Team-triggered coordination of networked systems

Cameron Nowzari Jorge Cortés

***Abstract*— This paper proposes an approach for improved methods of performing event- and self-triggered communication and control on networked systems. Current self-triggered strategies are known to be quite conservative whereas event-triggered approaches are costly to implement on distributed systems that rely on wireless communication for information transmission. To overcome these limitations, we propose a novel class of team-triggered coordination laws that combine ideas from event- and self-triggered control, are implementable on networked systems, and maintain desired levels of performance. We characterize the asymptotic convergence properties of team-triggered strategies and show that they perform no worse than self-triggered approaches in terms of required communication. Simulations on a multi-agent formation control problem illustrate our results.**

I. Introduction

A growing field of research is the design and implementation of aperiodic controllers for networked sensors and actuators. This interest is motivated by the emphasis on efficient and autonomous operations. Energy usage is correlated with the rate at which sensors take samples, computers recompute control inputs, and signals are transmitted over a network. Periodically performing these tasks is costly and not always necessary. This paper merges ideas from event- and self-triggered control into a novel approach for real-time implementation of distributed controllers in networked systems.

Literature review: This paper builds on results from discrete-event systems [1], event-triggered control [2], [3], and self-triggered control [4], [5], [6], [7] of sensor and actuator networks. These works trade computation for less communication, sensing, or actuator effort while preserving stability. These ideas have also studied decentralized systems from both event-triggered [8] and self-triggered control perspectives [9]. These works differ from our setup in that there is only one plant to be controlled wirelessly through a sensor/actuator network. An idea to apply event-triggering to decentralized systems with multiple plants is presented in [10]; however, agents require continuous information about each others' states in order to be implemented. In [11], [12] the authors apply self-triggered strategies to perform distributed control while guaranteeing a desired level of performance. The works [13], [14], closer in spirit to the ideas presented in this paper, consider distributed systems in which each subsystem broadcasts information to their neighbors only when certain local events are triggered.

Statement of contributions: We propose a novel scheme for networked systems, termed team-triggered, that combines ideas from event- and self-triggered strategies. The basic idea is that agents make promises to one another about where their future states will be and warn each other when they need to be broken. Agents then use this information to compute the next time they will need updated information from their neighbors. The benefits of the team-triggered scheme are threefold. First, agents do not require continuous information about neighboring agents. This is in contrast to many event-triggered control strategies for distributed systems that rely on continuous information. Second, the amount of wireless communication required by our algorithm can be much less than self-triggered strategies by taking a less conservative approach. Lastly, the system is still guaranteed to preserve the stability properties of the other strategies. We illustrate our results through simulation. For reasons of space, the proofs have been omitted and will appear elsewhere.

Notation: We let $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, and $\mathbb{Z}_{\geq 0}$ denote the sets of real, nonnegative real, and nonnegative integer numbers, respectively. The two-norm of a vector is denoted by $\|\cdot\|_2$. Given $x \in \mathbb{R}^d$ and $\delta \in \mathbb{R}_{\geq 0}$, $\overline{B}(x,\delta)$ denotes the closed ball centered at x with radius δ. Given a set S, we denote by $|S|$ its cardinality. We let $\mathbb{P}^{\text{c}}(S)$, respectively $\mathbb{P}^{\text{cc}}(S)$, denote the collection of compact, respectively, compact and connected, subsets of S. For $S_1, S_2 \subset \mathbb{R}^d$, the Hausdorff distance is

$$d_H(S_1,S_2) = \max\{\sup_{x\in S_1}\inf_{y\in S_2}\|x-y\|_2, \sup_{y\in S_2}\inf_{x\in S_1}\|x-y\|_2\}.$$

The Hausdorff distance is a metric on the set of all non-empty compact subsets of $\mathbb{R}^d$. Given two bounded set-valued functions $C_1, C_2 \in \mathcal{C}^0(I \subset \mathbb{R}; \mathbb{P}^{\text{c}}(\mathbb{R}^d))$, we define the distance between them as $d_{\text{func}}(C_1,C_2) = \sup_{t\in I} d_H(C_1(t),C_2(t))$. An undirected graph $\mathcal{G} = (V,E)$ is a pair consisting of a set of vertices $V = \{1,\dots,N\}$ and a set of edges $E \subset V \times V$ such that if $(i,j) \in E$, then $(j,i) \in E$. The set of neighbors of a vertex i is given by $\mathcal{N}(i) = \{j \in V \mid (i,j) \in E\}$. Given $v \in \prod_{i=1}^N \mathbb{R}^{n_i}$, we let $v^i_{\mathcal{N}} = (v_i, \{v_j\}_{j\in\mathcal{N}(i)})$ denote the components of v that correspond to i and its neighbors.

II. Problem statement

In this paper we consider a distributed control problem carried out over a wireless network. Consider N agents whose communication topology is described by an undirected graph $\mathcal{G}$. The fact that (i,j) belongs to E models the ability of agents i and j to communicate with one another. The set of all agents that i can communicate with is then given by its set of neighbors $\mathcal{N}(i)$ in the graph $\mathcal{G}$. The state of agent $i \in \{1,\dots,N\}$, denoted x_i, belongs to a closed set $\mathcal{X}_i \subset \mathbb{R}^{n_i}$. The network state $x = (x_1,\dots,x_N)$ therefore belongs to $\mathcal{X} = \prod_{i=1}^N \mathcal{X}_i$. By the above discussion, agent i

The authors are with the Department of Mechanical and Aerospace Engineering, University of California, San Diego, CA 92093, USA, {cnowzari,cortes}@ucsd.edu

can access $x^i_{\mathcal{N}}$ when it communicates with its neighbors. We assume each agent has access to its own state at all times.

We consider linear dynamics for each $i \in \{1, \dots, N\}$,

$$\dot{x}_i = f_i(x_i, u_i) = A_i x_i + B_i u_i, \tag{1}$$

with $A_i \in \mathbb{R}^{n_i \times n_i}$, $B_i \in \mathbb{R}^{n_i \times m_i}$, and $u_i \in \mathcal{U}_i$. Here, $\mathcal{U}_i \subset \mathbb{R}^{m_i}$ is a compact set of allowable controls for agent i. We assume that the pair (A_i, B_i) is controllable with controls taking values in $\mathcal{U}_i$. We further assume the existence of a *safe-mode* controller $u_i^{\text{sf}} : \mathcal{X}_i \to \mathcal{U}_i$ such that $f_i(x_i, u_i^{\text{sf}}(x_i)) = 0$, i.e., a controller able to keep agent i's state fixed.

The network goal is to drive the agents' states to some desired closed set $D \subset \mathcal{X}$ that captures different coordination tasks. The scope of this paper is not to design the controller to achieve this, but rather explore various implementations.

Given the agent dynamics (1), the graph $\mathcal{G}$, and the set D, our starting point is the availability of a control law that drives the system asymptotically to D. Formally, let $m = \sum_{i=1}^N m_i$, and assume that a continuous map $u^* : \mathcal{X} \to \mathbb{R}^m$ and a continuously differentiable function $V : \mathcal{X} \to \mathbb{R}$, bounded from below, exist such that for all $i \in \{1, \dots, N\}$ and $x \notin D$,

$$\nabla_i V(x) \left(A_i x_i + B_i u_i^*(x)\right) \leq 0, \tag{2a}$$

$$\sum_{i=1}^N \nabla_i V(x) \left(A_i x_i + B_i u_i^*(x)\right) < 0. \tag{2b}$$

We assume that both the control law u^* and the gradient ∇V are distributed over $\mathcal{G}$ meaning that for each i, the ith component of these objects only depends on $x^i_{\mathcal{N}}$, rather than on x. With a slight abuse of notation, we write $u_i^*(x^i_{\mathcal{N}})$ and $\nabla_i V(x^i_{\mathcal{N}})$ to emphasize this fact when convenient. This means that agent i can compute u_i^* and $\nabla_i V$ with just information about its neighbors in $\mathcal{G}$. We refer to u^* as the continuous communication and control law because it requires exact information at all times to be implemented.

III. Periodic, event-, and self-triggered control

This section reviews existing methods for implementing controllers when continuous information is not available.

A. Periodic communication and control

The easiest way to relax the continuous communication requirement is to use a periodic strategy in which agents communicate with a constant period $T \in \mathbb{R}_{>0}$. In this case, agents only get updated information every T seconds.

Let t_{last} be the last time at which agent i received updated information from its neighbors. The next time it receives information is then $t_{\text{next}} = t_{\text{last}} + T$. In-between updates, it uses the zero-order hold estimate and control

$$\widehat{x}^i_j(t) = x_j(t_{\text{last}}), \qquad u_i^{\text{per}}(t) = u_i^*(\widehat{x}^i_{\mathcal{N}}(t)),$$

for $t \in [t_{\text{last}}, t_{\text{next}})$. The time derivative of the Lyapunov function along the trajectories of (1) with $u = u^{\text{per}}$ is

$$\frac{d}{dt} V(x(t)) = \sum_{i=1}^N \nabla_i V(x^i_{\mathcal{N}}(t)) \left(A_i x_i(t) + B_i u_i^{\text{per}}(t)\right). \tag{3}$$

By definition of u^{per}, at the times when the information available to the agents is exact (i.e., $\widehat{x}^i_{\mathcal{N}}(t_{\text{last}}) = x^i_{\mathcal{N}}(t_{\text{last}})$ for all $i \in \{1, \dots, N\}$), the inequality (2b) implies that $\frac{d}{dt} V(x(t_{\text{last}})) < 0$ if $x(t_{\text{last}}) \notin D$. Then, during the time interval $t \in (t_{\text{last}}, t_{\text{next}})$, as the errors $\widehat{x}^i_j(t) - x_j(t)$ begin to grow, the communication period must be chosen small enough such that (3) remains less than 0 at all times.

A shortcoming of this method is that the period cannot be determined in a distributed way, but requires information about the full network. Furthermore, because the period must be chosen regardless of the current state of the network, it must be small enough to deal with worst-case scenarios.

B. Event-triggered communication and control

Instead of using a fixed period, event-triggered laws do not communicate for new information or update their control laws until it is imperative. Let t_{last} be the last time at which all agents received information from their neighbors. Unlike in the periodic approach, the next time t_{next} at which an update should occur is not known a priori. Until then, agent $i \in \{1, \dots, N\}$ uses the zero-order hold estimate and control

$$\widehat{x}^i_j(t) = x_j(t_{\text{last}}), \qquad u_i^{\text{event}}(t) = u_i^*(\widehat{x}^i_{\mathcal{N}}(t_{\text{last}})),$$

for $t \in [t_{\text{last}}, t_{\text{next}})$. The time t_{next} at which an update becomes necessary is determined by the first time after t_{last} when the time derivative of V along the trajectory of (1) with $u = u^{\text{event}}$ is no longer negative. Formally, the event for when agents should request updated information is

$$\sum_{i=1}^N \nabla_i V(x(t_{\text{next}})) \left(A_i x_i(t_{\text{next}}) + B_i u_i^{\text{event}}(t_{\text{last}})\right) = 0. \tag{4}$$

Unfortunately, (4) cannot be checked in a distributed way because it requires global information. Instead, we must design events that can be checked locally. Letting t^i_{last} be some time at which agent i receives updated information, $t^i_{\text{next}} \geq t^i_{\text{last}}$ is the first time such that

$$\nabla_i V(x(t^i_{\text{next}})) \left(A_i x_i(t^i_{\text{next}}) + B_i u_i^{\text{event}}(t^i_{\text{last}})\right) = 0. \tag{5}$$

This means that as long as each agent i can ensure the local event (5) has not yet occurred, it is guaranteed that (4) has not yet occurred either. The shortcoming of this approach is that each agent $i \in \{1, \dots, N\}$ needs to have continuous access to information about the state of its neighbors $\mathcal{N}(i)$ in order to evaluate $\nabla_i V(x) = \nabla_i V(x^i_{\mathcal{N}})$ and check condition (5). This makes the event-triggered approach impractical when this information is only available through communication.

C. Self-triggered communication and control

The self-triggered approach seeks to identify criteria that can be checked autonomously by individual agents in order to decide when updated information is necessary. To achieve this, the basic idea is to remove the requirement on continuous availability of information to check the test (5) by providing agents with possibly inexact information about the state of their neighbors. To do so, we introduce the notion of

reachability sets. Given $y \in \mathcal{X}_i$, let $\mathcal{R}_i(s,y)$ be the reachable set of points under (1) starting from y in s seconds,

$$\mathcal{R}_i(s,y) = \{z \in \mathcal{X}_i \mid \exists\, u_i : [0,s] \to \mathcal{U}_i \text{ such that } z = e^{A_i s} y + \int_0^s e^{A_i(s-\tau)} B_i u_i(\tau) d\tau\}.$$

Agents then create sets that are guaranteed to contain their neighbors' states. Let t^i_{last} be the last time at which agent i received state information $x_j(t^i_{\text{last}})$ from its neighbors, then

$$\mathbf{X}^i_j(t, x_j(t^i_{\text{last}})) = \mathcal{R}_j(t - t^i_{\text{last}}, x_j(t^i_{\text{last}})) \subset \mathcal{X}_j \qquad (6)$$

is guaranteed to contain $x_j(t)$ for all $t \geq t^i_{\text{last}}$. We refer to these as *guaranteed sets*. For simplicity, we let $\mathbf{X}^i_j(t) = \mathbf{X}^i_j(t, x_j(t^i_{\text{last}}))$ when the starting state $x_j(t^i_{\text{last}})$ and time t^i_{last} do not need to be emphasized. We denote by $\mathbf{X}^i_{\mathcal{N}}(t) = (x_i(t), \{\mathbf{X}^i_j(t)\}_{j\in\mathcal{N}(i)})$ the guaranteed set information available to an agent i at time t. In between updates, agent i uses the zero-order hold estimate and control

$$\widehat{x}^i_j(t) = x_j(t^i_{\text{last}}), \qquad u^{\text{self}}_i(t) = u^*_i(\widehat{x}^i_{\mathcal{N}}(t^i_{\text{last}})),$$

for $t \in [t^i_{\text{last}}, t^i_{\text{next}})$. At time t^i_{last}, agent i computes the next time $t^i_{\text{next}} \geq t^i_{\text{last}}$ at which information should be acquired via

$$\sup_{y_{\mathcal{N}} \in \mathbf{X}^i_{\mathcal{N}}(t^i_{\text{next}})} \nabla_i V(y_{\mathcal{N}}) \left(A_i x_i(t^i_{\text{next}}) + B_i u^{\text{self}}_i(t^i_{\text{last}})\right) = 0. \quad (7)$$

By (2a), we know at time t^i_{last} that (7) becomes simply

$$\nabla_i V(x^i_{\mathcal{N}} i(t^i_{\text{last}})) \left(A_i x_i(t^i_{\text{last}}) + B_i u^{\text{self}}_i(t^i_{\text{last}})\right) \leq 0.$$

If all agents use this triggering criterium for updating information, it is guaranteed that $\frac{d}{dt}V(x(t)) \leq 0$ at all times.

The condition (7) is appealing because it can be solved by agent i with the information it possesses at time t^i_{last}. Once determined, agent i schedules to request updated information from its neighbors at time t^i_{next}. The problem with the self-triggered approach is that the resulting times are often conservative because the guaranteed sets can grow large quickly as they capture all possible trajectories of neighboring agents. The following section takes a less conservative approach in order to increase this time between updates.

IV. Team-triggered coordination

This section presents the novel team-triggered approach. Agents make promises to their neighbors about their future states and inform them if these promises are violated later (hence the connection with event-triggered control). With the information provided by the promises, each agent computes the next time that an update is required to guarantee the monotonicity of the Lyapunov function V introduced in Section II (hence the connection with self-triggered control).

A. Promises

A *promise* can be either a time-varying set of states (state promise) or controls (control promise) that an agent sends to another agent. Specifically, a state promise that agent j makes to agent i at time t is a set-valued, continuous (with respect to the Hausdorff distance) function $X^i_j[t] \in \mathcal{C}^0([t,\infty); \mathbb{P}^{\text{cc}}(\mathcal{X}_j))$. This means that agent j promises to agent i that its state at any time $t' \geq t$ will satisfy $x_j(t') \in X^i_j[t](t')$. Similarly, a control promise that agent j makes to agent i at time t is conveyed by a set-valued, continuous function $U^i_j[t] \in \mathcal{C}^0([t,\infty); \mathbb{P}^{\text{c}}(\mathcal{U}_j))$. This means that agent j promises to agent i to only use controls $u_j(t') \in U^i_j[t](t')$ for all $t' \geq t$. Given the dynamics of agent j and state $x_j(t)$ at time t, agent i can compute the state promise for $t' \geq t$,

$$X^i_j[t](t') = \{z \in \mathcal{X}_j \mid \exists\, u_j : [t,t'] \to \mathcal{U}_j \text{ with } u_j(s) \in U^i_j[t](s) \text{ for } s \in [t,t'] \text{ such that } z = e^{A_j(t'-t)} x_j(t) + \int_t^{t'} e^{A_j(t'-\tau)} B_j u_j(\tau) d\tau\}. \quad (8)$$

For simplicity, when the time at which the promise is received is not relevant, we use the notation $X^i_j[\cdot]$ and $U^i_j[\cdot]$ or simply X^i_j and U^i_j, respectively. All promise information available to agent $i \in \{1,\dots,N\}$ at some time t is given by $X^i_{\mathcal{N}}[\cdot]_{|[t,\infty)} = (x_{i|[t,\infty)}, \{X^i_j[\cdot]_{|[t,\infty)}\}_{j\in\mathcal{N}(i)}) \in \mathcal{C}^0\left([t,\infty); \prod_{j\in\mathcal{N}(i)\cup\{i\}} \mathbb{P}^{\text{cc}}(\mathcal{X}_j)\right)$. To extract information from this about a specific time t', we use $X^i_{\mathcal{N}}[\cdot](t')$ or simply $X^i_{\mathcal{N}}(t') = (x_i(t'), \{X^i_j[\cdot](t')\}_{j\in\mathcal{N}(i)})$. The generality of the above definitions allow promise sets to be arbitrarily complex. Here, we restrict ourselves to promise sets that can be described with a finite number of parameters.

A *promise rule* is a method to create promises. Formally, a state promise rule for agent $j \in \{1,\dots,N\}$ generated at time t is a continuous (with respect to the distance d_{func} between set-valued functions) map of the form $R^{\text{s}}_j : \mathcal{C}^0\left([t,\infty); \prod_{i\in\mathcal{N}(j)\cup\{j\}} \mathbb{P}^{\text{cc}}(\mathcal{X}_i)\right) \to \mathcal{C}^0\left([t,\infty); \mathbb{P}^{\text{cc}}(\mathcal{X}_j)\right)$. This means that if agent j must send information to agent i at time t, it sends the state promise $X^i_j[t] = R^{\text{s}}_j(X^j_{\mathcal{N}}[\cdot]_{|[t,\infty)})$. A control promise rule for agent $j \in \{1,\dots,N\}$ generated at time t is a continuous map $R^{\text{c}}_j : \mathcal{C}^0\left([t,\infty); \prod_{i\in\mathcal{N}(j)\cup\{j\}} \mathbb{P}^{\text{cc}}(\mathcal{X}_i)\right) \to \mathcal{C}^0\left([t,\infty); \mathbb{P}^{\text{c}}(\mathcal{U}_j)\right)$. This means that when agent j must send information to agent i at time t, it sends the control promise $U^i_j[t] = R^{\text{c}}_j(X^j_{\mathcal{N}}[\cdot]_{|[t,\infty)})$. We make the assumption that, in the absence of communication delays or noise in the state measurements, the promises generated by these rules have the property that $X^i_j[t](t) = \{x_j(t)\}$.

Example IV.1 (Static ball-radius promise rule) Here we describe a simple control promise rule, termed the static ball-radius rule, to create promises that can be described with a finite number of parameters. Given $j \in \{1,\dots,N\}$, a continuous control law $u_j : \prod_{i\in\mathcal{N}(j)\cup\{j\}} \mathbb{P}^{\text{cc}}(\mathcal{X}_i) \to \mathbb{R}^{m_j}$, and $\delta > 0$, the static ball-radius control promise rule for agent j generated at time t is

$$R^{\text{sb}}_j(X^j_{\mathcal{N}}[\cdot]_{|[t,\infty)})(t') = \overline{B}(u_j(X^j_{\mathcal{N}}(t)), \delta) \cap \mathcal{U}_j, \qquad (9)$$

for $t' \geq t$. Note that this promise is a fixed ball of radius δ in the control space $\mathcal{U}_j$ centered at the control signal used at time t. This promise can be sent with two parameters (assuming δ is known by all agents), the state $x_j(t)$ at time t, and the control action $u_j(X^j_{\mathcal{N}}(t))$ at that time. •

B. Controllers on set-valued information models

Here we discuss the type of controllers that the team-triggered approach relies on. The underlying idea is, that since agents possess set-valued information about the state of other agents through promises, controllers themselves should be defined on sets, rather than on points. Our starting point is then a continuous controller $u^{**} : \prod_{j\in\{1,\dots,N\}} \mathbb{P}^{\text{cc}}(\mathcal{X}_j) \to \mathbb{R}^m$ that satisfies, for all $i \in \{1,\dots,N\}$,

$$\nabla_i V(x)\,(A_i x_i + B_i u_i^{**}(\{x\})) \le 0, \tag{10a}$$

$$\sum_{i=1}^{N} \nabla_i V(x)\,(A_i x_i + B_i u_i^{**}(\{x\})) < 0. \tag{10b}$$

In other words, if exact, singleton-valued information is available to the agents, then u^{**} guarantees the monotonic evolution of V. We assume that u^{**} is distributed over $\mathcal{G}$. As before, this means that for each $i \in \{1,\dots,N\}$, the ith component u_i^{**} can be computed with information in $\prod_{j\in\mathcal{N}(i)\cup\{i\}} \mathbb{P}^{\text{cc}}(\mathcal{X}_j)$ rather than in the full space.

Controllers of the form described above can be obtained using a number of design methods. We do not enter into the specific details, but briefly mention how one such controller can be derived from the availability of the controller $u^* : \mathcal{X} \to \mathbb{R}^m$ introduced in Section II. Let $E : \prod_{j=1}^N \mathbb{P}^{\text{cc}}(\mathcal{X}_j) \to \mathcal{X}$ be a continuous map that is distributed over $\mathcal{G}$ and satisfies, for each $i \in \{1,\dots,N\}$, that $E_i(Y) \in Y_i$ for each $Y \in \prod_{j=1}^N \mathbb{P}^{\text{cc}}(\mathcal{X}_j)$ and $E_i(\{y\}) = y_i$ for each $y \in \mathcal{X}$. Let

$$u^{**}(Y) = u^*(E(Y)). \tag{11}$$

Note that this controller satisfies (10a) and (10b) because u^* satisfies (2a) and (2b).

Example IV.2 (Controller definition with the ball-radius promise rule) Here we construct a controller u^{**} using (11) when promises are generated according to the ball-radius control rule, cf. Example IV.1. To do so, note that it is sufficient to define the map $E : \prod_{j=1}^N \mathbb{P}^{\text{cc}}(\mathcal{X}_j) \to \mathcal{X}_j$ only for tuples of sets of the form given in (8), where the corresponding control promise is defined by (9). Define E by

$$\begin{aligned} &E_j(X_1[t](t'),\dots,X_N[t](t')) \\ &= e^{A_j(t'-t)}x_j(t) + \int_t^{t'} e^{A_j(t'-\tau)}B_j u_j(X_{\mathcal{N}}^j(t))d\tau, \end{aligned}$$

which is guaranteed to be in $X_j[t](t')$ for $t' \ge t$. •

C. Self-triggered information updates

Here we discuss how agents use the promises received from other agents to generate self-triggered information requests in the future. Let t^i_{last} be some time at which agent i receives updated information (i.e., promises) from its neighbors. Until the next time information is obtained, agent i has access to the collection of sets $X^i_{\mathcal{N}}$ describing its neighbors' states and can compute its evolution under the controller u^{**} via

$$\begin{aligned} x_i(t) =& e^{A_i(t-t^i_{\text{last}})}x_i(t^i_{\text{last}}) \\ &+ \int_{t^i_{\text{last}}}^{t} e^{A_i(t-\tau)}B_i u_i^{**}(X^i_{\mathcal{N}}(\tau))d\tau, \quad t \ge t^i_{\text{last}}. \end{aligned} \tag{12}$$

Note that this evolution of agent i can be viewed as a promise that it makes to itself, i.e., $X_i^i[\cdot](t) = \{x_i(t)\}$. With this in place, agent i can schedule the next time t^i_{next} at which it will need updated information from its neighbors. To do so, we define, for any $Y_{\mathcal{N}} \in \prod_{j\in\mathcal{N}(i)\cup\{i\}} \mathbb{P}^{\text{cc}}(\mathcal{X}_j)$,

$$\mathcal{L}_i V^{\text{sup}}(Y_{\mathcal{N}}) = \sup_{y_{\mathcal{N}}\in Y_{\mathcal{N}}} \nabla_i V(y_{\mathcal{N}})\,(A_i y_i + B_i u_i^{**}(Y_{\mathcal{N}})), \tag{13}$$

where y_i is the element of $y_{\mathcal{N}}$ corresponding to i. The trigger for when agent i needs new information from its neighbors is similar to (7), where promise sets are used instead of guaranteed ones. Specifically, the time at which information is requested is $t^i_{\text{next}} = \max\{t^i_{\text{last}} + T_{\text{d,self}}, t^*\}$, where $T_{\text{d,self}} > 0$ is an a priori chosen parameter that we discuss below and t^* is implicitly defined as the first time $t^* \ge t^i_{\text{last}}$ such that

$$\mathcal{L}_i V^{\text{sup}}(X^i_{\mathcal{N}}(t^*)) = 0. \tag{14}$$

As long as (14) has not yet occurred for all agents $i \in \{1,\dots,N\}$ for some time t and the promises have not been broken, (10a) and (10b) and the continuity of (13) guarantee

$$\frac{d}{dt}V(x(t)) \le \sum_{i=1}^{N} \mathcal{L}_i V^{\text{sup}}(X^i_{\mathcal{N}}(t)) < 0.$$

The parameter $T_{\text{d,self}} > 0$ is the *self-triggered dwell time*. We introduce it because, in general, it is possible that $t^* = t^i_{\text{last}}$, implying that continuous communication would be required. The dwell time is used to prevent this behavior. Note that $\mathcal{L}_i V^{\text{sup}}(X^i_{\mathcal{N}}(t')) \le 0$ is only guaranteed while $t' \in [t^i_{\text{last}}, t^*]$. Therefore, in case $t^* < t^i_{\text{last}} + T_{\text{d,self}}$, the agent uses the safe-mode control during $t' \in (t^*, t^i_{\text{last}} + T_{\text{d,self}}]$ to leave its state fixed. This design ensures the monotonicity of the evolution of V along the network execution. The team-triggered controller is therefore defined, for $t \in [t^i_{\text{last}}, t^i_{\text{next}})$, by

$$u_i^{\text{team}}(t) = \begin{cases} u_i^{**}(X^i_{\mathcal{N}}(t)), & \text{if } \mathcal{L}_i V^{\text{sup}}(X^i_{\mathcal{N}}(t)) \le 0, \\ u_i^{\text{sf}}(x_i(t)), & \text{if } \mathcal{L}_i V^{\text{sup}}(X^i_{\mathcal{N}}(t)) > 0. \end{cases} \tag{15}$$

D. Event-triggered information updates

The discussion up to this point has assumed that all promises are kept at all times, but this may not be the case. Consider an agent $i \in \{1,\dots,N\}$ that has sent a promise $X_i^j[t_{\text{last}}]$ to a neighboring agent j at some time t_{last}. If agent i ends up breaking its promise at time $t^* > t_{\text{last}}$, i.e., $x_i(t^*) \notin X_i^j[t_{\text{last}}](t^*)$, then it is responsible for sending a new promise $X_i^j[t_{\text{next}}]$ to agent j at time $t_{\text{next}} = \max\{t_{\text{last}} + T_{\text{d,event}}, t^*\}$, where $T_{\text{d,event}} > 0$ is an a priori chosen parameter that we discuss below. This implies that agent i must keep track of promises made to its neighbors and monitor them in case they are broken. This mechanism is implementable since each agent only needs information about its own state and its promises to determine whether the trigger is satisfied.

The parameter $T_{\text{d,event}} > 0$ is the *event-triggered dwell time*. We introduce it because, in general, the time $t^* - t_{\text{last}}$ between when agent i makes and breaks a promise to an agent j might be arbitrarily small. However, to prevent agent j from operating under incorrect information about agent i for $t \in [t^*, t_{\text{last}} + T_{\text{d,event}})$, we introduce a warning message WARN that agent i must send to agent j when it breaks its promise at time $t^* < t_{\text{last}} + T_{\text{d,event}}$. If agent j receives such a warning message, it redefines the promise X_j^i as follows,

$$X_i^j[\cdot](t) = \bigcup_{x_i \in X_i^j[\cdot](t^*)} \mathcal{R}(t - t^*, x_i), \tag{16}$$

for $t \geq t^*$, until the new message arrives at time $t_{\text{next}} = t_{\text{last}} + T_{\text{d,event}}$. By definition of reachable set, the promise $X_i^j[\cdot](t)$ is guaranteed to contain $x_i(t)$ for $t \geq t^*$.

V. Convergence analysis

The combination of the self- and event-triggered information updates described above together with the controller u^{team} defined in (15) yields the TEAM-TRIGGERED LAW, presented in Algorithm 1. We next analyze its convergence properties.

Algorithm 1: TEAM-TRIGGERED LAW

(Self-trigger information update)
At any time t agent $i \in \{1, \dots, N\}$ receives new promise(s) $X_j^i[t]$ from neighbor(s) $j \in \mathcal{N}(i)$, agent i performs:

1: compute own state evolution $x_i(t')$ for $t' \geq t$ using (12)
2: compute first time $t^* \geq t$ such that $\mathcal{L}_i V^{\sup}(X_{\mathcal{N}}^i(t^*)) = 0$
3: schedule information request to neighbors in $\max\{t^* - t, T_{\text{d,self}}\}$ seconds
4: apply controller $u^{\text{team}}(t')$ for $t' \in [t, t + \max\{t^* - t, T_{\text{d,self}}\})$

(Respond to information request)
At any time t a neighbor $j \in \mathcal{N}(i)$ requests information, agent i performs:

1: send new promise $X_i^j[t] = R_i^{\text{s}}(X_{\mathcal{N}}^i[\cdot]_{[t,\infty)})$ to agent j

(Event-trigger information update)
At all times t, agent i performs:

1: **if** there exists $j \in \mathcal{N}(i)$ such that $x_i(t) \notin X_i^j[\cdot](t)$ **then**
2: **if** agent i has sent a promise to j at some time $t_{\text{last}} \in (t - T_{\text{d,event}}, t]$ **then**
3: send warning message WARN to agent j at time t
4: schedule to send new promise $X_i^j[t_{\text{last}} + T_{\text{d,event}}] = R_i^{\text{s}}(X_{\mathcal{N}}^i[\cdot]_{|[t_{\text{last}} + T_{\text{d,event}}, \infty)})$ to agent j in $t_{\text{last}} + T_{\text{d,event}} - t$ seconds
5: **else**
6: send new promise $X_i^j[t] = R_i^{\text{s}}(X_{\mathcal{N}}^i[\cdot]_{|[t,\infty)})$ to agent j at time t
7: **end if**
8: **end if**

(Respond to warning message)
At any time t agent $i \in \{1, \dots, N\}$ receives a warning message WARN from agent $j \in \mathcal{N}(i)$

1: redefine promise set $X_j^i[\cdot](t') = \cup_{x_j \in X_j^i[\cdot](t)} \mathcal{R}_j(t' - t, x_j)$ for $t' \geq t$

Our first result establishes the monotonic evolution of V.

Proposition V.1 *Consider a networked system as described in Section II executing the* TEAM-TRIGGERED LAW *(cf. Algorithm 1). Then, the function V is monotonically non-increasing along the network evolution.*

Next we characterize the convergence properties of coordination strategies designed with the team-triggered approach.

Proposition V.2 *Consider a networked system as described in Section II executing the* TEAM-TRIGGERED LAW *(cf. Algorithm 1) with dwell times $T_{d,self}, T_{d,event} > 0$. Then any bounded network evolution with uniformly bounded promises asymptotically approaches D.*

The next result ensures that continuous communication is never required under the TEAM-TRIGGERED LAW.

Lemma V.3 (Not Zeno) *Consider a networked system as described in Section II executing the* TEAM-TRIGGERED LAW *(cf. Algorithm 1) with dwell times $T_{d,self}, T_{d,event} > 0$. Then the network does not exhibit Zeno behavior.*

VI. Simulations

In this section we present simulations of the team- and self-triggered approaches in a planar multi-agent formation control problem. Our starting point is the distributed coordination algorithm based on graph rigidity analyzed in [15]. Consider 4 agents communicating over the complete graph which seek to attain a rectangle formation of side lengths 1 and 2. The dynamics of each agent is a single integrator $\dot{x}_i = u_i$ for all $i \in \{1, \dots, N\}$, where $\|u_i\|_2 \leq u_{\max} = 50$. The safe-mode controller is then simply $u_i^{\text{sf}} \equiv 0$. The distributed continuous-time controller that makes the network asymptotically achieve the desired formation is given by

$$u_i^*(x) = \sum_{j \in \mathcal{N}(i)} \left(\|x_j - x_i\|_2^2 - d_{ij}^2\right) \text{unit}(x_j - x_i), \tag{17}$$

where d_{ij} is the pre-specified desired distance between agents i and j, and $\text{unit}(v)$ denotes the unit vector in the direction of v. In turn, this controller corresponds to the gradient descent law for $V : \left(\mathbb{R}^2\right)^N \to \mathbb{R}_{\geq 0}$,

$$V(x) = \frac{1}{2} \sum_{(i,j) \in E} \left(\|x_j - x_i\|_2^2 - d_{ij}^2\right)^2,$$

used to establish the correctness of u^*. For the team-triggered approach, the controller u^{team} is defined by (15), where controller u^{**} is given by (11) as described in Example IV.2.

Figure 1 shows the number of communications required in the self-triggered approach and the team-triggered approach using dwell times of $T_{\text{d,self}} = 0.03$ and $T_{\text{d,event}} = 0.0003$ and the static ball-radius promise of Example IV.1 with $\delta = 0.50$. To compare the two strategies, we let N_S^i be the number of times agent i has requested new information and N_E^i be the number of messages an agent i has sent to a neighboring agent because it broke its promise. Given that each agent has 3 neighbors, the total number of messages for an execution is then given by $N_{\text{comm}} = \sum_{i=1}^4 3N_S^i + N_E^i$. Remarkably, the team-triggered approach outperforms the self-triggered approach, both in terms of required communication and time to convergence. Figure 1(a) shows that very quickly all agents are requesting information as often as they can in the self-triggered approach, due to the conservative nature of the request time computations. Figure 1(b), instead, shows a much better behavior in the team-triggered approach.

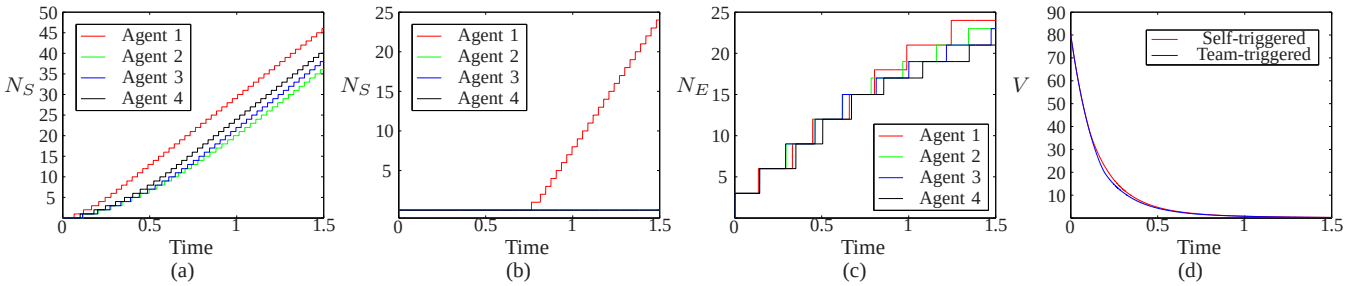

Fig. 1. Plots (a) and (b) show, respectively, the number of self-triggered information requests made by each agent in the self-triggered and team-triggered implementations. Plot (c) shows the number of event-triggered messages sent by each agent in the team-triggered implementation. Plot (d) compares the evolution of the Lyapunov function for both implementations.

Interestingly, only agent 1 requests information due to the self-triggered update times being far less conservative. This means the other agents are only receiving information when promises to them are broken, cf. Figure 1(c).

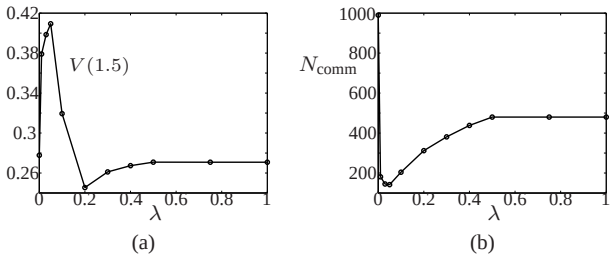

Fig. 2. Plots of (a) the value of the Lyapunov function at a fixed time (1.5 sec) and (b) the total number of messages exchanged in the network by this time for the team-triggered approach with varying tightness of promises λ.

Figure 2 illustrates the role that the tightness of promises has on the network performance. With the notation of Example IV.1, let $\lambda = \frac{\delta}{2}$, so that $\lambda = 0$ corresponds to exact promises and $\lambda = 1$ corresponds to no promises at all (i.e., the self-triggered approach). Figure 2 compares the value of the Lyapunov function after a fixed amount of time (1.5 seconds) and the total number of messages sent N_{comm} between agents by this time for varying tightness of promises. One can observe that a suitable choice of λ optimizes the rate of convergence while still requiring less communication than the self-triggered approach.

VII. Conclusions

We have proposed the team-triggered approach as a result of the combination of event and self-triggered control on networked systems. When information between subsystems must be obtained through wireless communication, it is costly to perform event-triggered communication, and self-triggered communication strategies are typically conservative. The team-triggered approach combines ideas from both event and self-triggered strategies into a useful, implementable strategy that requires less communication than self-triggered communication while maintaining desired levels of performance. The backbone of the team-triggered approach is the quality of promises that agents make to one another. Furthermore, we showed that self-triggered communication is just a specific case of the TEAM-TRIGGERED LAW in which no promises are made. Future work will be devoted to incorporating delays and relaxing the somewhat specific conditions on the Lyapunov function and the safe-mode availability in the dynamics of subsystems.

Acknowledgments

This research was supported by NSF award CCF-0917166.

References

[1] C. G. Cassandras and S. Lafortune, *Introduction to Discrete-Event Systems*. Springer, 2 ed., 2007.

[2] P. Wan and M. D. Lemmon, "Event-triggered distributed optimization in sensor networks," in *Symposium on Information Processing of Sensor Networks*, (San Francisco, CA), pp. 49–60, 2009.

[3] A. Eqtami, D. V. Dimarogonas, and K. J. Kyriakopoulos, "Event-triggered control for discrete-time systems," in *American Control Conference*, (Baltimore, MD), pp. 4719–4724, July 2010.

[4] M. Velasco, P. Marti, and J. M. Fuertes, "The self triggered task model for real-time control systems," in *Proceedings of the 24th IEEE Real-Time Systems Symposium*, pp. 67–70, 2003.

[5] R. Subramanian and F. Fekri, "Sleep scheduling and lifetime maximization in sensor networks," in *Symposium on Information Processing of Sensor Networks*, (New York, NY), pp. 218–225, 2006.

[6] X. Wang and M. D. Lemmon, "Self-triggered feedback control systems with finite-gain L_2 stability," *IEEE Transactions on Automatic Control*, vol. 54, no. 3, pp. 452–467, 2009.

[7] A. Anta and P. Tabuada, "To sample or not to sample: self-triggered control for nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 55, no. 9, pp. 2030–2042, 2010.

[8] M. Mazo Jr. and P. Tabuada, "Decentralized event-triggered control over wireless sensor/actuator networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 10, pp. 2456–2461, 2011.

[9] M. Mazo Jr. and P. Tabuada, "On event-triggered and self-triggered control over sensor/actuator networks," in *IEEE Conf. on Decision and Control*, (Cancun, Mexico), pp. 435–440, 2008.

[10] D. V. Dimarogonas and K. H. Johansson, "Event-triggered control for multi-agent systems," in *IEEE Conf. on Decision and Control*, (Shanghai, China), pp. 7131–7136, 2009.

[11] D. V. Dimarogonas, E. Frazzoli, and K. H. Johansson, "Distributed event-triggered control for multi-agent systems," *IEEE Transactions on Automatic Control*, vol. 57, no. 5, pp. 1291–1297, 2012.

[12] C. Nowzari and J. Cortés, "Self-triggered coordination of robotic networks for optimal deployment," *Automatica*, vol. 48, no. 6, pp. 1077–1087, 2012.

[13] X. Wang and M. D. Lemmon, "Event-triggered broadcasting across distributed networked control systems," in *American Control Conference*, (Seattle, WA), pp. 3139–3144, June 2008.

[14] E. Garcia and P. J. Antsaklis, "Decentralized model-based event-triggered control of networked systems," Tech. Rep. isis-2012-002, University of Notre Dame, Feb. 2012.

[15] L. Krick, M. E. Broucke, and B. Francis, "Stabilization of infinitesimally rigid formations of multi-robot networks," *International Journal of Control*, vol. 82, no. 3, pp. 423–439, 2009.