Toward Safe Interoperations in Network Connected Medical Cyber-Physical Systems Using Open-Loop Safe Protocols

Andrew Y.-Z. Ou*, Maryam Rahmaniheris*, Yu Jiang†, Po-Liang Wu* and Lui Sha*
*University of Illinois at Urbana-Champaign, USA
†Tsinghua University, China

***Abstract*— Using wireless networks in medical Cyber-Physical Systems could be challenging. Because the medical system not only assists the medical personnel to deliver medical services to the patient but also needs to deal with accidental situations such as communication failures without compromising the patient's safety. Previous research work tackled the communication failure problems in medical CPS from architecture perspectives. However, as medical devices configurations become more complex when a medical CPS is composed of many medical devices, we need to know that whether the certain configuration and a combination of the devices will not compromise the patient's safety. We present an algorithm to tackle the problem that whether a given system configuration exists a possible series of system transitions that allows the physicians to perform medical operations; in the mean time, the system transitions ensure the patient's safety while communication failures may happen during the transitions.**

I. Introduction

In a network connected medical cyber-physical system, networks provide communication among medical devices. It also allows the possibility of configuration and combinations of medical devices for different medical procedures. However, within the network environment, a medical CPS needs to deal with the situation when the networks do not provide services which will impact the safety and effectiveness of medical procedures.

One of the most important issues when using wireless networks for medical CPSs is that failures or glitches of the communication networks should not lead to negative impacts to the patient as well as the medical procedures. This problem is known as the "open-loop" problem because the supervisory control loop becomes open if the communication networks fail. In another case, even if the communication networks do not fail but suffer from a long and unexpected delay, it may cause negative impacts to the system. For example, in the widely used Medical Device Plug-and-Play (MD PnP) framework [1], if the supervisor cannot query the state of the devices due to communication loss, the supervisor will not be able to decide the following actions for each device without the state information of each device. For this case, medical devices cannot further coordinate because of lacking communications networks. Without further coordinations, the system may violate the predefined safety constraints and hence brings the patient to a dangerous situation. As a result, it is required to handle the situation when the communications network may fail without compromising the patient's safety.

Kim *et al.* [2] and Tan *et al.* [3], [4] did the pioneering work on open-loop safe problems from different perspectives. Kim *et al.* propose a vector-based solution that the centralized supervisor sends out a vector including a list of future actions for the device. A device performs the actions specified in the vector when communication fails. Tan *et al.* propose a design pattern to tackle the open-loop safe problem and use formal methods to prove that the proposed design pattern is correct. A common assumption is that the system needs to be formalized as suggested by the paper to deliver the safety guarantee. However, it is not known that given a system model whether the system may be communication failure resistant or not. This is especially important because of the increasing number of devices of generic medical scenarios and the dynamic random failure mode of wireless communication bringing new challenges to the open-loop safe problem. More specially, the increasing number of medical devices will result in more complex interlaced safety constraints and make it even more difficult to determine whether the system can safely operate under communication failures.

In this paper, we present algorithms for medical CPSs to determine whether the system can guarantee safe system state transitions in the presence of communication failures. First, we analyze a medical system of tracheotomy and the safety constraints for the patient. Based on the medical scenario, we provide a formal representation of a medical system. With the system models, we propose an algorithm to analyze the system models to determine whether the system can safely transit among system states and can handle the situation of communication failures or suffering from a long delay. Further more, if there exist multiple possible system transitions, we propose an algorithm to select the path with the longest period that allows medical personnel to perform medical operations.

For evaluation, we provide a case study with multiple devices and a set of safety requirements to examine the proposed path finding algorithm on the tracheotomy systems to find the configuration of the system to operate safely. The results show that the resulting system with the generated configuration and predefined safety constraints is safe under communication failures.

Overall, our major contributions in tackling the open-loop safe problem in medical cyber-physical systems are to determine whether a system could be open-loop safe, and, if the system could be open-loop safe, what could be the configuration of the system that allows medical personnel to perform medical services.

The paper is structured as follows: We derive safety hazards and fault model of the tracheotomy in Section II. The open-

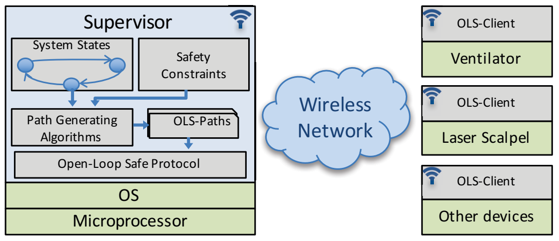

Fig. 1: The architecture of an open-loop safe cyber-physical medical system, where we integrate the Open-Loop Safe Protocol into the legacy supervisory control architecture of medical systems (e.g., OpenICE).

loop safe cyber-physical medical system is introduced and formalized in Section III. The algorithms to determine the system whether it is open-loop safe is presented in Section IV. A case study of the algorithms for the extended tracheotomy scenario and generic medical system setting is presented in Section V. We present the related work in Section VI and conclude the paper in Section VII.

II. Medical CPS and Medical Scenarios

We present a motivation system architecture based on MD PnP. Then, we present the Tracheotomy as a motivation example that adapts MD PnP for the supervisory control system, and its safety requirements and system fault models.

A. System Architecture

The motivated supervisory control architecture is similar to the Open-source Integrated Clinical Environment (OpenICE) [5]. Figure 1 depicts the architecture for the supervisory control systems. OpenICE is developed by the MD PnP program. The main goal of OpenICE is to connect network nodes (*e.g.,* medical devices, decision support system) of medical systems in the medical environment.

As presented in Figure 1, we define two roles of devices in an open-loop safe medical cyber-physical system (OLSSystem), an Open-Loop Safe Supervisor (OLS-Supervisor) and an Open-Loop Safe Client (OLS-Client). In an OLSSystem, it only has one OLS-Supervisor exist and could have at least one OLS-Client. The setting is similar to typical traditional architecture such as OpenICE that consists of one supervisor and multiple device adapters.

The OLS-Supervisor is designed to coordinate the medical devices to deliver medical services. It is the center of the control logic. The system designers can specify how the system should transit between different system states in the OLS-Supervisor. For example, the OLS-Supervisor in the tracheotomy can regulate the oxygen supply to avoid the patient suffering hypoxia.

On the other hand, an OLS-Client is an adapter that receives, reads and executes the control commands from the OLS-Supervisor through the communication networks. An OLS-Client is a middle layer serves as an interface between the OLS-Supervisor and a medical device. For example, if an OLS-Clientreceives a command to resume the oxygen supply from the OLS-Supervisor, the OLS-Client then performs the designated actions to resupply oxygen in the ventilator. Here, the goal is to minimize the functions in OLS-Client adapters in order to decrease the state space and hence the complexity in a distributed environment.

B. Airway Laser Tracheotomy

In the tracheotomy, a surgeon first needs to stop the oxygen flow of the *ventilator* to the patient with a mask on, and supply with plain air flow. Then, the surgeon operates a *laser scalpel* to unblock the airway of the patient. During the surgery, there are two safety requirements. First, the surgery should have no fire accident. That is, the laser scalpel should only operate on the patient when the ventilator is not supplying oxygen but plain air. It implies that the ventilator should be configured to stop the oxygen prior using laser scalpel. Second, the patient should not suffer hypoxia because of the late oxygen resupply. It implies that the oxygen should resupply in time to avoid hypoxia, and the laser scalpel operations should finish before the oxygen resupply or it will cause a surgical fire. Without an integrated medical system for such medical scenarios to coordinate multiple medical devices, the safety requirements are only achieved on a best-effort basis.

In the tracheotomy, the medical system has a ventilator to supply the oxygen or plain air, and a laser scalpel to emit laser. A detailed model is illustrated in [6], [3]. In a simplified and yet realistic model, both the ventilator and the laser scalpel can be modeled as binary-valued devices. A binary-valued device is a device with two statuses, and each status can transit to the other status directly. For the laser scalpel, "no-operation, 0" represents it does not emit the laser and "in-operation, 1" represents it emits the laser for operation. For the ventilator, "in-operation, 1" represents it supplies oxygen mixed with air and "no-operation, 0" represents it supplies plain air only.

III. System Models

We introduce a system model for an open-loop safe system. An open-loop safe system model is a three-tuple, $\langle \mathbf{S, SC, P} \rangle$ where S is a set of system states and each system state is a tuple of devices' status. For example, for a medical system with two binary-valued medical devices, S is a set of system states s. Each system state s is a n-tuple of devices' status $d_i.status$ where i is the *i-th* device in the system and $status \in \{0, 1\}$ and n is the number of devices.

Each system state s has a type which could be either *Open-Loop Safe State* (OLSState), *Transient Safe State* (TSState), *Operation State* (OState), or *UnSafe State* (USState). A system can stay at an OLSState permanently (in comparison to transient) and safely. For example, a patient with a ventilator supply oxygen is considered as an OLSState. A system staying at TSState is said to be safe and only allowed in the state for a certain period of time. Furthermore, a system staying at OState has the same properties as a TSState. The difference is that an OState is also the destination state of a series of state transitions to let medical personnel perform medical operations. Last, a system entering or staying at an USState is unsafe which the system shall avoid entering the state in any condition.

The second element in the system model is a set of safety constraints, SC. Each sc_i in SC is a two-tuple, $\langle state, period \rangle$, that allows a system staying at the given state safely for the specified period of time. On the other hand, for a sc, if the specified period of time is zero, it means that the system is not safe in the given state.

The third element in the system model is a set of open-loop safe paths, P. For each open-loop safe path $p \in P$, p has a source state, a destination state and a series intermediate states between the source and the destination state. The source state of p is an OLSState and the destination state is an OState. After the system reaches the destination state, the system will reside in the OState safely for a period of time which allows medical operations, then the system follows the reverse path and transits to the original source state. The intermediate states on a path are a series of TSState. Furthermore, a path should not have any USState. Here, we assume that, in an unit of time, the system should only transit to another state with distance one.

Any state transition between two immediate states should have state distance one. For example, in a two-binary-valued-devices system, a transition from a state $(0, 1)$ to $(1, 1)$ has the state distance one and it takes one unit of time to transit. Because it only requires the first device in the tuple to change the status from zero to one and the status of the second device remains unchanged. Furthermore, a transition from a state $(0, 1)$ to $(1, 0)$ has the state distance two and it takes two unit of time. Because it requires each device to change its own status and each change of status requires one unit of time.

IV. Finding an Open-Loop Safe System

In this section, we first present an algorithm to examine whether a given system model is open-loop safe and find out possible open-loop safe paths in a given set of system states and safety constraints. After finding out possible open-loop safe paths, if there are multiple open-loop safe paths in the system, we present an algorithm to select an open-loop safe path with the maximum period that a system can stay in the operation state to perform the designated medical task.

A. Determining an open-loop safe system

The first question to ask is whether exist one or more open-loop safe paths in a given medical system model. Because an open-loop safe path is a fundamental component that can ensure that the system remains open-loop safe while transitioning between states in the presence of communication failures. As such, by following an open-loop safe path, it guarantees that the system transitions meet its own safety constraints.

Figure 2 presents the work-flow toward developing an open-loop safe system. The work-flow includes two phases. First, with the system parameters listed in Figure 2, the phase I is to decide the existence of open-loop safe paths given a system

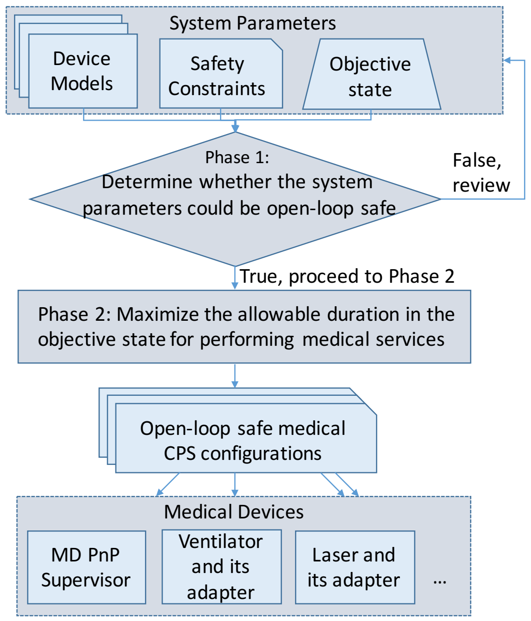
System Parameters
Device Models
Safety Constraints
Objective state
Phase 1: Determine whether the system parameters could be open-loop safe
False, review
True, proceed to Phase 2
Phase 2: Maximize the allowable duration in the objective state for performing medical services
Open-loop safe medical CPS configurations
Medical Devices
MD PnP Supervisor
Ventilator and its adapter
Laser and its adapter
...

Fig. 2: Workflow for open-loop safe system developments

model, safety constraints and the objection state. If there exists at least one open-loop safe path, then we move to the the phase II. The phase II is to find out the path that can allow a system to stay for the longest period at an operation state to perform the medical task. With the configurations for medical devices generated in phase II, a developer then configures the devices to transit safely and against communication failures. Here, we focus on the phase I of the work-flow. We will return to introduce the second phase in Section IV-B.

In the phase I, the first step is to construct an undirected weighted graph based on the given system states and safety constraints. With the graph, next, we use the shortest path algorithm to find out the shortest path and compare the length of the path with the state distance between the source and the destination state.

Constructing a System Graph Algotihm 1 presents the phase I. We first construct a graph based on the given system states and safety constraints in L:4 (Line 4 in Algorithm 1) with the given system model. A system graph G is a graph representation of a system that incorporates with system states, system state types, and safety constraints as defined in Section III. We name the graph, *system graph* $G = (V, E)$, where V is a set of system states in the given system model M and represents a set of vertexes in G, and E is a set of weighted and undirected edges in G. Each vertex $v \in V$ in the system graph G represents a relative system state in a given system model M. After labeling each vertex with the corresponding system state, we connect the two adjacent vertexes that have state distance one. As a result, for every $v \in V$ in G, each vertex only connects to its adjacent vertex(es) with state distance one. Next, we set up the state type of each vertex according its system state in L:5.

To assign an integer weight of an edge, we label the weight of each edge $e_{ij} \in E$ based on the state type of the two connecting vertexes $v_i, v_j \in V$ and the safety constraints in L:6. The intuition of labeling edges with different weight is that we will use this weighted graph as the input for the shortest path algorithm later. For each edge e, we assign the weight of the edge e as either one or infinite.

More specifically, for each vertex v labeled with USState, we label the weight of each edge connecting to the vertex v with infinite weight. For any other edges in the graph, we label the weight of the edge with one. Since only unsafe transitions to or from a vertex v labeled with USState have infinite weight, the weighted shortest path algorithm avoids selecting these edges. To this end, we finish constructing a system graph of a given system. We are ready to use this system graph as an input for the weighted shortest path algorithm to find out open-loop safe paths.

Finding Open-Loop Safe Paths With the generated system graph, we use the weighted shortest path algorithm to find out the shortest path from the source state, *i.e.,* an open-loop safe state, to the destination vertex, *i.e.,* an operation state. This step is presented in L:7 in Algorithm 1. Since we have labeled all the edges connected to each unsafe state vertex with infinite weight, and all other edges with weight one, when the shortest path algorithm selects an edge, it attempts to select the edge with weight one which avoid selecting the edges connected to the unsafe state vertexes.

After the algorithm finds out the shortest path, we then compare the summation of weights along the path with the state distance between the source and the destination state in L:8. Here, we assume the algorithm finds only one shortest path in a given system graph. Since the weight of each edge is not unique, so there could be multiple paths with the same weight. We will discuss the case if there are multiple paths in a system shortly. On the other hand, if the total weight of the found path is larger than the state distance between the two states, then it means the algorithm selects more edges than necessary to transit form the source state to the destination state. Therefore, we can conclude that there does not exist an open-loop safe path in the given system model.

Furthermore, the total weight of the found path will never be smaller than the state distance between the two states. Because the state distance between the two states is a minimum number of required state change. So, with the given graph, the minimum weight of a path for a source state to reach the destination state is as the same as the state distance between the two states.

Finally, if the weight of the found path has the same state distance between the two states, then it means the algorithm selects the shortest path between the source and the destination states. We can proceed to the next step to find out the period of time for each device to stay at the certain status.

As mentioned earlier, the algorithm may find out more than one path with the same weight between the source and the destination states. For example, when the algorithm selects one of two outgoing edges of a vertex. If there are two outgoing

ALGORITHM 1: Determine if a system has an open-loop safe path

```
1  Input: States $S$ and safety constraints $SC$ of the system model $M$;
2  Output: List of possible system transition paths;
3  begin
4      System graph $G = (V, E) \leftarrow$ setSystemGraphVertex(M.states);
5      G $\leftarrow$ G.setVertexType(M.states.types);
6      G $\leftarrow$ G.setEdgeWeight();
7      M.paths $\leftarrow$ G.getShortestPath(M.safetyConstraints);
8      M.paths $\leftarrow$ removePathWithWeightLargerThanStateDistance
           (sourceState, destinationState, M.paths);
9      if Size of M.paths equals zero then
10         return No path found;
11     else
12         return M.paths;
```

edges have weight equal one, then the algorithm might select either one of them. In this case, we need to determine which one of the found paths may have the longest period to stay at the operation state which allows more time for medical personnel to perform medical tasks. In the next section, we will introduce the algorithm to select the path of interest.

B. Finding the Optimum Open-Loop Safe Path

With the definition of transient safe in Section III, we can now define a *transient safe period (TSP)* is a period of time that a device can stay at the certain status so that the whole system remains temporarily safe. A TSP for a device can be configured as a timer by an OLS-Client adapter so that the adapter on the medical device can change the device's status when the timer starts and resume the device's status after the timer with TSP configured expires.

We start from calculating the TSP configurations for the ventilator and the laser scale in the tracheotomy as an motivation example. After that, we will generalize the algorithm for period configurations for multiple devices. With the found path from Algorithm 1, we can find the TSP on each device according to the specified safety constraints. If there are multiple paths, after finding out the period of each timer on one path, we then sort the paths in non-increasing order of the period that the system can stay in an operation state. We choose the path with the longest period that the system can stay in an operation state.

Figure 3 presents the found path transitioning from an OLSState through a TSState to an OState and rolling back to the same OLSState. The ventilator is the first device to change its status. Assuming the safety constraint on ventilator suggests the safe period of pausing oxygen supply is $p_{vent.off.max}$. The safe period for the ventilator pausing oxygen supply is $d_{vent.off.timer} = d_{vent.off.max}$. The ventilator remains off (*i.e.,* 0) from t_v to $t_{v'}$ and change its status to on (*i.e.,* 1) at $t_{v'}$.

The laser scalpel is the second device to change its status, and we need to configure the timer for the laser scalpel such

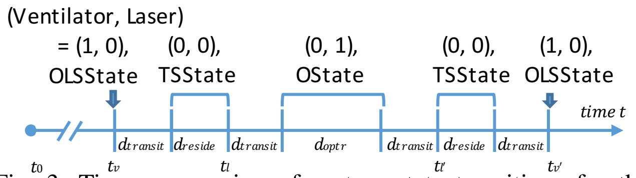

Fig. 3: Time progression of system state transitions for the extended Tracheotomy scenario

that it does not violate the safety constraint causing surgical fire. Assume each state transition takes the maximum duration d_{tran} and the system resides in a TSState for the maximum duration d_{resd}. From Figure 3, we derive the safe duration to avoid hypoxia is below.

$$2 \cdot 2d_{tran} + 2 \cdot d_{resd} + d_{ops} \le d_{vent.off.max} \tag{1}$$

And we can derive the maximum safe period d_{ops} for the laser scalpel operation

$$d_{ops} \le d_{vent.off.max} - 2(2\dot{d}_{tran} + d_{resd}) \tag{2}$$

As such, the TPS for the laser scalpel operations is

$$\begin{aligned} d_{laser.on.timer} &= d_{ops} + 2d_{tran} & (3)\\ &= d_{vent.off.max} - 2(d_{tran} + d_{resd}) & (4)\end{aligned}$$

The timer for the laser starts at the time t_l and fires at $t_{l'}$. And since $d_{vent.off.timer} = d_{vent.off.max}$, we can further derive $d_{vent.off.timer}$ as:

$$d_{laser.on.timer} = d_{vent.off.timer} - 2(d_{tran} + d_{resd}) \tag{5}$$

In the above mentioned example, the duration of the laser timer $d_{laser.on.timer}$ is limited by the duration of ventilator $d_{vent.off.timer}$ to resume the oxygen. We further extend this example to add another safety constraint that limits the continuously operation time of the laser scalpel to protect the medical device. The maximum duration for the laser scalpel to operation is given as $d_{laser.on.max}$. As such, the duration for the laser timer in Equation 5 is as follows.

$$\begin{aligned} d_{laser.on.timer} = \min(&d_{vent.off.timer} - 2 \cdot 2d_{tran} - 2 \cdot d_{resd},\\ &d_{laser.on.max}) \end{aligned} \tag{6}$$

Specifically, we present the algorithm of finding the TPS for each device in Algorithm 2. The idea is that we gradually shrink the timer duration of each device along a path. If a device also has a safety constraint specifying the limited period of time staying in the certain status, then we take it into account when calculating the TSP for the device. First, after running Algorithm 1 mentioned above, we retrieve pathCandidates from the system model M in L:4 in Algorithm 2. For each path in the pathCandidates, the algorithm extracts the devices that need to change its status. As mentioned earlier, in each unit of time, there is only one device changes its status.

ALGORITHM 2: Finding the path with longest duration in the operation state

```
1  Input: A system model
      $M = \langle States, SafetyConstraints(SC), Paths\rangle$ with the
      paths generated from Algorithm 1
2  Output: An open-loop safe path
3  begin
4      pathCandidates ← M.Paths;
5      constant $d_{tran}$, $d_{resd}$ duration for transitioning and
         residing;
6      for each path p in pathCandidates do
7          deviceList ← getDevicesToBeConfigured($p$);
8          for each device $m_i$ in deviceList do
9              /* Shrink the timer duration from the previous
                 device */
10             if $m_i$ has safe operation time constraint in M.SC
                 OR
11             $m_{i-1}$ has the timer configured then
12                 timer duration $d_{m_i}$ ← min(
                     $d_{m_i-1} - 2(d_{tran} + d_{resd})$, maximum safe
                     duration on $m_i$ specified in M.SC);
13                 $m_i$.setTimer($d_{m_i}$);
14             else
15                 mark $m_i$ as unhandled;
16             $i++$; /* Move to the next device */
17         /* Reversely expand the timer duration from the next
             device, starting from $m_i$ where it is immediate
             preceding to the first device has timer configured */
18         for each device $m_i$ marked as unhandled in deviceList
             do
19             $d_{m_i} \leftarrow d_{m_i+1} + 2(d_{tran} + d_{resd})$;
20             $m_i$.setTimer($d_{m_i}$);
21             $i--$; /* Move to the previous device */
22         $d_{ops} \leftarrow d_{lastDeviceWithTimerConfigured} - 2d_{tran}$ ;
23         $p$.add($d_{ops}$) ;
24     sortPathsByDurationInOperationState(pathCandidates);
25     return the path with largest $d_{ops}$;
```

So, for each path in the pathCandidates, we can retrieve an ordered list of devices where each device will change it status sequentially as in L:7.

We present the TPS configurations from L:8 to L:23 in Algorithm 2. First, if a device has a safety constraint specified in the system model or the preceding device has the timer configured, then we calculate the TSP for the current device. Otherwise, we mark the device as *unhandled* and will handle it later in L:18. For the current device, set the TSP of the device to the minimum of the safe period to stay at its current status from the safety constraint, or the timer duration from the previous device, d_{m_i-1}, minus the two transitioning time and residing time, $2(d_{tran} + d_{resd})$. To see that why we need to subtract the the two transitioning time and residing time, Figure 3 gives the visual illustration. Each system transition to a state takes one transitioning time (d_{tran}) and one residing time (d_{resd}) at the other state. For example, the timer duration for the laser scalpel is $d_{t_l,t'_l} = d_{t_v,t'_v} - 2(d_{tran} + d_{resd})$ which shortens the timer duration for the ventilator by $2(d_{tran} + d_{resd})$.

To handle the device(s) that hasn't had timer configuration yet, we handle it in L:18. Now, assume the first *i* devices in the deviceList do not have its TSP calculated. We then calculate the TSP for the *i-th* to the first device in the deviceList. Similar to the timing duration shrinking in L:11, we now need to expand the TSP from the *i+1-th* device for the *i-th* device to accommodate the two times state transition and residing time. Hence, the timer duration d_{m_i} equals the timer configuration for the next device $d_{m_{i+1}}$ plus the two times $d_{tran} + d_{resd}$ in L:18. Then, we calculate the TSP of the *i-1-th* device until finish calculating the TSP of the first device in the deviceList.

After calculating TSP for every device on the path, we then calculate the duration for safely performing medical operation d_{ops} by subtracting the two transition time from the TSP of the last device as described in L:22. Finally, we sort the path with non-increasing order of the medical operation time d_{ops} in every path and return the path with longest medical operation time.

V. Case Study

We further extend the tracheotomy example. We add another device model to represent the ventilator supplying plain air. The new device state model $s \in S$ is a three tuple $(d_{oxygen}, d_{plainAir}, d_{laser})$, where the first element in the tuple represents that oxygen is supplying when the value is one and pausing the oxygen when the value is zero. The second element in the tuple is the status of plain air supply where the value one represents the device is providing plain air otherwise the value is zero. The third element in the tuple is the status of laser scalpel which has the same notion as before.

With the two original requirements of no hypoxia and no surgical fire, we add additional two safety requirements. The first requirement is the laser scalpel can only operate safely continuously within a period of time. The second new requirement is that once the oxygen supply is paused, it is required to enable the plain air supply.

We list the new system states in Table I. In the new system states, both USStates and TSStates increase from one on each to three states on each. With the system states and the safety requirements, we then construct the system graph in Figure 4 and label the weight of each edges based on the connecting system states at two ends according to Algorithm 1.

Next, with the system graph, we can use the path finding algorithm in Algorithm 1 and TSP calculations described in Algorithm 2 to select a path with the longest time staying in the OState. Figure 5 presents the two open-loop safe paths with the transient safe period staying in each system state. There are two potential paths from the source OLSState $(1, 0, 0)$ to the destination OState$(0, 1, 1)$. Each path has the same weight of three. The weight equals the distance between the OLSState $(1, 0, 0)$ and the destination OState$(0, 1, 1)$. The first path is $P_1 = (1, 0, 0) \rightarrow (0, 0, 0) \rightarrow (0, 1, 0) \rightarrow (0, 1, 1)$ where the oxygen is paused and then the plain air is supplying as shown in the top time-line in Figure 5. The second path is $P_2 = (1, 0, 0) \rightarrow (1, 1, 0) \rightarrow (0, 1, 0) \rightarrow (0, 1, 1)$ where the oxygen supply is paused after the oxygen supply is enabled as shown in the bottom time-line in Figure 5. As shown in Figure 5, P_2 has larger d_{ops} than P_1 by saving two transition time d_{tran} and two residing time d_{resd}. Because, in P_1, the oxygen is paused after the plain air is enabled such that it allows more time for the system to transit to and stay at the OState to perform surgeries.

TABLE I: System states with types for the extended Tracheotomy scenario

State Types	System states $(d_{oxygen}, d_{plainAir}, d_{laser})$
OLSState	(1,0,0)
USState	(1,1,1), (1,0,1), (1,1,0)
OState	(0,1,1)
TSState	(0,0,1), (0,1,0), (0,0,0)

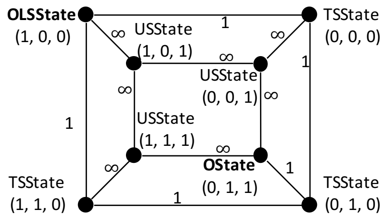

Fig. 4: System graph (Undirected Acyclic Graph) with weight edges, system states, and system types for the extended Tracheotomy scenario.

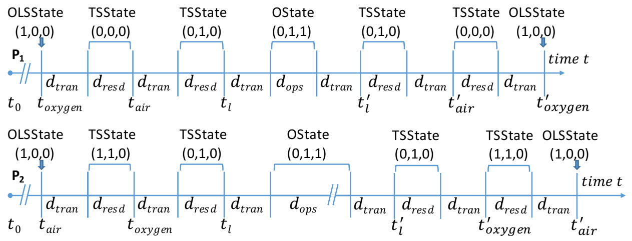
OLSState (1,0,0)
TSState (0,0,0)
TSState (0,1,0)
OState (0,1,1)
TSState (0,1,0)
TSState (0,0,0)
OLSState (1,0,0)
P1
time t
d_{tran} d_{resd} d_{tran} d_{resd} d_{tran} d_{ops} d_{tran} d_{resd} d_{tran} d_{resd} d_{tran}
t_0 t_{oxygen} t_{air} t_l t'_l t'_{air} t'_{oxygen}
OLSState (1,0,0)
TSState (1,1,0)
TSState (0,1,0)
OState (0,1,1)
TSState (0,1,0)
TSState (1,1,0)
OLSState (1,0,0)
P2
time t
d_{tran} d_{resd} d_{tran} d_{resd} d_{tran} d_{ops} d_{tran} d_{resd} d_{tran} d_{resd} d_{tran}
t_0 t_{air} t_{oxygen} t_l t'_l t'_{oxygen} t'_{air}

Fig. 5: The two open-loop safe paths. The top time-line is for the first path (P1) and the bottom one is P2

VI. Related Work

Close-loop control systems [7], [8], [9], [10], [11], [12] for medical use while ensuring safety properties such as glucose control systems for diabetes management and patient-controlled analgesia (PCA) infusion have been well studied based on formal verfication tehniques [13], [14], [15], [16]. The work [12] presents a model-driven approach for building a close-loop cyber-physical medical system while guarantees the safety properties. In the paper [12], Pajic *et al.*also discusses the fail safe design for dealing with an open-loop safe problem in PCA when the supervisor does not receive the expected value for HR or SpO_2. However, it is not clear how the proposed method can be adapted to a system with more than one network-connected medical device to coordinate the task and has failsafe design against communication failures.

With the increasing interest of adapting communication networks in medical systems, researchers pay noticeable efforts to address the open-loop safe problem caused by communication failures [17], [2], [18], [19], [20], [21] with enhanced system architecture. For example, the papers [2], [18] propose an architecture named Network-Aware Supervisory System (NASS) based on the OpenICE architecture to safely integrate networked medical devices. The key idea is to keep a copy of execution sequences in each device called a vector; the vector contains a plan of the following actions performed on the device for future cycles in the case of a network failure of packet losses. However, it is not clear how to generalize the framework to a scenario with multiple devices and safety constraints.

Our work focuses on providing a series of open-loop safe system transitions as a foundation to generalize an open-loop safe supervisory system with multiple medical devices, and incorporate safety requirements of interactions between devices. Furthermore, our proposed algorithms aim to select a system transition path that can allow the medical personnel with the longest operation time for performing surgeries.

VII. Discussion and Conclusion

We present a work-flow toward building an open-loop safe system in medical CPSs. Specifically, we provide an algorithm to determine whether a given system-of-interest model could be open-loop safe in the presence of communication failures and an algorithm to find a state transition path that has the longest duration staying at the operation state for medical personnel to perform medical operations.

In this paper, we do not address the question about the communication protocol between the supervisor and medical devices. With the open-loop safe paths generated by the proposed algorithms in this paper, we are able to design an open-loop safe protocol that based on the system transition paths for networked connected medical devices to coordinate for the medical tasks. Currently, our assumption of system state is solely based on devices' statuses. In some medical cases, the system model could vary because of the situation changes of the patient. This research challenge should be addressed in the future work.

Acknowledgment

This work is supported in part by NSF CNS 1329886, by NSF CNS 1545002 and in part by ONR N00014-14-1-0717.

References

[1] J. Goldman, J. Jackson, S. Whitehead, T. Rausch, and S. Weininger, "The medical device "plug-and-play" (md pnp) interoperability program," *COMPUTER*, vol. 39, no. 4, pp. 30–31, 2006.

[2] C. Kim, M. Sun, S. Mohan, H. Yun, L. Sha, and T. F. Abdelzaher, "A framework for the safe interoperability of medical devices in the presence of network failures," in *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*. ACM, 2010, pp. 149–158.

[3] F. Tan, Y. Wang, Q. Wang, L. Bu, R. Zheng, and N. Suri, "Guaranteeing proper-temporal-embedding safety rules in wireless cps: A hybrid formal modeling approach," in *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*. IEEE, 2013, pp. 1–12.

[4] F. Tan, Y. Wang, Q. Wang, L. Bu, and N. Suri, "A lease based hybrid design pattern for proper-temporal-embedding of wireless cps interlocking," *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[5] J. Plourde, D. Arney, and J. M. Goldman, "Openice: An open, interoperable platform for medical cyber-physical systems," in *Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on*. IEEE, 2014, pp. 221–221.

[6] T. Li, F. Tan, Q. Wang, L. Bu, J.-n. Cao, and X. Liu, "From offline toward real-time: A hybrid systems model checking and cps co-design approach for medical device plug-and-play (mdpnp)," in *Cyber-Physical Systems (ICCPS), 2012 IEEE/ACM Third International Conference on*. IEEE, 2012, pp. 13–22.

[7] D. Bruttomesso, A. Farret, S. Costa, M. C. Marescotti, M. Vettore, A. Avogaro, A. Tiengo, C. Dalla Man, J. Place, A. Facchinetti *et al.*, "Closed-loop artificial pancreas using subcutaneous glucose sensing and insulin delivery and a model predictive control algorithm: preliminary studies in padova and montpellier," *Journal of diabetes science and technology*, vol. 3, no. 5, pp. 1014–1021, 2009.

[8] E. Cengiz, K. L. Swan, W. V. Tamborlane, G. M. Steil, A. T. Steffen, and S. A. Weinzimer, "Is an automatic pump suspension feature safe for children with type 1 diabetes? an exploratory analysis with a closed-loop system," *Diabetes Technology & Therapeutics*, vol. 11, no. 4, pp. 207–210, 2009.

[9] R. Hovorka, "Continuous glucose monitoring and closed-loop systems," *Diabetic medicine*, vol. 23, no. 1, pp. 1–12, 2006.

[10] M. J. O'grady, A. J. Retterath, D. B. Keenan, N. Kurtz, M. Cantwell, G. Spital, M. N. Kremliovsky, A. Roy, E. A. Davis, T. W. Jones *et al.*, "The use of an automated, portable glucose control system for overnight glucose control in adolescents and young adults with type 1 diabetes," *Diabetes Care*, vol. 35, no. 11, pp. 2182–2187, 2012.

[11] D. Arney, M. Pajic, J. M. Goldman, I. Lee, R. Mangharam, and O. Sokolsky, "Toward patient safety in closed-loop medical device systems," in *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*. ACM, 2010, pp. 139–148.

[12] M. Pajic, R. Mangharam, O. Sokolsky, D. Arney, J. Goldman, and I. Lee, "Model-driven safety analysis of closed-loop medical systems," *Industrial Informatics, IEEE Transactions on*, vol. 10, no. 1, pp. 3–16, 2014.

[13] Y. Jiang, Y. Yang, H. Liu, H. Kong, M. Gu, J. Sun, and L. Sha, "From stateflow simulation to verified implementation: A verification approach and a real-time train controller design," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016 IEEE*. IEEE, 2016, pp. 1–11.

[14] Y. Jiang, H. Zhang, Z. Li, Y. Deng, X. Song, M. Gu, and J. Sun, "Design and optimization of multiclocked embedded systems using formal techniques," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 2, pp. 1270–1278, 2015.

[15] Y. Yang, Y. Jiang, M. Gu, and J. Sun, "Verifying simulink stateflow model: timed automata approach," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2016, pp. 852–857.

[16] Y. Jiang, H. Liu, H. Song, H. Kong, M. Gu, J. Sun, and L. Sha, "Safety-assured formal model-driven design of the multifunction vehicle bus controller," in *FM 2016: Formal Methods: 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings 21*. Springer, 2016, pp. 757–763.

[17] W. Kang, L. Sha, R. B. Berlin, and J. M. Goldman, "The design of safe networked supervisory medical systems using organ-centric hierarchical control architecture," 2014.

[18] C. Kim, M. Sun, M. Rahmaniheris, and L. Sha, "How to reliably integrate medical devices over wireless," in *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2012 9th Annual IEEE Communications Society Conference on*. IEEE, 2012, pp. 85–87.

[19] A. Y.-Z. Ou, Y. Jiang, P.-L. Wu, L. Sha, and R. B. Berlin, "Preventable medical errors driven modeling of medical best practice guidance systems," *Journal of Medical Systems*, vol. 41, no. 1, p. 9, 2017.

[20] Y. Jiang, H. Song, R. Wang, M. Gu, J. Sun, and L. Sha, "Data-centered runtime verification of wireless medical cyber-physical system," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1900–1909, 2017.

[21] Y. Jiang, H. Liu, H. Kong, R. Wang, M. Hosseini, J. Sun, and L. Sha, "Use runtime verification to improve the quality of medical care practice," in *Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on*. IEEE, 2016, pp. 112–121.