is a good solution to overcome this challenge.

Using the globally useful reachable set, the global safety verification problem is equivalent to checking whether the globally useful reachable set intersects with the global unsafe region defined by $\mathcal{U} \triangleq Cx \leq d$, where $x = [x_1^T, x_2^T, \cdots, x_N^T]^T$ and $x_i$ is the state vector of agent $A_i$. The procedure for global safety verification is summarized in Algorithm 5.3.

▶ **Lemma 5.** *The decentralized real-time global safety verification algorithm is sound.*

**Algorithm 5.3** Decentralized Real-Time Global Safety Verification for Agent $A_i$.

**Input**: $\mathcal{U}$, % global unsafe constraints
**Output**: $global\_safe, T_{global\_safe}$ % global safe flag and safe time interval in the future

1: **procedure** INITIALIZATION
2: $global\_safe = true$ % global safety flag
3: **procedure** GLOBAL SAFETY VERIFICATION
4: **if** all useful messages are available **then**
5: $t_c^i = A_i.current\_time()$
6: recheck if all messages are still useful
7: construct globally useful reach set $globalReach$
8: **if** $(globalReach \cap \mathcal{U} \neq \emptyset)$ **then**
9: $global\_safe = uncertain$
10: $T_{global\_safe} = [\,]$
11: **else**
12: $global\_safe = true$
13: $T_{global\_safe} = T + min\{t_{rs}^i - \delta_i - \delta_j\}, j \neq i, 1 \leq j \leq N$

**Proof.** Similar to Lemma 3, the soundness of the verification algorithm is guaranteed because of the soundness of the globally useful reachable set containing all possible trajectories of all agents at any time $t \in \mathcal{T}$, where $\mathcal{T} \triangleq (t_c^i \leq t \leq T + min\{t_{rs}^i - \delta_i - \delta_j\}, j \neq i, 1 \leq j \leq N)$. ◀

# 6 Case study

The decentralized real-time safety verification for distributed CPS proposed in this paper is implemented in Java as a package called *drreach*. This package is currently integrated as a library in StarL, which is a novel platform-independent framework for programming reliable distributed robotics applications on Android [24]. StarL is specifically suitable for controlling a distributed network of robots over WiFi since it provides many useful functions and sophisticated algorithms for distributed applications. In our approach, we use the reliable communication network of StarL which is assumed to be asynchronous and peer-to-peer. There may be message dropouts and transmission delays; however, every message that an agent tries to send is eventually delivered with some time guarantees. All experimental results of our approach are reproducible and available online at: `http://www.verivital.com/rtreach/`.

## 6.1 Experiment setup

We evaluate the proposed approach via a distributed search application using quadcopters[2] in which each quadcopter executes its search mission provided by users as a list of way-points depicted in Figure 3. These quadcopters follow the way-points to search for some specific objects. For safety reasons, they are required to work only in a specific region defined by users. In this case study, the quadcopters are controlled to operate at the same constant altitude. It has been shown from the experiments that the proposed approach is promisingly scalable as it works well for a different number of quadcopters. We choose to present in this section the experimental results for the distributed search application with eight quadcopters.

[2] A video recording is available at: `https://youtu.be/YC_7BChsIf0`

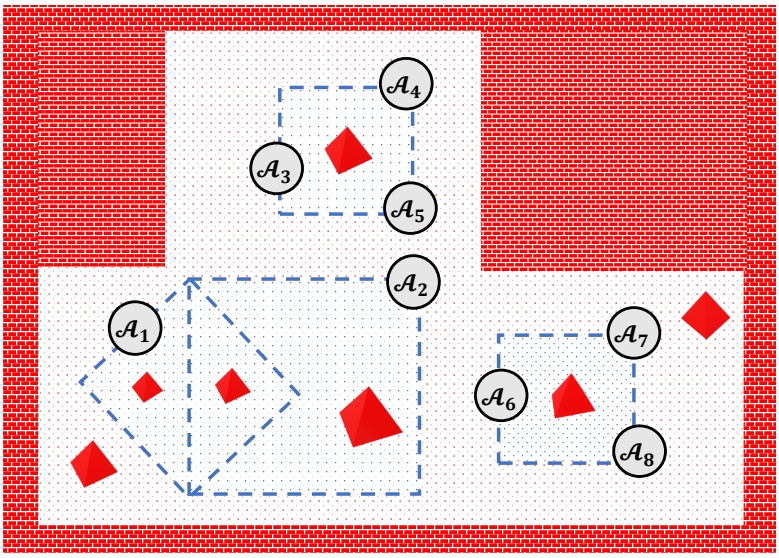


**Figure 3** Distributed Search Application Using Quadcopters.

The first step in our approach is locally computing the reachable set of each quadcopter using the face-lifting method [6,9,18]. The quadcopter has nonlinear motion dynamics given in Equation 3 in which $\theta$, $\phi$, and $\psi$ are the pitch, roll, and yaw angles, $f = \Sigma_{i=1}^{4} T_i$ is the sum of the propeller forces, $m$ is the mass of the quadcopter and $g = 9.81 m/s^2$ is the gravitational acceleration constant. As the quadcopter is set to operate on a constant altitude, we have $\ddot{z} = 0$ which yields the following constraint: $f = \frac{mg}{\cos(\theta)\cos(\phi)}$. Let $v_x$ and $v_y$ be the velocities of a quadcopter along with x- and y- axes. Using the constraint on the total force, the motion dynamics of the quadcopter can be rewritten as a 4-dimensional nonlinear ODE as depicted in Equation 4.

$$\begin{aligned} \ddot{x} &= \frac{f}{m}(\sin(\psi)\sin(\phi) + \cos(\psi)\sin(\theta)\cos(\phi)), \\ \ddot{y} &= \frac{f}{m}(\sin(\psi)\sin(\theta)\cos(\phi) - \sin(\phi)\cos(\psi)), \\ \ddot{z} &= \frac{f}{m}\cos(\theta)\cos(\phi) - g, \end{aligned} \tag{3}$$

$$\begin{aligned} \dot{x} &= v_x, \\ \dot{v}_x &= g\tan(\theta), \\ \dot{y} &= v_y, \\ \dot{v}_y &= g\frac{\tan(\phi)}{\cos(\theta)}. \end{aligned} \tag{4}$$

A PID controller (a proportional-integral-derivative controller widely used in industrial control systems [2]) is designed to control the quadcopter to move from its current position to desired way-points. Details about the controller parameters can be found in the available source code. The PID controller has a control period of $T_c = 200$ milliseconds. In every control period, the control inputs pitch ($\theta$) and roll ($\phi$) are computed based on the current positions of the quadcopter and the current target position (i.e., the current way-point it needs to go). Using the control inputs, the current positions and velocities given from GPS and the motion dynamics of the quadcopter, the real-time reachable set computation algorithm (Algorithm 3.1) is executed *inside* the controller. This algorithm computes the reachable set of a quadcopter from its current local time to the next $T = 2$ seconds. The allowable run-time for this algorithm is $T_{runtime} = 10$ milliseconds. The local safety property is verified by the real-time reachable set computation algorithm at run-time. The computed reachable set is then encoded and sent to another quadcopter. When a reachable set message arrives, the quadcopter decodes the message to reconstruct the current reachable set of the sender. The GPS error is assumed to be 2%. The time-synchronization error between the

quadcopter7 finishes computing reach set and stores the reach set
quadcopter6 finishes computing reach set and stores the reach set
quadcopter5 finishes computing reach set and stores the reach set
quadcopter1 finishes computing reach set and stores the reach set
quadcopter2 encodes its reach set to send out in 0.027134 milliseconds
quadcopter5 encodes its reach set to send out in 0.012657 milliseconds
quadcopter5 broadcasts its reach set to others
quadcopter2 broadcasts its reach set to others
quadcopter7 encodes its reach set to send out in 0.013169 milliseconds
quadcopter7 broadcasts its reach set to others
quadcopter0 encodes its reach set to send out in 0.012709 milliseconds
quadcopter0 broadcasts its reach set to others
quadcopter0 does not violate its local safety property
quadcopter1 encodes its reach set to send out in 0.012707 milliseconds
quadcopter1 broadcasts its reach set to others
quadcopter6 encodes its reach set to send out in 0.011081 milliseconds
quadcopter6 broadcasts its reach set to others
quadcopter1 does not violate its local safety property
quadcopter2 does not violate its local safety property
quadcopter5 does not violate its local safety property
quadcopter7 does not violate its local safety property
quadcopter6 may violates its local safety specification at time 2019-02-17 17:29:51.344
quadcopter7 finishes computing reach set and stores the reach set
quadcopter5 finishes computing reach set and stores the reach set
quadcopter6 finishes computing reach set and stores the reach set
Reach set (hull) of quadcopter0 that is valid from 2019-02-17 17:29:49.075 to 2019-02-17 17:29:51.074 of its local time is:
dim = 0 -> [-263.98, 1034.28]
dim = 1 -> [-329.46, -287.49]
dim = 2 -> [129.36, 301.87]
dim = 3 -> [30.00, 58.48]
Current reach set (hull) of quadcopter1 that is valid from 2019-02-17 17:29:49.386 to 2019-02-17 17:29:51.383 of its local time is:
dim = 0 -> [1959.99, 2040.00]
dim = 1 -> [-0.00, -0.00]
dim = 2 -> [-285.97, 395.76]
dim = 3 -> [-177.55, -149.38]
Current local time of quadcopter1 is 2019-02-17 17:29:49.423
Useful time for checking collision and global safety property for quadcopter1 is 1645 milliseconds
The received reachable set from quadcopter0 is useful
quadcopter1 will not collide with quadcopter0 in the next 1.645 seconds

**Figure 4** A sample of events for verifying the local safety property and collision occurrence.

quadcopters is $\delta = 3$ milliseconds. We want to verify in real-time: 1) local safety property for each quadcopter; 2) collision occurrence; and 3) geospatial free property. The local safety property is defined by $v_x \leq 500$, i.e., the maximum allowable velocities along the x-axis of two arbitrary quadcopters are not larger than $500m/s$. The collision is checked using the minimum allowable distance between two arbitrary quadcopters $d_{min} = 100$. The geospatial free property requires that the some quadcopters never go into a specific region at the same time.

## 6.2 Verifying local safety property and collision occurrence

Figure 4 presents a sample of a sequence of events happening in the distributed search application. One can see that each quadcopter can determine based on its local clocks if there is no collision to some known time in the future. In addition, the local safety property can also be verified at run-time. For example, in the figure, the quadcopter 1 receives a reachable set message from the quadcopter 0 which is valid from $17:29:49.075$ to $17:29:51.074$ of the quadcopter 0's clock. After decoding this message, taking into account the time-synchronization error $\delta$, quadcopter 1 realizes that the received reachable set message is useful for checking collision for the next 1.645 seconds of its clock. After checking collision, quadcopter 1 knows that it will not collide with the quadcopter 0 in the next 1.645 seconds (based on its clock).

It should be noted that we can intuitively verify the collision occurrences by observing the intermediate reachable sets of all quadcopters and their interval hulls. The *intermediate* reachable sets of the quadcopters in every $[0, 2s]$ time interval computed by the real-time reachable set

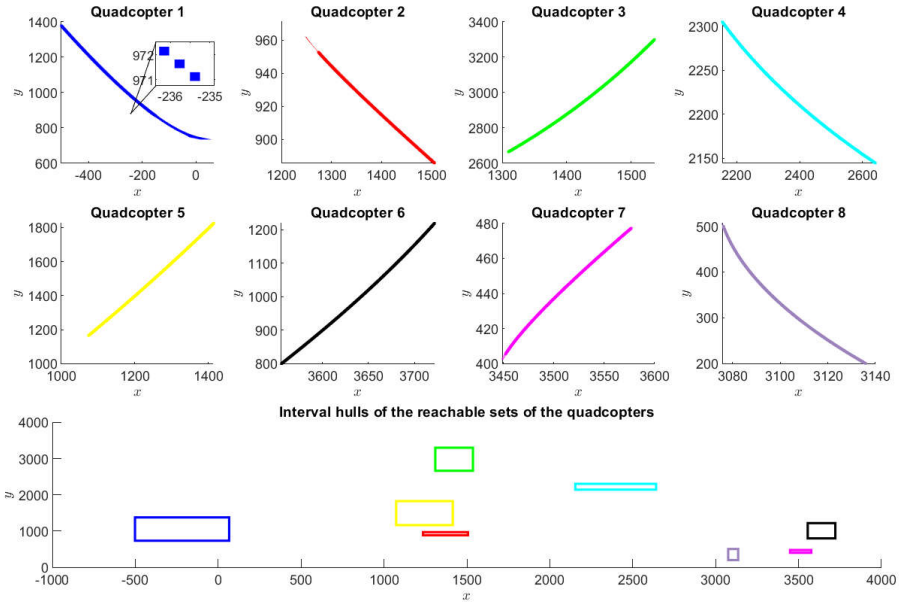


**Figure 5** One sample of the reachable sets of eight quadcopters in an $[0, 2s]$ time interval and their interval hulls.

**Table 1** The average encoding time $\tau_e$, decoding time $\tau_d$, transferring time $\tau_{tf}$, collision checking time $\tau_c$ and total verification time $VT$ of the quadcopters.

| Time | Quad. 1 | Quad. 2 | Quad. 3 | Quad. 4 | Quad. 5 | Quad. 6 | Quad. 7 | Quad. 8 |
|---|---|---|---|---|---|---|---|---|
| *Encoding Time $\tau_e$ (ms)* | 0.058 | 0.055 | 0.0553 | 0.0525 | 0.0557 | 0.0583 | 0.0584 | 0.0597 |
| *Decoding Time $\tau_d$ (ms)* | 0.0169 | 0.0193 | 0.0197 | 0.019 | 0.0210 | 0.0181 | 0.0177 | 0.022 |
| *Transferring Time $\tau_{tf}$ (ms)* | 2.64 | 2.48 | 1.42 | 1.11 | 1.12 | 1.08 | 1.05 | 1.13 |
| *Collision Checking Time $\tau_c$ (ms)* | 0.04 | 0.05 | 0.07 | 0.05 | 0.03 | 0.07 | 0.07 | 0.14 |
| *Total Verification Time $VT$ (ms)* | 28.9363 | 27.9 | 20.6232 | 18.3055 | 18.2527 | 18.235 | 18.0223 | 19.1037 |

computation algorithm (i.e., Algorithm 3.1) is described in Figure 5. The zoom plot within the figure presents a very short-time interval reachable set of the quadcopters. We note that the intermediate reachable set of a quadcopter is represented as a list of hyper-rectangles and is used for verifying the local safety property at run-time. The reachable set that is sent to another quadcopter is the interval hull of these hyper-rectangles. The intermediate reachable set cannot be transferred via a network since it is very large (i.e., hundreds of hyper-rectangles). The interval hull of all hyper-rectangles contained in the intermediate reachable set covers all possible trajectories of a quadcopter in the time interval of $[0, 2s]$. Therefore, it can be used for safety verification. One may question why we use the interval hull instead of using the convex hull of the reachable set since the former one results in a more conservative result. The reason is that the convex hull of hundreds of hyper-rectangles is a time-consuming operation that cannot be used in a real-time setting. Therefore, in the real-time setting, interval hull operation is a suitable solution. From the figure, we can see that the interval hulls of the reachable set of all quadcopters do not intersect with each other. Therefore, there is no collision occurrence (in the next 2 seconds of global time).

Since we implement the decentralized real-time safety verification algorithm inside the quadcopter's controller, it is important to analyze whether or not the verification procedure affects the control performance of the controller. To reason about this, we measure the average encoding,

decoding, transferring and collision checking times for all quadcopters using 100 samples which are presented in Table 1. We note that the transferring time $\tau_{tf}$ is the average time for one message transferred from other quadcopters to the $i^{th}$ quadcopter. It can be seen that the encoding, decoding and collision checking times at each quadcopter constitute a tiny amount of time. The total verification time is the sum of the reachable set computation, encoding, transferring, decoding and collision checking times. Note that the allowable runtime for reachable set computation algorithm is specified by users as $T_{runtime} = 10$ milliseconds. Therefore, the (average) total time for the safety verification procedure on each quadcopter is

$$VT_i = T_{runtime} + \tau_e^i + (N-1) \times (\tau_{tf}^i + \tau_d^i + \tau_c^i), \tag{5}$$

where $i = 1, 2, \ldots, N$, and $N$ is the number of quadcopters. As shown in Table 1, the (average) total verification time for each quadcopter is small ($< 30$ milliseconds), compared to the control period $T_c = 200$ milliseconds. Besides, from the experiment, we observe that the computation time for the control signal of the PID controller $\tau_{control}^i$ (not presented in the table) is also small, i.e., from 5 to 10 milliseconds. Since $VT_i + \tau_{control}^i < T_c/4 = 50$ milliseconds, we can conclude that the verification procedure does not affect the control performance of the controller.

Interestingly, from the verification time formula (5), we can estimate the range of the number of agents that the decentralized real-time verification procedure can deal with. The idea is that, in each control period $T_c$, after computing the control signal, the remaining time bandwidth $T_c - \tau_{control}$ can be used for verification. Let $\bar{\tau}_e(\underline{\tau}_e)$, $\bar{\tau}_{tf}(\underline{\tau}_{tf})$, $\bar{\tau}_d(\underline{\tau}_d)$, $\bar{\tau}_c(\underline{\tau}_c)$ be the maximum (minimum) encoding, transferring, decoding and collision checking times on a quadcopter, $\bar{\tau}_{control}(\underline{\tau}_{control})$ be the maximum (minimum) control signal computation time for each control period $T_c$, then the number of agents that the decentralized real-time safety verification procedure can deal with (with assumption that the communication network works well) satisfies the following constraint:

$$\frac{T_c - \bar{\tau}_{control} - T_{runtime} - \bar{\tau}_e}{\bar{\tau}_{tf} + \bar{\tau}_d + \bar{\tau}_c} + 1 \leq N \leq \frac{T_c - \underline{\tau}_{control} - T_{runtime} - \underline{\tau}_e}{\underline{\tau}_{tf} + \underline{\tau}_d + \underline{\tau}_c} + 1. \tag{6}$$

Let consider our case study, from the Table, we assume that $\bar{\tau}_e = 0.0597$, $\underline{\tau}_e = 0.0525$, $\bar{\tau}_{tf} = 2.64$, $\underline{\tau}_{tf} = 1.05$, $\bar{\tau}_d = 0.022$, $\underline{\tau}_d = 0.0169$, $\bar{\tau}_c = 0.14$, $\underline{\tau}_c = 0.03$ milliseconds. Also, we assume that $\bar{\tau}_{control} = 10$ and $\underline{\tau}_{control} = 5$ milliseconds. We can theoretically estimate the number of quadcopters that our verification approach can deal with is $64 \leq N \leq 168$.

## 6.3 Verifying the geospatial free property

To illustrate how our approach verifies the global behavior of a distributed CPS, we consider the geospatial free property which requires that the some (or all) quadcopters never go into a specific region at the same time. For simplification, we reconsider the distributed search application with two quadcopters (quad 1 and quad 2) whose forbidden region is defined by $900 < x_0 < 1200 \wedge 900 < x_1 < 1200$. Figure 6 describes a sample of events describing that the quadcopter 2 can verify (based on its local clock) that it will not collide with the quadcopter 1 and the global geospatial free property is guaranteed in the next 1.838 seconds.

# 7 Discussion

**Software architecture.** The current implementation of our approach deploys the safety verifier of each agent inside the controller, and a single thread is used to execute the control and verification tasks. The main drawback of this implementation is that it may decrease the overall performance of the controller and even cause the controller to crash. To prevent that happens, in practice,

```
quadcopter0 computes it reach set from 2019-09-16 17:26:20.957 to 2019-09-16 17:26:22.956
quadcopter0 encodes its reach set to send out in 0.017749 milliseconds
quadcopter0 broadcasts its reach set to others
quadcopter0 may violates its local safety specification at time 2019-09-16 17:26:22.956
quadcopter1 receives reach set (hull) from quadcopter0
Time for transferring this reach set over network is around (not considering clock mismatch) 66
milliseconds
Decoding message from quadcopter0 takes 0.029057 milliseconds
Reach set (hull) of quadcopter0 that is valid from 2019-09-16 17:26:20.957 to 2019-09-16
17:26:22.956 of its local time is:
dim = 0 -> [89.18, 280.21]
dim = 1 -> [39.31, 58.43]
dim = 2 -> [804.58, 1240.06]
dim = 3 -> [91.37, 126.26]
Current reach set (hull) of quadcopter1 that is valid from 2019-09-16 17:26:20.119 to 2019-09-16
17:26:22.118 of its local time is:
dim = 0 -> [1946.28, 2060.66]
dim = 1 -> [25.17, 33.29]
dim = 2 -> [787.92, 1566.04]
dim = 3 -> [184.97, 203.86]
Current local time of quadcopter1 is 2019-09-16 17:26:21.112
Useful time for checking collision and global safety property is 1838 milliseconds
The received reachable set from quadcopter0 is useful
quadcopter1 will not collide with quadcopter0 in the next 1.838 seconds
The geospatial free property is guarantee in the next 1.838 seconds
```

**Figure 6** A sample of events for verifying the geospatial free property.

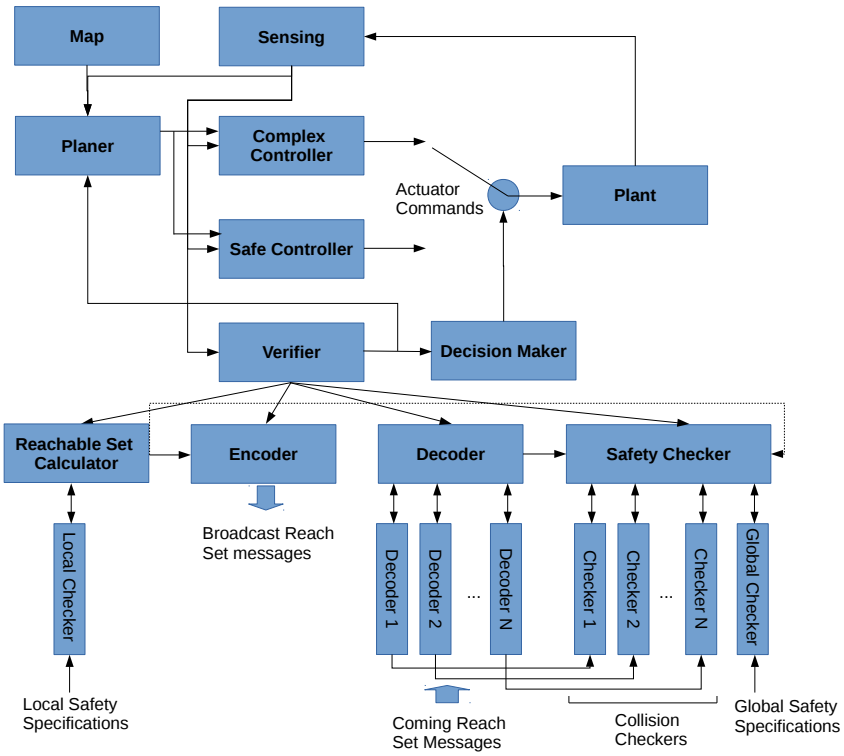


**Figure 7** Software architecture for deploying decentralized real-time safety verification approach on a real platform.

the controller and verifier should be implemented in two separate software components. In this case, the computation burden for safety checks in the verifier does not affect the performance of the controller. The control task and the verification task can be executed efficiently in parallel as depicted in Figure 7. More importantly, this software architecture adopts the architecture of a fault-tolerant system [15] to prevent the propagation of failure from one component to others. It also benefits the use of a simplex-architecture for safety control in the case of dangerous circumstances.

As shown in Figure 7, the verifier component consists of four sub-components including a reachable set calculator, a encoder, a decoder, and a safety checker. These sub-components should also be implemented conveniently for parallel execution. The local safety property is verified inside the reachable set calculator at runtime. As the number of reachable set messages that need to be decoded increases with the number of participating agents, it is necessary to have multiple decoders working in parallel. These decoders listen to upcoming reachable set messages on different ports assigned to them by the verifier and immediately decode any arrived message. This parallel decoding helps to reduce the decoding time significantly. The decoded reachable sets are then sent to the safety checker containing multiple checkers run in parallel in which each checker is responsible for checking collision between the agent with another. The $i^{th}$ checker and the $i^{th}$ decoder is a pair worker, i.e., the checker only waits for the decoded reachable set of its corresponding co-worker. Therefore, the pair to pair collision detection task can be done very quickly. The safety checker also has a global checker which is responsible for checking global properties. The global checker is only triggered when the decoder component finishes decoding all arrived reachable set messages. For this reason, having parallel working decoders is essential to speed up the overall verification time which is required to be very small to work in the real-time setting.

Let $\bar{\tau}_{rs}, \bar{\tau}_e, \bar{\tau}_{tf}$ and $\bar{\tau}_d$ respectively be the worst case times of reachable set computation, encoding, transferring and decoding, $\bar{\tau}_{cc}$ and $\bar{\tau}_{gc}$ be the worst case times of peer-to-peer collision detection and global safety verification. For a system with $N$ agents, the total worst-case verification time is $\bar{\tau}_{total} = \bar{\tau}_{rs} + \bar{\tau}_e + \bar{\tau}_{tf} + \bar{\tau}_d + \bar{\tau}_{cc} + \bar{\tau}_{gc}$. If we do the verification in *sequential way*, i.e., using only one port for reachable set communication and one checker for all peer-to-peer collision detection and global safety verification, the total worst-case verification is: $\bar{\tau}^*_{total} = \bar{\tau}_{rs} + \bar{\tau}_e + \bar{\tau}_{tf} + N\bar{\tau}_d + N\bar{\tau}_{cc} + \bar{\tau}_{gc} >> \bar{\tau}_{total}$.

**Scalability.** From the above discussion, one can see that the software architecture plays an important role when we implement our approach in a real platform. In practice, if each participating agent has the powerful hardware for communication and computation, and the software for our approach is implemented in a parallel manner as proposed above, then the worst-case verification time does not depend on the number of agents in the system. Therefore, our decentralized real-time safety verification approach is scalable for systems with a large number of agents. Also, the proposed software architecture is especially useful in the case that there are losses of reachable set messages. In this hazardous situation, the agent still has some partial information to check if a collision occurs based on the available, reachable set messages. Therefore, the planner still can re-perform path planning algorithm based on the current information and past information it has to find the safest path for the agent for this incomplete information situation.

**Effect of time synchronization error.** The time synchronization error directly affects the ability to receive useful reachable sets and globally useful reachable sets for verification. If the time synchronization error is too large, all exchanged reachable sets may not be useful for verification. In this case, we cannot verify the collision avoidance and global safety properties. For example, in

the definition of the globally useful reachable set, it is required that after all exchanged reachable sets have been received and decoded at an agent $A_i$ at time local current time $t_c^i$, each received reachable set is only useful if and only if $T + min\{t_{rs}^i - \delta_i - \delta_j\} > t_c^i$. One can see that if the time synchronization error is too large, then the related requirement cannot be satisfied. Therefore, the maximum tolerance for the time synchronization error can be estimated roughly as $T + min\{t_{rs}^i - 2\delta_{max}\} >> t_c^i \implies \delta_{max} << (T + t_{rs}^i - t_c^i)/2$.

## 8 Related Work

Our work is inspired by the static and dynamic analysis of timed distributed traces [11] and the real-time reachability analysis for verified simplex design [6]. The former one proposes a sound method of constructing a global reachable set for a distributed CPS based on the recorded traces and time synchronization errors of participating agents. Then the global reachable set is used to verify a global property using Z3 [10]. This method can be considered to be a *centralized analysis* where the reachable set of the whole system is constructed and verified by one analyzer. Such a verification approach is offline which is fundamentally different from our approach as we deal with online verification in a decentralized manner. Our real-time verification method borrows the face-lifting technique developed in [6] and applies it to a distributed CPS.

Another interesting aspect of real-time monitoring for linear systems was recently published in [8]. In this work, the authors proposed an approach that combines offline and online computation to decide if a given plant model has entered an uncontrollable state which is a state that no control strategy can be applied to prevent the plant go to the unsafe region. This method is useful for a single real-time CPS, but not a distributed CPS with multiple agents.

Additionally, there has been other significant works for verifying distributed CPS. Authors of [12, 29, 33] presented a real-time software for distributed CPS but did not perform a safety verification of individual components and a whole system. The works presented in [3,19,21] can be used to verify distributed CPS, but they do not consider a real-time aspect. An interesting work proposed in [26] can formally model and verify a distributed car control system against several safety objectives such as collision avoidance for an arbitrary number of cars. However, it does not address the verification problem of distributed CPS in a real-time manner. The novelty of our approach is that it can over-approximate of the reachable set of each agent whose dynamics are non-linear with a high precision degree in real-time.

The most related work to our scheme was recently introduced in [25]. The authors proposed an online verification using reachability analysis that can guarantee safe motion of mobile robots with respective to walking pedestrians modeled as hybrid systems. This work utilizes CORA toolbox [1] to perform reachability analysis while our work uses a face-lifting technique. However, this work does not consider the time-elapse for encoding, transferring and decoding the reachable set messages between each agent, which play an important role in distributed systems.

## 9 Conclusion and Future Work

We have proposed a decentralized real-time safety verification method for distributed cyber-physical systems. By utilizing the timing information and the reachable set information from exchanged reachable set messages, a sound guarantee about the safety of the whole system is obtained for each participant based on its local time. Our method has been successfully applied for a distributed search application using quadcopters built upon the StarL framework. The main benefit of our approach is that it allows participants to take advantages of formal guarantees available locally in real-time to perform intelligent actions in dangerous situations. This work is a

fundamental step in dealing with real-time safe motion/path planing for distributed robots. For future work, we seek to deploy this method on a distributed autonomous driving testbed using the F1Tenth racing platform [28] and extend it to distributed CPS with heterogeneous agents where the agents can have different motion dynamics and thus they have different control periods. In addition, the scalability of the proposed method can be improved by exploiting the benefit of parallel processing, i.e., each agent handles multiple reachable set messages and checks for collisions in parallel.

## References


1 Matthias Althoff. An introduction to cora 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.

2 Karl Johan Aström, Tore Hägglund, and Karl J Astrom. *Advanced PID control*, volume 461. ISA-The Instrumentation, Systems, and Automation Society, 2006.

3 Kyungmin Bae, Joshua Krisiloff, José Meseguer, and Peter Csaba Ölveczky. Designing and verifying distributed cyber-physical systems using multirate pals: An airplane turning control system case study. *Science of Computer Programming*, 103:13–50, 2015.

4 Stanley Bak and Parasara Sridhar Duggirala. Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, pages 173–178. ACM, 2017.

5 Stanley Bak and Parasara Sridhar Duggirala. Simulation-equivalent reachability of large linear systems with inputs. In *International Conference on Computer Aided Verification*, pages 401–420. Springer, 2017.

6 Stanley Bak, Taylor T Johnson, Marco Caccamo, and Lui Sha. Real-time reachability for verified simplex design. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 138–148. IEEE, 2014.

7 Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for nonlinear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.

8 Xin Chen and Sriram Sankaranarayanan. Model predictive real-time monitoring of linear systems. In *Real-Time Systems Symposium (RTSS), 2017 IEEE*, pages 297–306. IEEE, 2017.

9 Thao Dang and Oded Maler. Reachability analysis via face lifting. In *Hybrid Systems: Computation and Control (HSCC '98)*, pages 96–109. Springer, 1998. LNCS 1386.

10 Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

11 Parasara Sridhar Duggirala, Taylor T Johnson, Adam Zimmerman, and Sayan Mitra. Static and dynamic analysis of timed distributed traces. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 173–182. IEEE, 2012.

12 John C Eidson, Edward A Lee, Slobodan Matic, Sanjit A Seshia, and Jia Zou. Distributed real-time software for cyber–physical systems. *Proceedings of the IEEE*, 100(1):45–59, 2012.

13 Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *Computer Aided Verification*, pages 379–395. Springer, 2011.

14 Antoine Girard, Colas Le Guernic, and Oded Maler. Efficient computation of reachable sets of linear time-invariant systems with inputs. In *Hybrid Systems: Computation and Control*, pages 257–271. Springer, 2006.

15 Alwyn E Goodloe and Lee Pike. Monitoring distributed real-time systems: A survey and future directions, 2010.

16 T. A. Henzinger. The theory of hybrid automata. In *IEEE Symposium on Logic in Computer Science (LICS)*, page 278, Washington, DC, USA, 1996. IEEE Computer Society.

17 Thomas A Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: A model checker for hybrid systems. In *Computer aided verification*, pages 460–463. Springer, 1997.

18 Taylor T. Johnson, Stanley Bak, Marco Caccamo, and Lui Sha. Real-time reachability for verified simplex design. *ACM Trans. Embed. Comput. Syst.*, 15(2), February 2016. doi:10.1145/2723871.

19 Taylor T Johnson and Sayan Mitra. Parametrized verification of distributed cyber-physical systems: An aircraft landing protocol case study. In *Cyber-Physical Systems (ICCPS), 2012 IEEE/ACM Third International Conference on*, pages 161–170. IEEE, 2012.

20 Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. *dReach: δ-Reachability Analysis for Hybrid Systems*, pages 200–205. Springer, 2015.

21 Pratyush Kumar, Dip Goswami, Samarjit Chakraborty, Anuradha Annaswamy, Kai Lampka, and Lothar Thiele. A hybrid approach to cyber-physical systems verification. In *Proceedings of the 49th Annual Design Automation Conference*, pages 688–696. ACM, 2012.

22 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

23 Colas Le Guernic and Antoine Girard. Reachability analysis of hybrid systems using support functions. In *Computer Aided Verification*, pages 540–554. Springer, 2009.

24 Yixiao Lin and Sayan Mitra. Starl: Towards a unified framework for programming, simulating and verifying distributed robotic systems. *CoRR*, abs/1502.06286, 2015. arXiv:1502.06286.

25 Stefan B Liu, Hendrik Roehm, Christian Heinzemann, Ingo Lütkebohle, Jens Oehlerking, and Matthias Althoff. Provably safe motion of mobile robots

in human environments. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 1351–1357. IEEE, 2017.

**26** Sarah M Loos, André Platzer, and Ligia Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In *International Symposium on Formal Methods*, pages 42–56. Springer, 2011.

**27** Nancy Lynch, Roberto Segala, Frits Vaandrager, and Henri B Weinberg. *Hybrid i/o automata*. Springer, 1996.

**28** Matthew O'Kelly, Hongrui Zheng, Dhruv Karthik, and Rahul Mangharam. F1tenth: An open-source evaluation environment for continuous control and reinforcement learning. *Proceedings of Machine Learning Research*, 123, 2020.

**29** Qinghui Tang, Sandeep KS Gupta, and Georgios Varsamopoulos. A unified methodology for scheduling in distributed cyber-physical systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 11(S2):57, 2012.

**30** Hoang-Dung Tran, Luan Viet Nguyen, Nathaniel Hamilton, Weiming Xiang, and Taylor T. Johnson. Reachability analysis for high-index linear differential algebraic equations (daes). In *17th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'19)*. Springer International Publishing, August 2019.

**31** Hoang-Dung Tran, Luan Viet Nguyen, Patrick Musau, Weiming Xiang, and Taylor T. Johnson. Decentralized real-time safety verification for distributed cyber-physical systems. In Jorge A. Pérez and Nobuko Yoshida, editors, *Formal Techniques for Distributed Objects, Components, and Systems (FORTE'19)*, pages 261–277, Cham, June 2019. Springer International Publishing.

**32** Hoang-Dung Tran, Luan Viet Nguyen, Weiming Xiang, and Taylor T Johnson. Order-reduction abstractions for safety verification of high-dimensional linear systems. *Discrete Event Dynamic Systems*, 27(2):443–461, 2017.

**33** Yuanfang Zhang, Christopher Gill, and Chenyang Lu. Reconfigurable real-time middleware for distributed cyber-physical systems with aperiodic events. In *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on*, pages 581–588. IEEE, 2008.