Fault-tolerant and Distributed Control in Medical Cyber-Physical Systems

Tao Li*, Jiannong Cao†, Qixin Wang*, Shao-jie Tang†, Lei Yang*, Chao Ma*

*Department of Computing, The Hong Kong Polytechnic University, Kowloon, Hong Kong
Email: {cstaoli,csjcao,csqwang,csleiyang,cscma}@comp.polyu.edu.hk
†Department of Computer Science, Illinois Institute of Technology, IL, USA
Email: stang7@hawk.iit.edu

Abstract—Medical cyber-physical systems (MCPS) integrate a number of medical devices to prevent safety hazards as well as improve efficiency of clinical procedures. Most of existing approaches for MCPS control are either in centralized manner or assuming ideal network conditions (e.g., no message loss). However, the high dynamicity of clinical environment calls for distributed control, and also invalidates the ideal network assumption. To fill this gap, we propose a distributed control approach for MCPS in the network environment with asynchronous and unreliable communications. This is very challenging because MCPS are safety-critical and tightly blend a variety of characteristics from different technical domains. Our solution to this problem is based on a modular software design, which not only makes algorithm design easier, but also reduces software complexity to address device manufacturers’ concern on cost and time-to-market. Moreover, we propose detailed algorithms for each functional module, and theoretically prove that those algorithms can tolerate message loss and prevent safety hazards. To evaluate the algorithms, we have conducted extensive simulation based on real-world traces. Simulation results show that while ensuring safety, our algorithms can achieve good system performance.

I. INTRODUCTION

Most medical devices so far in the market have been produced for isolated use, and lack interoperability. To conduct complex clinical procedures which often involve a number of medical devices, clinical staff have to control the devices and collaborate together. Unfortunately, manual control is error-prone. Improper operations, e.g., surgeon forgets to restart ventilator after taking x-ray image during surgery [1], can cause serious consequences to patient or even death. As reported in [2], at least 44,000 people die in hospitals each year as a result of medical errors that are actually preventable.

This situation urgently calls for safe medical device systems that are able to prevent/reduce adverse medical events. To this end, Medical Device Plug-and-Play (MD PnP) Interoperability Program [3] was launched in 2004 to promote medical device interoperability and safe integration. The integrated medical device systems are referred to as medical cyber-physical systems (MCPS) [4], which involve embedded software, networking capabilities, as well as patient’s body. Thus, they are recognized as typical cyber-physical systems (CPS).

To build safe control mechanism for MCPS, most existing work either is based on centralized architecture [5], [6], [7] or assumes ideal network conditions (e.g., no message loss) [1], [8], [9]. However, the clinical environment is quite dynamic.

Patients can be transferred from one place to another (e.g., from operation room to ICU); medical devices can be frequently turned on/off or move around (e.g., hemodialysis machine should be turned on 4 hours per day to clean a patient’s blood). Facing such high dynamicity, centralized integration approach is not flexible enough. Furthermore, communication in such dynamic environment is asynchronous and unreliable. For example, people sometimes trip over the jumbled wires, resulting in disconnection. This problem will become more pronounced in the future as it is a trend that more and more medical devices will adopt wireless technologies. This is driven by the fact that clinical staff and patients prefer wire-free connection [6]. Therefore, ideal network assumption actually does not hold in reality.

To fill this gap, our objective in this paper is to investigate distributed control mechanism for MCPS in such dynamic and unreliable environment. However, we face several challenges. i) MCPS are complex and safety-critical. It involves properties from different domains, such as distributed systems, control theory, real-time systems and clinical medicine [4]. Tight combination of those properties and safety requirements makes distributed algorithm design a non-trivial task. ii) Introducing message loss into MCPS will further complicate algorithm design, because it is necessary to have fault-tolerant solutions to deal with message loss. iii) In contrast with centralized counterpart, distributed approach increases the complexity of software in medical devices. From medical device manufacturers’ point of view, it is crucial to suppress the software complexity, so that development and time cost for their medical device products can be reduced.

To cope with those challenges, we come up with a modular device software design and devise a set of fault-tolerant algorithms. To the best of our knowledge, this is the first work dealing with distributed control of MCPS. Our major contributions in the paper are summarized as follows.

• We propose a modular software design for medical devices. The complex control unit in a device is decomposed into several modules/submodules dealing with specific tasks. This design helps to reduce the software complexity and development cost.
• We analyze the interdependency between medical devices. Therefore, to construct feedback control, a device only needs to establish feedback paths from the devices
that it depends on.

- A set of distributed algorithms are devised for constructing feedback control, open-loop control, and closed-loop control, respectively. Moreover, we theoretically prove that the proposed algorithms can prevent safety hazards.
- We construct a concrete clinical scenario, and conduct extensive simulation based on real-world traces. Simulation results show that our algorithms are resilient to message loss as well as retain nice performance.

The remainder of this paper is organized as follows. Section II describes the system model. The system execution framework and device design are introduced in Section III. Then, Section IV and V present the algorithmic details on feedback control construction, open-loop and closed-loop control. Our approaches are evaluated in Section VI. Related work is discussed in Section VII. Finally, in Section VIII, we conclude the paper and point out the future research directions.

II. SYSTEM MODEL

A MCPS $Sys$ is a 3-tuple: $\langle TS^{d_1}, TS^{d_2}, \ldots, TS^{d_n}, H, R \rangle$. It is composed of a set of medical devices $Dev = \{d_1, d_2, \ldots, d_n\}$ which are interconnected through network. Each constituent medical device $d_i$, $1 \leq i \leq n$ is modeled as a transition system $TS^{d_i}$. $H$ and $R$ are a set of safety hazards that the system $Sys$ has to circumvent and a set of control rules, respectively.

A. Device Model

Medical devices are basically embedded devices [4], so we model a medical device $d$ as a transition system $TS^d = \{M^d, I^d, S^d, T^d, A^d\}$. $M^d$ is a set of device modes (mode for short); $S^d$ is a set of device states (state for short); $I^d$ is the initial device modes; $T^d \subseteq M^d \times M^d$ denotes the transition relations between device modes; $A^d$ defines the allowable state region for each device mode in $M^d$. Fig. 1 shows an example of device models for laser tracheotomy surgery (or airway-laser surgery) [5], [9].

By introducing $A^d$, device $d$ is restricted to operate in certain safe states defined by $A^d_{\text{mode}}$ when its current mode is set to $\text{mode}$. By carefully defining $A^d$, medical device is safe for people to use. For example, in the laser tracheotomy surgery (Fig. 1), only when laser scalpel stays in the ALLOWED mode can it emit laser and let surgeon cut patient’s trachea. Surgeon’s request to emit laser triggers laser scalpel’s state to transit from $\text{Off}_\text{NoReq}$ to $\text{On}_\text{Req}$.

We say that the combination of a device mode $\text{mode}$ and a device state $\text{state}$ is a device condition (condition for short), denoted as $\text{cond} = (\text{mode}, \text{state})$. Device condition $\text{cond}$ is valid if $\text{state} \in A^d_{\text{mode}}$; otherwise, it is invalid.

B. Safety Hazards

Safety hazards commonly refer to some risky circumstances that we try to avoid. There are numerous types of hazards in the clinical domain, such as surgical fire, cerebral hypoxia, etc. However, from the system’s point of view, those hazards can be generally reflected by some adverse system states. For example, in laser tracheotomy surgery, emitting laser (laser scalpel’s state is $\text{Req}_\text{On}$) while the oxygen concentration in the trachea is high ($O_2$ sensor’s measurement $> 30\%$), can lead to surgical fire. Hence, MCPS interlock certain devices, like laser scalpel and $O_2$ sensor, and enforce that they never enter hazardous states simultaneously.

Definition 1. A safety hazard $h$ interlocks $p$ different medical devices $\{d_{h_1}, \ldots, d_{h_p}\}$, where $h \in H$ and $d_{h_i} \in Dev$; it specifies a combination of device states $[\text{state}_{d_{h_1}}, \ldots, \text{state}_{d_{h_p}}]$ that are prohibited to be entered simultaneously by those $p$ devices.

We denote $LkDev(h) = \{d_{h_1}, \ldots, d_{h_p}\}$. Any pair of devices in $LkDev(h)$ are said to have interlock relationship, and they are interlocked by $h$. Basically safety hazards are conjunctive predicates, and our objective is to prevent those predicates from happening. The following shows four safety hazards in laser tracheotomy surgery.

- H1: $[\text{Laser}=\text{On}_\text{Req}, O_2 > 30\%]$. It interlocks laser scalpel and $O_2$ sensor. Emitting laser when $O_2$ concentration is high will potentially yield surgical fire.
- H2: $[\text{Laser}=\text{On}_\text{Req}, \text{Vent}=\text{Vent}_\text{On}]$. It interlocks laser scalpel and ventilator. Before laser is allowed to be emitting, ventilator must stop pumping high-concentration oxygen into patient’s lung.
- H3: $[\text{Vent}=\text{Vent}_\text{Off}, \text{SpO}_2 < 90\%]$. It interlocks ventilator and pulse oximeter. Ventilator has to be turned on to resume $O_2$ supply before patient’s blood oxygen level ($\text{SpO}_2$) drops below certain threshold, e.g., 90%.
- H4: $[\text{Laser}=\text{On}_\text{Req}, \text{SpO}_2 < 90\%]$. It interlocks laser scalpel and pulse oximeter. Laser scalpel is not allowed to emit while patient’s $\text{SpO}_2$ is not high enough.

C. Open-loop Safe Mode

A medical device works in the open-loop manner when the feedback paths are yet to be established. As pointed out by [5], in general, there should exist an open-loop safe mode for any device. In the case when network is disconnected, a device tries to operate in the open-loop safe mode. Moreover,
if multiple devices operate in their open-loop safe modes at the same time, it does not result in any safety hazards. Usually, the initial setting for a MCPS is that all the devices are configured to their open-loop safe modes, and thus it is always safe before the clinical procedure begins. Therefore, we denote the open-loop safe mode as INIT for ease of narration. Take laser tracheotomy surgery as an example. In INIT mode, laser scalpel is forbidden to emit laser; meanwhile, ventilator is on to assist patient’s respiration. Note that open-loop safe mode is of great importance in this research. In our later design, medical devices are guided to gradually transit to the open-loop safe mode when message loss is detected.

D. Synchronization and Fault Model

In asynchronous distributed systems, the weakest assumption is that there is no time bound guarantee on processor execution speed, clock drift, and message transmission delay [10]. However, it poses great complexity to the upper-layer software design, resulting in higher probability of latent bugs. So in safety-critical real-time distributed systems, people often provide certain level of synchrony to facilitate the upper-layer software design, such as Time-Triggered Architecture (TTA) [11]. MCPS also fall into the category of safety-critical real-time systems. Because it demands prompt responses from medical devices or caregivers to ensure patient safety. Hence, in this work, we require all the medical devices to synchronize in CPU clocks when they join in the system. Existing synchronization algorithms for real-time distributed systems like [12], [11], [13], [14] can be utilized, and thus we consider this topic beyond the scope of this paper.

In this work, we aim at dealing with faults related to asynchronous and unreliable communication, and do not involve other types of faults. So we assume all the devices in the system operate by following their specifications and without crash. Most of existing work in MCPS made assumption of ideal network environments. However, in reality it is not uncommon that the network is unreliable, and network-related faults should be respected. For wired connection, a variety of factors, like jitter, buffer overflow and congestion, can lead to delayed message reception or message loss. Besides, people sometimes trip over the jumbled wires that connect various medical devices, resulting in disconnection. This situation becomes much more serious for wireless connection [6]. Wireless links are vulnerable to ambient interference. Multipath effect and self-fading can affect the link quality as well.

In fact, all the cases mentioned above can result in message loss in transit or long transmission delay. Since MCPS are safety-critical real-time system, to respect the timeliness, we associate each message with a deadline. Messages that miss deadline will be dropped even if they eventually reach the destination. This situation actually corresponds to the communication omission fault model [10], i.e., message loss may happen in communication channel, or between message buffer and the decision process. If critical messages get lost, it may lead to safety hazards and thus is risky. Therefore, fault-tolerant design is paramount.

III. SYSTEM DESIGN

This section presents the framework of system execution and our modular software design for medical devices.

A. Execution Framework

As all the devices are synchronized in CPU clock, they share a global time and the whole MCPS can be viewed to proceed cycle by cycle. The cycle length is denoted by $T$. In cycle $c$, a medical device $d$ constantly operates in mode $mode^d_c$. At the beginning of cycle $(c+1)$, device $d$ will switch to a new device mode $mode^d_{c+1}$. Transition from $mode^d_c$ to $mode^d_{c+1}$ has to comply with the transition relations defined in the device model $TS^d$, as well as the safety requirements so as to ensure patient safety.

As shown in Fig. 2, each cycle consists of three phases: Device Operation, Coordination, and Decision. The first phase, which lasts $T_{op}$, allows human subjects to operate the devices. Therefore, surgeons can carry out operation during this period. As time goes on, both the devices’ states and patient’s physiological state will change. Then, the system starts the Coordination phase to let devices exchange up-to-date feedback information. In our design, we expect all the messages to be received within $T_{tx}$; missing this deadline will cause the message to be dropped. Based on the obtained feedback information, devices enter the Decision phase to make decision and determine its mode for the next cycle. How to make safe local control for medical devices will be introduced in Section IV and V. Note that time span for Coordination or Decision phase is $T_{tx}$, and generally Device Operation phase should be much longer than the latter phases so as to satisfy surgeon’s or patient’s demands.

B. Modular Device Software Design

In this work, we view medical devices as smart devices rather than passive devices under other units’ control (e.g., surgeon or computer controller). Medical devices make local decision and collaborate with each other to achieve clinical goals as well as prevent hazards. Nevertheless, this capability will drastically complicate the device software. The side effect...
of increasing software complexity is that medical devices will become harder to be developed by device manufacturer, and verified by regulatory agencies, like U.S. FDA. In order to suppress the complexity, we come up with a modular software design, which is illustrated in Fig. 3.

First of all, there are three interfaces for a device. i) Human-Device Interface: This interface facilitates the interaction between medical device and human beings. ii) Communication Interface: It is used for interaction between medical devices. The communication can be either wired or wireless. iii) Device-Physical Environment Interface: Medical device interacts with the physical environment (patient is part of the physical world) via Sensors or Actuators.

These interfaces actually capture the major interactions which can happen in MCPS. However, not all the interfaces are necessary to exist in a device. For example, an oxygen sensor (O₂ sensor in Fig. 1) does not have actuators. If a medical device does not have actuators, then it will not take actions to change the physical environment. Instead, it keeps sampling and always stays in INIT mode. Thus, we call it sensing device; otherwise, it is an actuation device. Note that the MCPS Sys is composed of a set of sensing devices SD and a set of actuation devices AD, i.e., $Dev = SD \cup AD$.

In addition, a device has two functional modules: Message Processing and Decision Core. As discussed previously, MCPS are safety-critical real-time systems and we expect each message to be received within $T_{tx}$ after its sending time. Message Processing module basically deals with the timeliness issue of each message. While sending a message $m$, Message Processing module attaches the current cycle number $c^m$ and a deadline $D^m$ to the message before delivering it to the Communication Interface for transmission. On the other hand, as receiver, once a device receives a message from the Communication Interface, it checks the eligibility of the message, i.e., whether it meets the deadline constraint or not. To be specific, while a message $m$ is received, the receiver checks whether its current CPU time $t \leq c^m \times T + D^m$ or not. If the inequality holds, the message is considered as eligible and forwarded to the Decision Core for further processing. Otherwise, the message is ineligible, and thus filtered out.

Decision Core is the most critical module in a medical device. It collects information from the internal and external environment, as well as coordinates with others to determine the appropriate control decision. Decision Core comprises five submodules. i) Coordination: It initiates the coordination phase by exchanging messages with other devices. ii) Loop Constructor: It is responsible for establishing the feedback paths and closing the control loop. iii) Closed-loop Controller: In the case when a device is able to receive feedback messages and works in closed-loop mode, Closed-loop Controller is used by the device to make decision. iv) Open-loop Controller: After the feedback paths are established, a device may enter open-loop state if certain feedback messages are lost. In this case, Open-loop Controller is used to control the device. v) The above four submodules are under control of Switch, which has the logic to make the device switch to the appropriate submodule according to the specific context.

The rationale behind this modular design is that it enables us to develop each module/submodule separately. First, it eases the development of each module/submodule; we no longer need to design a monolithic and complex software which deals with all the possible situations. Second, the modules/submodules can be developed in parallel by different people or teams, and thus the time cost is reduced. Third, testing and verification of a medical device software can be decomposed into several parts with less complexity. Therefore, we believe in this way we can shorten the time-to-market for medical device products.

IV. DISTRIBUTED CONSTRUCTION OF FEEDBACK CONTROL FOR MCPS

This section introduces how a device $d$ is able to construct feedback control after it joins in the system. Establishing feedback paths is the key to close the control loop.

A. Device Interdependency

It is not necessary for $d$ to obtain feedback from all the other devices in the system. Apparently, if another device $d_i$ does not have interlock relationship with $d$, then $d$’s local decision does not rely on $d_i$. Based on this observation, we thus try to discover the device interdependency, so that $d$ only needs feedback information from those it depends on. In fact, dependence analysis is widely used to analyze artifacts in software and system design, such as [15], [16]. But none of them focus on medical device interdependency.

**Definition 2.** An actuation device $d$ DEPENDS on device $d_i$, denoted as $d \leftrightarrow d_i$, if they are interlocked by a safety hazard $h \in H$. Sensing devices do not DEPEND on any other devices, as they only keep doing sampling.

We use $Dp(d)$ to denote the set of devices that device $d$ DEPENDS on. If $d \in SD$, we define $Dp(d) = \emptyset$. We also denote the actuation devices in $Dp(d)$ and the sensing devices in $Dp(d)$ by $DpAD(d)$ and $DpSD(d)$, respectively. Device $d$ and $Dp(d)$ together constitute a cluster, denoted as $Ct(d)$, i.e., $Ct(d) = \{d\} \cup Dp(d)$. Device $d$ is the head of $Ct(d)$. For example, in Fig. 4, $Ct(Ventilator) = \{Pulse Oximeter, Laser Scalpel\}$. Note that if $d$ and $d_i$ are
actuation devices and \( d \leftrightarrow d_i \), then \( d_i \rightarrow d \) also holds (symmetric property). The motivation for defining cluster originates from the fact that cluster \( Ct(d) \) essentially creates a full execution environment for \( d \). Devices outside of \( Ct(d) \) do not directly affect \( d \)’s decision.

### B. Control Rules

In order to assist surgeon to work and prevent hazards, it is also necessary to define control rules to guide medical devices.

**Definition 3.** A control rule \( r \in Rules \) for device \( d \) consists of a trigger and a decision field:

\[
(\land_{d_i\in Dp(d)} \text{cond}^{d_i}) \land \text{cond}^d \rightarrow \text{mode}^d. \tag{1}
\]

Trigger field describes the device condition requirements that each device in \( Ct(d) = \{d_j\} \cup Dp(d) \) has to satisfy. If trigger field is satisfied, rule \( r \) leads \( d \) to switch to a new mode \( \text{mode}^d \). This definition implies that to make control decision, \( d \) has to be aware of the states of all the other devices that \( d \) DEPENDs on. For a control rule \( r \in \mathcal{R} \), we use \( \text{mode}(r) \) to denote the mode in \( r \)’s decision field. We say that \( d \)’s feedback control is constructed if the feedback paths from \( Dp(d) \) to \( d \) have been established.

Note that it is system designer’s responsibility to rigorously define control rules for a MCPS. On the one hand, executing a control rule should not result in any safety hazards. On the other hand, multiple devices may execute their own control rules simultaneously. However, if two actuation devices have interlock relationship, they have to execute control rules sequentially. Suppose \( d_i \) and \( d_j \) are interlocked actuation devices, and execute control rules simultaneously. Because, to ensure the safety of executing a control rule, \( d_i \) has to be aware of \( d_j \)’s condition since \( d_j \in Dp(d_i) \). However, once \( d_i \) executes a rule and enters a new mode, \( d_j \) changes its condition at the same time by executing its own rule. In this case, from \( d_j \)’s perspective, it does not know \( d_j \)’s up-to-date condition and thus cannot guarantee the safety of executing its control rule. Hence, this can potentially leak the system safety, and has to be prevented. For example, in laser tracheotomy surgery, ventilator and laser scalpel always execute control rules sequentially, e.g., ventilator executes control rule to deactivate itself first, and then laser scalpel is allowed to enter ALLOWED mode.

### C. Coordination Submodule

When a device \( d \) first joins in the system, it continuously synchronizes with other devices until success. Afterwards, it will launch Coordination submodule to construct feedback control. Alg. 1 illustrates how Coordination submodule works. A flag, named \( \text{lp\_success} \), is used to indicate whether device \( d \)’s feedback control has been constructed or not. The default value of \( \text{lp\_success} \) is \text{false} when \( d \) first joins in the system. To establish feedback paths, \( d \) sends \text{REQUEST} message to devices it DEPENDs on, and wait for them to reply. Meanwhile, it will also receive \text{REQUEST} messages from other devices. In this case, \( \text{ReqDev}^d \) is used to record those devices whose \text{REQUEST} messages have been received in current cycle \( c \) (line 10-12). Later on, in Decision phase, \( d \) will reply them by \text{REPLY} message. Note that \( d \) also maintains a set of devices that have already replied to \( d \), which is denoted by \( \text{RepDev}^d \). Hence, \( d \) only need to send \text{REQUEST} message to \( \text{Dp}(d) \setminus \text{RepDev}^d \) (line 3-4). Also note that all the messages that come into and out of Decision Core have to be processed by Message Processing module.

**Algorithm 1: Coordination in cycle \( c \)**

1. **if** Coordination phase starts **then**
   2. **if** \( \text{lp\_success} == \text{false} \) **then**
   3. **foreach** \( d_i \in \text{Dp}(d) \setminus \text{RepDev}^d \) **do**
      4. send \text{REQUEST} to \( d_i \);
      5. \( \text{ReqDev}^d \leftarrow \emptyset \);
   6. **else**
      7. **foreach** \( d_i \in \text{ReqDev}^d \) **do**
         8. compose and send \text{FDBK} to \( d_i \);
      9. \( \text{FbDev}^d \leftarrow \emptyset \);
   10. **while** Coordination phase does not end **do**
      11. **if** receive \text{REQUEST} from \( d_i \) **then**
         12. \( \text{ReqDev}^d \leftarrow \text{ReqDev}^d \cup \{d_i\} \);
         13. \( \text{ReqDev}^d \leftarrow \text{ReqDev}^d \cup \{d_i\} \);
      14. **if** receive \text{FDBK} from \( d_i \) **then**
         15. \( \text{FbDev}^d \leftarrow \text{FbDev}^d \cup \{d_i\} \);

### D. Switch Submodule

While Coordination phase ends and Decision phase starts, Switch submodule will initiate the Loop Constructor. The detailed algorithm for Switch submodule is show in Alg. 2. If \( \text{lp\_success} = \text{false} \), it invokes Loop Constructor to continue setting up feedback paths. Otherwise, Closed-loop Controller or Open-loop Controller will be enabled to make control decision (line 6-10).

### E. Loop Constructor Submodule

Now let us look at the algorithm for Loop Constructor (shown in Alg. 3). In this stage, medical devices exchange \text{REPLY} messages based on the \text{REQUEST} messages they have received in the former phase. First of all, \( d \) replies to the devices in \( \text{ReqDev}^d \). Then, it continuously updates \( \text{RepDev}^d \) when receiving reply messages (line 4-6). Finally, at the end of the process, the decision is made and the handling procedure is executed.
Algorithm 2: Switch in cycle \( c \)

1. if Coordination phase starts then
   2.   enable Coordination submodule;
3. if Decision phase starts then
   4.   if \( lp\_success == false \) then
      5.     enable Loop Constructor submodule;
   6.   else
      7.     if \( FbDev^d \neq Dp(d) \) then
         8.       enable Closed-loop Controller submodule;
      9.   else
         10.  enable Closed-loop Controller submodule;

of Decision phase, \( d \) checks whether local feedback control has been set up or not by comparing \( RepDev^d \) with \( Dp(d) \). If these two sets are equivalent, \( d \) believes all the feedback paths from \( Dp(d) \) have been successfully established. Thus, it updates \( lp\_success \) to true, and performs initialization for the future distributed control (line 7-10). We denote the cycle number of this success time by \( c_0 \). From cycle \( c_0 \) on, \( d \) will start exchanging feedback messages with others. Meanwhile, Loop Constructor will not be enabled any more. Note that since sensing devices do not perform feedback control, flag \( lp\_success \) inside them is always false. However, they still should reply other devices’ requests.

Algorithm 3: Loop Constructor in cycle \( c \)

1. if Decision phase starts then
   2.   foreach \( d_i \in ReqDev^d \) do
      3.     send REPLY to \( d_i \);
4. while Decision phase does not end do
   5.   if receive REPLY from \( d_i \) then
      6.     \( RepDev^d \leftarrow RepDev^d \cup \{d_i\} \);
7. if Decision phase ends then
   8.   if \( RepDev^d == Dp(d) \) then
      9.     \( lp\_success \leftarrow true \);
10. initialize condition vector;

F. Safety Analysis

Under the communication omission fault model, both REQUEST and REPLY messages may get lost. In fact, the procedure for constructing feedback control will be repeated until success. We observe that device \( d \) never changes its device mode during the entire construction process, namely, it always stays in INIT. What’s more, \( d \) does not send feedback message to others in this process (Safety Rule 1). However, due to the variability of network condition, the success time of constructing feedback control in some devices may be earlier than others. To ensure safety in this case, we embed another rule into each medical device: after cycle \( c_0 \), only if all the feedback messages from \( Dp(d) \) have been received can device \( d \) be allowed to leave INIT mode (Safety Rule 2). In fact, after cycle \( c_0 \), a device will operate in the open-loop control mode if some feedback messages from \( Dp(d) \) are lost. Later on in next section, we will show that Safety Rule 2 can be fulfilled by Open-loop Controller.

Now, we start to analyze the safety of our proposed algorithms based on the above safety rules. First of all, we give the definition about “safe”: an algorithm is said to be safe if it does not lead to any safety hazards in any cycles.

Theorem 1. The algorithms for distributed construction of feedback control are safe.

The detailed proof for Thm. 1 can be found in the Appendix of technical report [17].

V. DISTRIBUTED OPEN-LOOP AND CLOSED-LOOP CONTROL FOR MCPS

In this section we will discuss how each medical device performs distributed open-loop and closed-loop control.

A. Condition Vector

In ideal network where there is no message loss, device \( d \) computes a new device mode cycle by cycle in the Decision phase. To tolerate message miss, our basic idea is that while making decision in each cycle \( c \), in addition to calculating a device mode \( mode^d_{c+1} \) for the coming cycle, a device \( d \) plans a vector of device modes for the future \( l \) cycles, that is, from cycle \( (c + 2) \) to \( (c + l + 1) \). Therefore, \( d \) is able to use the planned modes in case of message miss.

We create a special data structure, called condition vector, to organize such planned device modes. A condition vector generated by \( d \) for cycle \( c \) is denoted by \( Vec^d_c \). The \( k \)th element in the vector is denoted by \( Vec^d_c[k] \), which comprises a device mode \( Vec^d_c[k].md \) and a device state \( Vec^d_c[k].st \). Following the same notation, we use \( "\ast" \) to represent any allowable device state in a mode. So, condition \( (mode\_d, \ast) \) essentially represents a mode. A valid condition vector \( Vec^d_c \) for cycle \( c \) has to satisfy the following rules.

- R1: The mode in the last element of \( Vec^d_c \) is the open-loop safe mode, i.e., \( Vec^d_c[l] = (INIT, \ast) \);
- R2: Suppose current device mode is \( mode^d_l \). Transition from \( mode^d_k \) to \( Vec^d_k[1] \), and from \( Vec^d_k[k] \) to \( Vec^d_k[k+1] \) for all \( 1 \leq k \leq l-1 \), must comply with the transition relations \( \mathcal{T}^d \) defined in \( d \)'s device model.

If \( l = 0 \), then \( Vec^d_c \) is empty. This reduces to the case for ideal network. Rule R2 basically means that device \( d \) finally switches to the open-loop safe mode after \( l \) cycles of consecutive message misses. The rationale under rule R2 is that after \( l \) consecutive message misses, it is reasonable for \( d \) to view the current network condition is fairly poor or even disconnected. Therefore, it tries to operate on the open-loop safe mode under this situation. Configuration of \( l \) actually is related to the network condition and will also affect system performance. We will evaluate the impact of \( l \) on the system.
in Section VI. As initialization, in cycle $c_0$ when actuation device $d$'s feedback control is constructed (line 10 in Alg. 3), it generates the first condition vector $Vec_{c_0+1}^d$ by setting all the elements to $(\text{INIT}, \ast)$.

Because of the introduction of condition vector, making decision in cycle $c$ turns to be generating a new mode $mode_{c+1}^d$ as well as a condition vector $Vec_{c+1}^d$ for the next cycle. Now we introduce a basic operation on condition vector, called pad operation, which is denoted by $Pad()$. The output of pad operation on condition vector $Vec_c^d$ is a padded vector $PVec^d$ of length $(l + 1)$. To be specific, $PVec^d$ reuses a range of elements in the $Vec_c^d$ from $1$ to $l$, and attaches the open-loop safe mode in its tail: $PVec^d[k] = Vec_c^d[k]$ for $1 \leq k \leq l$; and $PVec^d[l + 1] = (\text{INIT}, \ast)$.

B. Coordination Submodule

The detailed algorithm for Coordination submodule has already illustrated in Alg. 1. Since $lp_{\text{success}}$ becomes true now, Coordination submodule will exchange feedback messages. When Coordination phase starts, feedback message, denoted by $FDBK$, is composed. If $d \in AD$, then $d$'s $FDBK$ message consists of two parts: i) $d$'s up-to-date condition $cond_c^d = (mode_c^d, \text{state}_c^d)$; ii) its current condition vector $Vec_c^d$. On the other hand, if $d \in SD$, $FDBK$ message only contains part i). Then, $d$ will send $FDBK$ message to the devices in $ReqDev_c^d$, which represents the devices whose $REQUEST$ messages have been received so far (line 7-8). Meanwhile, a set, $FbDev_c^d$, is continuously updated to record the devices while their feedback messages are received in current cycle $c$ (line 14-15).

C. Open-loop Controller Submodule

When the Decision phase starts, Switch component checks whether or not all the $FDBK$ messages from $Dp(d)$ have been received, e.g., whether $FbDev_c^d = Dp(d)$. If they are equal, Switch submodule enables the Closed-loop Controller; otherwise, Open-loop Controller will be chosen to make decision (line 7-10 in Alg. 2).

Algorithm 4: Open-loop Controller in cycle $c$

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if Decision phase starts then</td>
</tr>
<tr>
<td>2</td>
<td>foreach $d_i \in ReqDev_c^d$ do</td>
</tr>
<tr>
<td>3</td>
<td>send REPLY to $d_i$;</td>
</tr>
<tr>
<td>4</td>
<td>Load($Vec_c^d$);</td>
</tr>
</tbody>
</table>

The algorithm for Open-loop Controller is demonstrated in Alg. 4. Due to the variability of network condition, there is a possibility that $d$ constructed feedback control earlier than others in $Ct(d)$. So $d$ can still receive REQUEST message from other devices in current cycle $c$. In this case, $d$ should reply them with REPLY message (line 1-3). At the time of making decision, what $d$ does is invoking a load procedure, denoted by $Load()$, which uses the planned mode in the condition vector as its next operational mode and generates a new condition vector $Vec_{c+1}^d$ (line 4). To be specific,

- $mode_{c+1}^d = Vec_{c+1}^d[1]$;
- For $1 \leq k < l$, $Vec_{c+1}^d[k] = Vec_c^d[k + 1]$;
- $Vec_{c+1}^d[l] = (\text{INIT}, \ast)$, i.e., add INIT mode at the end.

Now let us discuss about Safety Rule 2. After cycle $c_0$, in the case of missing feedback messages from some devices in $Dp(d)$, device $d$ actually launches Open-loop Controller to make decision. Note that the initial condition vector $Vec_{c_0+1}^d$ only contains INIT mode. Meanwhile, load procedure always generates new condition vector by attaching INIT in the tail. Therefore, $d$ will continuously operate in INIT mode after $c_0$ till it receives all the feedback messages from $Dp(d)$. Thus, Safety Rule 2 can be ensured.

D. Closed-loop Controller Submodule

In this subsection, we describe the algorithm for Closed-loop Controller (shown in Alg. 5). Feedback message contains a device’s up-to-date condition, thus based on the feedback messages from $Dp(d)$, device $d$ is now able to determine whether it should execute control rule or not. If none of the control rules’ trigger fields is satisfied, $d$ will launch load procedure to make decision (line 1-2). Otherwise, suppose a control rule $r$ is satisfied, $d$ will instead invoke a procedure, called $GenLongest()$, to compute $mode_{c+1}^d$ and $Vec_{c+1}^d$.

Algorithm 5: Closed-loop Controller in cycle $c$

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if no control rule satisfied then</td>
</tr>
<tr>
<td>2</td>
<td>Load($Vec_c^d$);</td>
</tr>
<tr>
<td>3</td>
<td>else</td>
</tr>
<tr>
<td>4</td>
<td>generate local view $View_{c+1}^d$;</td>
</tr>
<tr>
<td>5</td>
<td>if $GenLongest()$ returns null then</td>
</tr>
<tr>
<td>6</td>
<td>Load($Vec_c^d$);</td>
</tr>
<tr>
<td>7</td>
<td>else</td>
</tr>
<tr>
<td>8</td>
<td>$GenLongest()$;</td>
</tr>
</tbody>
</table>

Fig. 5 shows how $GenLongest()$ procedure works. First of all, based on the obtained feedback messages in cycle $c$, device $d$ creates a local view on the conditions for devices in $Dp(d)$. We use $View_{c+1}^d$ to denote this local view, and $View_{c+1}^d(d_i)$ to represent its view on a specific device $d_i$. Basically, $View_{c+1}^d$ is a local estimation of other devices’ conditions in cycle $(c + 1)$ and subsequent $l$ cycles. Based on $View_{c+1}^d$, $d$ is then able to figure out its own decision. If the local view can reflect the real conditions of other devices, making local decision based on the local view is thus safe. So maintaining local view correctly is the key to distributed control. Specifically, if $d_i \in Dp(d)$ is an actuation device, $View_{c+1}^d(d_i)$ is calculated by performing pad operation on the received condition vector $Vec_{c}^d$, i.e., $View_{c+1}^d(d_i) = Pad(Vec_{c}^d)$. On the other hand, if $d_i$ is a sensing device, it generates $View_{c+1}^d(d_i)$ according to the following procedure, denoted by $Predict()$.

- For all $1 \leq k \leq l + 1$, $View_{c+1}^d(d_i)[k], md = \text{INIT}$;
- $View_{c+1}^d(d_i)[1], \text{st} = \text{cond}_{c_i}^d, \text{st}$;
For all $1 < k \leq l + 1$, $View_{c+1}^d(d_i)[k].st$ is calculated according to the worst-case prediction, i.e., $View_{c+1}^d(d_i)[k].st = \text{Worst}(View_{c+1}^d(d_i)[k-1].st)$, where $\text{Worst}(\cdot)$ is the worst-case prediction function for a specific physical variable.

Through $\text{Predict}()$, we actually determines the worst-case sensor measurements for the future $l$ cycles. For example, suppose $d_i$ is a pulse oximeter, $cond_{c+1}^{d_i}.st = 98\%$ and the maximum decrease of patient’s SpO2 in one cycle is $3\%$. Device $d$ can infer that the worst-case SpO2 measurements for the coming two cycles are $View_{c+1}^d(d_i)[2].st = 95\%$ and $View_{c+1}^d(d_i)[3].st = 92\%$. Note that worst-case prediction function can be derived from either patient model or empirical medical studies.

Now based on the local view, $d$ is able to calculate $mode_{c+1}^d$ and $Vec_{c+1}^d$. The calculation has to follow several rules:

- **R3:** $mode_{c+1}^d = mode(r)$;
- **R4:** For $1 \leq k \leq l$, $Vec_{c+1}^d[k]$ has to satisfy rule R1 and R2;
- **R4:** The combination of $mode_{c+1}^d$ or $Vec_{c+1}^d$ with $View_{c+1}^d$ does not result in any safety hazards.

Basically, rule R3 implies that by executing rule $r$ device $d$ intends to operate in mode $mode(r)$. To ensure R4, it is necessary for $d$ to check every safety hazard related to $d$. For example, in laser tracheotomy surgery, ventilator has to check both hazard H2 and H3. Therefore, control rule $r$ may not be allowed to be executed, if hazard can happen. In this case, $\text{GenLongest}()$ returns null, and load procedure is invoked again (line 5-6). On the contrary, if there exist a set of qualified condition vectors for $Vec_{c+1}^d$, denoted by $QuVeVec_{c+1}^d$, we intentionally select one of them according to an additional rule R5: For each element $VeVec_{c+1}^d$ in $QuVeVec_{c+1}^d$, starting from $VeVec_{c+1}^d[1]$, there is a sequence of consecutive device condition $\langle mode(r), * \rangle$. The element, which has the longest such sequence, is selected as the final $VeVec_{c+1}^d$. If there are multiple such longest condition vectors, we randomly select one of them.

The interpretation for rule R5 is that if $d$ intends to execute a control rule $r$ and switch to mode $mode(r)$, we allocate the longest time for $d$ to let it continuously operate in $mode(r)$. This approach is based on the observation that the intended mode $mode(r)$ commonly is a desirable state that the system want to reach/maintain. For example, in the laser tracheotomy surgery, if laser scalpel intends to operate in ALLOWED mode, we assign the longest time for it to continue operating in ALLOWED mode so that surgeon can have enough time to carry out the surgery. If we consider the overall operation time in such desirable state as a metric of system performance, by rule R5, high performance can be retained.

### E. Safety Analysis

Apparently, no matter open-loop or closed-loop controller is used, the ultimate decision is based on either $\text{Load}()$ or $\text{GenLongest}()$ procedure. In either case, actually we can prove that no safety hazards can be resulted in.

**Theorem 2.** The algorithms for distributed open-loop and closed-loop control are safe.

Also, the detailed proof for Thm. 2 can be found in the Appendix of [17]. Finally, Thm. 1 and Thm. 2 together show that all the algorithms for distributed control are safe.

### F. Discussion

In the algorithms above, an important concern is about the configuration of cycle length $T$. Within each cycle, we assume the measurements for sensing devices are stable, so as to figure out the modes for actuation devices. So if the cycle length is very long, the measurements for some physical/physiological variables may change drastically in one cycle period. In addition, we use the worst-case prediction to estimate the future measurements for sensing devices. Our previous work [9] shows that the prediction accuracy of patient’s physiological state decreases as prediction period gets longer. Therefore, in this sense, cycle length should not be too long. On the other hand, shorter cycle length leads to more message exchange and thus introduces higher overhead. In fact, we should consider all the above aspects to balance accuracy and overhead.

### VI. Evaluation

This section evaluates our distributed algorithms by simulating a concrete MCPS, i.e., laser tracheotomy surgery, whose device models have been described in Fig. 1. We collect real world traces to serve as the measurements for pulse oximeter and O2 sensor. Specifically, we use two sets of traces. i) PhysioNet Trace: This trace is collected from PhysioNet database [18], which is set up by NIH, NIBIB and NIGMS, and stores real-world medical traces logged by hospitals. ii) HKPolyU Trace: This trace is retrieved from an emulated laser tracheotomy surgery [9] conducted by two human subjects. The SpO2 and CO2 measurements are recorded by medical device, Nonin 9843. Note that both traces last 20 minutes, i.e., the surgery lasts 20 minutes.

#### A. Trace Analysis

Our first step is to analyze the traces to obtain the worst-case prediction functions. However, we confronted two problems. First, due to the limitation of existing devices and traces, we are not able to collect traces about airway O2 concentration. Fortunately, CO2 data can be found, and thus we use CO2 data to replace O2 measurements because of the negative
correlation between O₂ and CO₂ measurement in patient’s trachea [9], i.e., high (low) O₂ concentration implies low (high) CO₂ concentration. Second, sampling periods in the traces are not equal to the cycle length T.

In laser tracheotomy surgery, in terms of worst-case variation of SpO₂ and CO₂ measurements, we are particularly interested in their maximum decrease in one cycle. Let WCS and WCC denote the maximum decrease for one sampling period and one cycle, respectively. Suppose the sampling period is P. If P < T, there may exist ⌊T/P⌋ samples in a cycle. We calculate the maximum decrease for those ⌊T/P⌋ samples in a cycle, and use it as WCC. On the other hand, if P ≥ T, then WCC = WCS. In this case, it is conservative to think that the maximum decrease in a short period time (a cycle) is equal to that in a long period of time (a sampling period). Thereby, in this way we are able to obtain the worst-case decreases for different Ts. Tab. I shows the properties of our two traces, and the analysis results for T = 200 ms.

<table>
<thead>
<tr>
<th>Trace</th>
<th>Parameter</th>
<th>Unit</th>
<th>Sampling Period P</th>
<th>Worst-case Decrease in One Cycle (T = 200 ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhysioNet</td>
<td>SpO₂</td>
<td>%</td>
<td>8 ms</td>
<td>1.872</td>
</tr>
<tr>
<td></td>
<td>CO₂</td>
<td>mV</td>
<td>3 ms</td>
<td>0.014</td>
</tr>
<tr>
<td>HKPolyU</td>
<td>SpO₂</td>
<td>%</td>
<td>2 s</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>CO₂</td>
<td>mmHg</td>
<td>2 s</td>
<td>27</td>
</tr>
</tbody>
</table>

**B. Simulation Setup and Results**

In this paper, we consider message loss as fault in the system. To test our algorithms, we follow the well-known fault-error-failure cycle [19]. During the simulation, we inject two types of faults into the system. i) Random message loss: Each message will not be received by receiver’s Decision Core with probability Ploss. This type of fault characterizes the network instability, e.g., ambient interference. ii) Consecutive message loss: This fault type can capture network disconnection for a certain period of time. Specifically, laser scalpel is disconnected from cycle 1500; ventilator is disconnected from cycle 3000; and both of them are disconnected from cycle 4500. The disconnection period for each above case is 2 seconds.

Meanwhile, we set the cycle length T = 200 ms. The surgeon who operates the laser scalpel randomly requests the laser to emit with probability 0.5 in each cycle. To evaluate the algorithms, we mainly focus on two metrics: the observable failures (safety hazards), and the percentage of cycles in which laser scalpel’s mode is ALLOWED. Only in ALLOWED mode can surgeon carry out the surgery. So the second metric reflects the performance of our designed algorithms.

Firstly, we test the algorithms for constructing feedback control. Since sensing devices always operate in INIT mode, we only demonstrate the mode diagram for laser scalpel and ventilator. From Fig. 6, we can see that both laser scalpel and ventilator stay in INIT mode before their feedback control is successfully constructed (arrows in the figure indicate the success time). This result confirms that the protocols for constructing feedback control are safe. Due to space limitation, we only show the figure when Ploss = 0.1 and l = 4. In fact, results for other settings show the same conclusion.

Secondly, we check whether safety hazards can happen or not during open-loop and closed-loop control. Due to space constraint, we only show the mode diagram from cycle 3000 to 3010 when ventilator is disconnected from the network. In PhysioNet Trace (Fig. 7(a)), ventilator continues to operate in DEACTIVATED mode in cycle 3001. Meanwhile, laser scalpel operates in ALLOWED mode for one more cycle after losing contact with ventilator. This is because of rule R5, by which we allocate the longest time for laser scalpel to be in ALLOWED mode. Hence, even when messages get lost, laser scalpel is still able to sustain in ALLOWED mode. Furthermore, finally both laser scalpel and ventilator will transit to the INIT mode after consecutive message losses. Fig. 7(b) shows that for the HKPolyU Trace, both laser scalpel and ventilator operate in INIT mode during the entire disconnection period, and thus they never result in safety hazards.

At last, we evaluate the performance of our algorithms. The performance results are shown in Fig. 8. Blue curves represent the performance for perfect network scenario (Ploss = 0), where there is no need to use condition vector. First of all, from Fig. 8, we can see that in contrast with perfect network scenario, our algorithms can achieve comparable performance.
in the environment with small message loss probability, e.g., $P_{\text{loss}} = 0.1$. Second, we fix $P_{\text{loss}}$ and vary $l$. Clearly system performance increases as $l$ becomes larger. This is because longer condition vector can accommodate more ALLOWED modes, and thus gives more chances for laser scalpel to operate in ALLOWED mode while message losses happen. However, there exists a threshold $\delta$ for the vector length. When $l$ becomes larger than $\delta$, system performance can not be improved any more. Note that $\delta$ is equal to 4 and 5 for PhysioNet and HKPolyU Trace, respectively. This result is easy to understand because by using worst-case prediction, several cycles later the predicted SpO2 or CO2 measurement becomes quite low, and thus does not allow laser scalpel to continue staying in ALLOWED mode for infinite time. Also note that if $l = 1$, system performance is 0, because condition vector of length 1 contains only one INIT mode. Similarly, laser scalpel’s local view contains only INIT mode as well, thus laser scalpel is not allowed to execute any control rule to leave INIT mode. Third, while fixing $l$, it is apparent that higher message loss probability will lead to lower system performance. As network condition becomes poorer, ventilator and laser scalpel will spend more time staying in INIT mode.

VII. RELATED WORK

How to make safe control is crucial in MCPS, and thus has attracted a lot of research efforts, such as [5], [6], [7], [8]. However, our approach differs from these work in several aspects. First, none of the existing work has considered the mechanism for constructing feedback control. Second, even though [5] [6] share the same assumption that the network is unreliable, they deal with the safe control problem by centralized approach. Whereas, our objective is to design distributed control mechanisms. Arney et al. [7] also discussed the impact of network disconnection on MCPS. However, its main goal is to involve patient in the control loop and formally verify the system using UPPAAL tool. Third, [8] does not focus on control mechanism design; instead, it tries to provide a coordination framework to support closed-loop control.

Another category of related work is the distributed control mechanisms proposed in other domains, and generally their goals are different from ours, which is to achieve safe medical device integration. i) Vehicular cyber-physical systems. Fallah et al. [20] proposed algorithms to control vehicle’s communication power and speed so as to avoid collision. ii) Sensor/actuator networks. Xia et al. [21] proposed a paradigm to let sensors and actuators coordinate without sink node’s help. iii) Robotic networks. There have been a wealth of research work on autonomous robot design. We refer the interested readers to survey paper [22]. The common goals in robotic networks include motion coordination, localization, navigation, and task assignment.

VIII. CONCLUSION

In this work, we investigated the problem of distributed control in MCPS. We proposed a modular software design for medical devices, and devised a set of distributed algorithms to deal with constructing feedback control, open-loop and closed-loop control, respectively. The prominent feature of our algorithms is that they are able to tolerate message losses and prevent safety hazards. Through this study, we show that distributed control can be a feasible and promising solution for medical device integration. In the future, we plan to collect various realistic fault types from clinical staff, such as physical disconnection (e.g., pulse oximeter detached from patient’s finger), and investigate the fault-tolerance issues.

REFERENCES

APPENDIX

A. Proof for Theorem 1

Proof: In the proof, we look at an arbitrary cluster $Ct(d)$, where $d \in AD$. Cycle $c_0$ is the success time of $d$’s feedback control. First of all, we will show that for any cycle $c \leq c_0$, devices in $Ct(d)$ always stay in \texttt{INIT} mode. Since cluster $Ct(d) = \{d\} \cup DpSD(d) \cup DpAD(d)$, we consider devices in $\{d\}$, $DpSD(d)$ and $DpAD(d)$, separately.

Case 1: $d_i \in \{d\}$. It is obvious that $mode_{c_i} = \texttt{INIT}$ for any cycle $c \leq c_0$.

Case 2: $d_i \in DpSD(d)$. It is trivial that $d_i$ always stays in \texttt{INIT} mode, as $d_i$ is a sensing device.

Case 3: $d_i \in DpAD(d)$. Since $d$ and $d_i$ are actuation devices, they must be mutually dependent on each other due to symmetric property of $\texttt{INIT}$ mode. Thus, $d \in Dp(d_i)$. According to Safety Rule 2, $d_i$ has to wait for feedback message from $d$ in order to leave \texttt{INIT} mode. However, before cycle $c_0$, device $d$ has not constructed feedback control yet, and thus it is impossible for $d$ to send feedback message to $d_i$ (Safety Rule 1). Missing feedback message from any of the dependent devices will lead $d_i$ to remain in \texttt{INIT} mode, no matter $d_i$’s feedback control has been constructed or not.

Combining Case 1, Case 2 and Case 3, we easily conclude that for any cycle $c \leq c_0$, devices in $Ct(d)$ stay in \texttt{INIT} mode. As \texttt{INIT} mode is open-loop safe, it must be safe if all the devices in $Ct(d)$ stay in \texttt{INIT} mode. Furthermore, since $Ct(d)$ is an arbitrary cluster in MCPS $Sys$, the algorithms for constructing feedback control are thus safe for $Sys$.

B. Proof for Theorem 2

Proof: Again we focus on an arbitrary cluster $Ct(d)$, where $d$ is an actuation device. In fact, devices in $Ct(d)$ can possibly cause some safety hazards, which form a subset of $\mathcal{H}$. Let us denote this subset by $CtHd(d)$. To prove the safety property of open-loop and closed-loop control algorithms, we have to show that in any cycle $c > c_0$, none of the safety hazards in $CtHd(d)$ can happen in cluster $Ct(d)$.

Suppose $h$ is an arbitrary safety hazard in $CtHd(d)$. First of all, let us prove that $h$ will certainly not happen in any cycle $c > c_0$. Specifically, we only consider devices in $Ct(d) \cap LkDev(h)$, because other devices in $Ct(d)$ are irrelevant to safety hazard $h$.

Case 1: Suppose before cycle $c$, the most recent time that a device in $Ct(d) \cap LkDev(h)$ launches $GenLongest()$ to make decision is in cycle $c_1$. As actuation devices that have interlock relationship have to execute control rules sequentially, it is impossible that multiple devices in $Ct(d) \cap LkDev(h)$ launching $GenLongest()$ in the same cycle. This implies that there is only one device launching $GenLongest()$ in cycle $c_1$, which is denoted by $d_i$. The resultant condition vector by executing $GenLongest()$ is denoted as $Vec_{c_1+1}^{d_i}$. From cycle $(c_1 + 1)$ to $c$, all the devices in $Ct(d) \cap LkDev(h)$ have to continuously launch load procedure to make decision. If $c_1 < c < c_1 + l + 1$, then for cycle $c$, an actuation device $d_j$ in $Ct(d) \cap LkDev(h)$ must operate in the mode which is planned in $Vec_{c_1}^{d_j}$. Meanwhile, for a sensing device in $Ct(d) \cap LkDev(h)$, its real measurement in cycle $c$ must be better than the corresponding measurement in $d_i$’s local view $View_{c_i+1}^{d_i}$, because $View_{c_i+1}^{d_i}$ is generated by worst-case prediction. In summary, local view $View_{c_i+1}^{d_i}$ reflects the real device modes for actuation devices in future $l$ cycles, and uses measurements that are worse than the real ones for sensing devices. While $d_i$ executes $GenLongest()$ based on local view $View_{c_i+1}^{d_i}$, its determined mode for cycle $c$ must be hazard-free. Therefore, using these planned modes in cycle $c$ by $Ct(d) \cap LkDev(h)$ will not result in any safety hazards. On the other hand, if $c \geq c_1 + l + 1$, as $d_i$ only checks safety hazards for future $l$ cycles, the device conditions for cycle $c$ are beyond this checking range. However, since the last element in the condition vector is \texttt{INIT} mode and load procedure always adds the \texttt{INIT} mode in a condition vector’s tail, in cycle $c$ all the devices in $Ct(d) \cap LkDev(h)$ must operate in \texttt{INIT} mode. Obviously, combination of open-loop safe modes will not cause safety hazard $h$ to occur.

Case 2: There is no such $c_1$, i.e., before cycle $c$, devices in $Ct(d) \cap LkDev(h)$ either have not yet constructed feedback control, or continuously launch load procedure on their initial condition vectors. Thereby, all the devices in $Ct(d) \cap LkDev(h)$ are certain to operate in \texttt{INIT} mode. In this case, safety hazard $h$ will not happen either in cycle $c$.

In summary, the algorithms for open-loop and closed-loop control can make sure $h$ will not happen in any cycle $c > c_0$. Because $h$ is an arbitrary safety hazard in $CtHd(d)$, we conclude that none of the safety hazards will be caused in cluster $Ct(d)$. Furthermore, by applying this conclusion to all the clusters in MCPS $Sys$, we know that no safety hazards will happen in the MCPS. Thus, the algorithms for distributed open-loop and closed-loop control are safe.