Safety-Ensured Coordination of Networked Medical Devices in MDPnP

Tao Li, and Jiannong Cao
Department of Computing
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
Email: {cstaoli,csjcao}@comp.polyu.edu.hk

***Abstract*—In recent years, research on medical device integration has attracted a lot of attentions because of its great potential to improve treatment safety and efficiency. The leading role in this research is the Medical Device Plug-and-Play (MDPnP) Interoperability program. The continuous effort of the program has led to the publication of the Integrated Clinical Environment (ICE) architecture which defines the functional components for an integrated medical system. However, while applying the architecture to enable *medical device coordination* in the realistic clinical settings, there still exist many issues, such as device heterogeneity, unreliable communications, etc. To address these issues, we aim at building a middleware (called *VirLoop*) which can provide rich services to ease this task. The prominent features of VirLoop are as follows. i) It considers the realistic network condition in the clinical environment, where communications are unreliable and asynchronous. ii) It enables automatic medical device composition to build an integrated system for easy coordination. iii) It creates a *virtually perfect network environment* for medical device coordination, so that development of the coordination logic is independent of the underlying network complications. iv) Safety can be ensured while invoking the middleware services. We build VirLoop on some emulated medical devices to test its safety property and performance. Trace-based emulation results show that VirLoop not only ensures safety, but also achieves high clinical efficiency, low response time and low communication overhead.**

I. Introduction

The past decades have witnessed the appearance and popularity of a variety of plug-and-play (PnP) technologies in pervasive computing environment [1], such as Universal Plug and Play (UPnP), Jini, Salutation. The primary goal of these technologies is to hide the underlying device heterogeneity and complications, as well as the environmental changes. Hence, the upper-layer user applications can continuously enjoy the services in the environment without being distracted.

Recently, the PnP concept has been introduced into the clinical area. This idea originates from the fact that in the clinical procedures, medical devices are commonly configured and controlled by human beings who are prone to generate medical errors [2]. As revealed in [2], medical errors are the leading cause of death and injury in the hospital. This situation raises the demand of medical device plug-and-play and automatic control to prevent the human error-induced accidents. This increasing demand has led to the establishment of the Medical Device Plug-and-Play (MDPnP) Interoperability program [3], which developed the first MDPnP standard. The standard proposes an *Integrated Clinical Environment (ICE)* architecture (Fig. 1a), which defines the functional components in an integrated device system. In particular, a controller, called *supervisor*, has been introduced to coordinate medical devices.

In this paper, we call the ICE-compliant medical device system for a particular clinical procedure as a *MDPnP system*. MDPnP systems are *safety-critical* in general, i.e., patient safety is of paramount importance. It is not acceptable for patients to be harmed by the MDPnP system in any circumstances. As the central controller, it is critical for the supervisor to run certain medical device coordination mechanisms to ensure safety and enable medical treatment. However, developing safe device coordination mechanism is non-trivial. It usually requires knowledge and collaboration of experts from multiple domains, such as clinical medicine, control theory and distributed systems. The ICE architecture standard [4] illustrates several effective coordination methods, like safety interlock and physiological closed-loop control, by concrete clinical scenarios [4]. But these scenarios all assume perfect inter-device communications. In reality, communications in the open clinical environment are *unreliable and asynchronous*. The increasing adoption of wireless medical devices (especially medical sensors) and occasional disconnection of cables can result in message loss [5][6][7]. Retransmission and network congestion may also lead to variations of message transmission delay.

Therefore, while applying the ICE architecture to the realistic clinical scenarios, it is necessary to take into account the non-perfect communications while designing device coordination mechanism. Furthermore, device heterogeneity is another big obstacle hindering medical device coordination. The majority of of existing medical devices are designed for isolated use and lack interoperability. Both semantic-level data inconsistency and communication protocol difference among devices can lead to failure of coordination, which compromises the safety. In this paper, we aim at building a middleware, called *VirLoop*, that can provide safety-assured support for general MDPnP scenarios. The design principle is that we try to make the device heterogeneity and communication problems transparent to the device coordination mechanisms. Hence, the coordination logic will be simplified, and thus become easy to design, check and verify. We summarize the prominent features of VirLoop as follows.

- VirLoop can automatically composes medical devices to an integrated system for easy device coordination, regardless of device heterogeneity.
- VirLoop is able to hide communication complications from the device coordination mechanisms. It provides a *virtually perfect network environment* for them.
- The provided middleware services are proved to be safe even in the cases when messages are lost and suffer long delay.

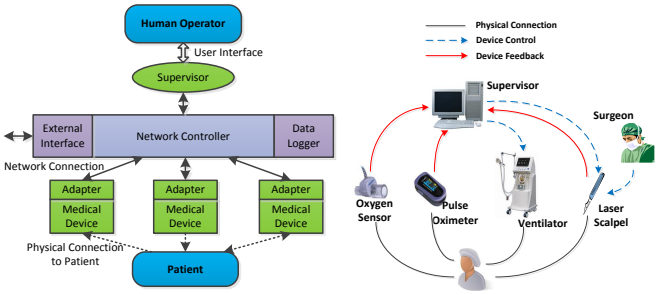
To evaluate the middleware, we build the middleware on some emulated medical devices, and construct a well-known MDPnP scenario, airway-laser surgery. Trace-based emulation results confirm that safety can be guaranteed by using the VirLoop services. In addition, we test the performance of VirLoop by contrasting it with a state-of-the-art device coordination framework, NASS [5], which can tolerate network failures. The results show that our middleware achieves high clinical efficiency, and drastically outperforms NASS in terms of response time and communication cost (with maximum decease by a factor of 10 and 100, respectively).

The reminder of the paper is organized as follows. Related work is discussed in Section II. Section III introduces system architecture, device coordination and interaction model. Middleware services are elaborated in Section IV, V and VI respectively. Then, in Section VII we evaluate the middleware. Finally, Section VIII concludes the paper with future research directions.

II. Related Work

The MDPnP Interoperability program [3] was launched with the purpose of promoting medical device interoperability, safe integration, as well as seeking for regulatory paths for system approval. The program proposed an architecture to create Integrated Clinical Environment (ICE) [4]. Major components of an ICE architecture are shown in Fig. 1a. This architecture is an standard model to follow while designing medical device coordination mechanisms, and has motivated a lot of work on safe coordination mechanism design. Li *et al.* propose to use run-time model checking to enhance the safety of device coordination [8]. King *et al.* [9] build a Medical Device Coordination Framework (MDCF) to enable model-based development of medical device coordination applications. However, both works neglect the problem of unreliable and asynchronous communication in the design.

Actually, so far there has been few work considering the network problems in designing device coordination mechanism. Arney *et al.* [10] proposed an UPPAAL-verified safe control mechanism to tolerate communication failures. But its limitation is that it requires an accurate patient model, and is specifically designed for patient-controlled analgesia scenario. NASS [5] and its successor PVSC [6] are two coordination frameworks that can ensure safety under communication failures. They rely on periodic supervisor-medical device interaction model, resulting in long response delay and high communication cost. Another drawback is that it is difficult to determine the proper cycle length for NASS and



(a) ICE architecture (b) Airway-laser MDPnP system

Fig. 1: ICE architecture and ICE-compliant MDPnP system

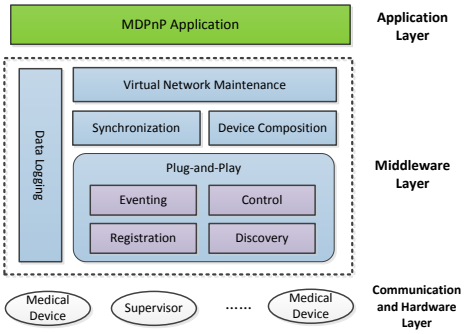


Fig. 2: System architecture

PVSC. However, our VirLoop middleware is applicable for more general MDPnP scenarios and overcomes the limitations of NASS and PVSC, as will shown later.

III. System Overview

A MDPnP system consists of a supervisor and a collection of medical devices that are connected through network. In general, medical devices can be categorized into two groups: *monitoring devices* and *delivery devices* [11]. The first group, such as heart rate and blood pressure sensors, monitors patient's physiological state or physical environment parameters. The supervisor is loaded with certain device coordination mechanism for a specific clinical procedure. In what follows, we will use *device* to refer to either a medical device or a supervisor.

A. System Overview

The architecture of a MDPnP system running VirLoop is shown in Fig. 2. The application layer runs the medical device coordination mechanisms to maintain safety for clinical scenarios. VirLoop middleware provides services to support the application. The middleware services are listed as follows.

- Plug-and-Play (PnP) service: it deals with the medical device heterogeneity problem. It enables device registration, discovery, control, and eventing. Based on PnP service, we then can build more sophisticated services.
- Synchronization service: it maintains a global clock for the system.
- Device Composition service: it composes the individual medical devices to an integrated system to enable device coordination.

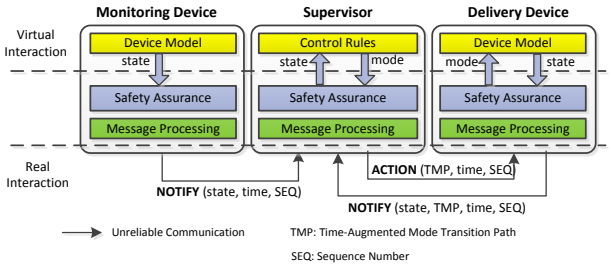


Fig. 3: Device interaction for medical device coordination

- Virtual Network Maintenance service: it ensures safety of medical device coordination, and maintains a virtually perfect network environment for the application layer.
- Data Logging service: it helps to log the critical device status and interaction behavior for online/offline analysis and forensic purposes.

The prerequisite of device coordination is that the required medical devices can interoperate and be composed together as an integrated system. Specifically, before medical device coordination phase, there are three steps that happen automatically by running VirLoop: i) device registration and discovery, ii) time synchronization, ii) device composition. Actually, VirLoop provides full support these steps.

B. Device Coordination Model in Perfect Network

Here, we introduce the device coordination model assuming ideal device interoperability and a *perfect network environment* where there is no message loss and transmission delay. A MDPnP system is modeled as a 4-tuple $Sys = \{\mathcal{S}, \mathcal{D}, \mathcal{H}, \mathcal{R}\}$. $\mathcal{D}$ is the set of medical devices. $\mathcal{S}$ is the supervisor that coordinates the medical devices. For each medical device $d \in \mathcal{D}$, $d = \{State^d, \xrightarrow{s}, Mode^d, \xrightarrow{m}, mode_0^d, Space^d\}$, where $State^d$ is the set of *device states* (states, for short); $\xrightarrow{s} \subset State^d \times State^d$ is a relation denoting the set of valid state transitions; $Mode^d$ is the set of supervisory modes (modes, for short) specified by the supervisor for d; $\xrightarrow{m} \subset Mode^d \times Mode^d$ is a relation denoting the set of valid mode transitions; $mode_0^d \in Mode^d$ is d's initial mode; and $Space^d$ is a function that defines the operational state space for each mode. When d is assigned with mode $mode^d \in Mode^d$ by the supervisor, it is restricted to operate within $Space^d(mode^d)$. On the contrary, state transition within $Space^d(mode^d)$ is free to happen. It should be noted that mode transition implies state transition, that is, a medical device is forced to change its state if mode transition happens. Besides, since monitoring devices keep measuring the physical parameters, we consider them always staying in the SAMPLING mode.

$\mathcal{H}$ is the set of hazards. A hazard $h \in \mathcal{H}$ is the combination of states of a medical device subset $\mathcal{D}^h$, where $\mathcal{D}^h \subseteq \mathcal{D}$, i.e., $h = \bigwedge_{d_i \in \mathcal{D}^h} state_h^{d_i}$. If all medical devices $d_i \in \mathcal{D}^h$ enter state $state_h^{d_i}$ simultaneously, then hazard h will be generated. Essentially a hazard h is a conjunctive predicate of the device states. There have been a lot of work on conjunctive predicate detection in distributed and pervasive environment, e.g., [12][13], but we differ from them in that our goal is conjunctive predicate avoidance. Actually, the safety interlock mechanism proposed in ICE standard [4] shares the similar idea with the hazard definition. A safety interlock links multiple devices to prohibit them from entering hazardous states simultaneously. Note that if multiple medical devices stay in their initial modes, no hazards must be yielded. Otherwise, it is nonsense that the initial system configuration is unsafe.

In essence, hazards are risky system states. The supervisor $\mathcal{S}$ runs a set of supervisory control rules (or control rules, for short), denoted by $\mathcal{R}$, to coordinate medical devices to circumvent the hazards. A control rule $r \in \mathcal{R}$ consists of a *condition field* and a *decision field*, i.e.,

$$\Big(\bigwedge_{d_i \in \mathcal{D}^r} state_r^{d_i}\Big) \wedge cmd^r \Longrightarrow mode_r^{d_r}. \tag{1}$$

The condition field defines the states for a medical device subset $\mathcal{D}^r$ as well as the input command cmd^r from the caregivers. If the condition field is satisfied, the supervisor will execute the rule to designate a new mode $mode_r^{d_r}$ to device d_r. In the case of automatic control, caregiver's intervention is not needed. So, Equ. 1 can be simplified as follows.

$$\bigwedge_{d_i \in \mathcal{D}^r} state_r^{d_i} \Longrightarrow mode_r^{d_r}. \tag{2}$$

C. Device Interaction in Perfect Network

Fig. 3 shows the *virtual interaction* and *real interaction* happening above and beneath the middleware layer during device coordination phase, respectively. The virtual interaction demonstrates the application layer's behavior by considering a perfect underlying network. As there is no need to update monitoring devices' modes (they always operate in SAMPLING mode), the supervisor only computes the modes for delivery devices. Each delivery device follows the mode from the supervisor to operate and treat the patient, leading to the change of patient's state. Then, the feedback information, containing the up-to-date states from the monitoring devices and the delivery devices, are sent back to the supervisor. The feedback state information may trigger certain control rules, and then the supervisor will execute the rules to update the modes of the delivery devices. In this way, control loop in the system is closed from the application layer's point of view.

D. MDPnP System Example

This subsection introduces a well-documented MDPnP system, airway-laser MDPnP system [5][8] to illustrate the device coordination model. The airway-laser MDPnP system (see Fig. 1b) involves following entities: surgeon, patient, O_2 sensor, pulse oximeter, ventilator, and supervisor. The application context is as follows. In the surgery, due to general anesthesia, the patient is paralyzed, hence has to depend on the ventilator to breathe. The surgeon requests the laser scalpel to emit laser so as to cut patient's trachea. When the laser scalpel is to emit laser, the oxygen concentration inside the trachea (measured by O_2 sensor) must be lower than a threshold, i.e., $\Theta_{O_2} = 30\%$. Otherwise, the laser may trigger *surgical*

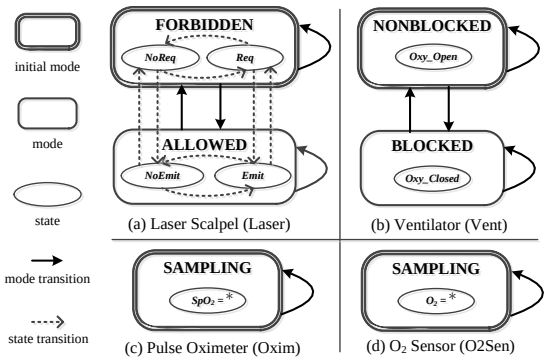


Fig. 4: System model for airway-laser MDPnP system ($*$ represents any valid sampling result)

fire, which can seriously burn the patient. Therefore, before the laser scalpel is allowed to emit laser, the ventilator must have blocked the oxygen flow for a while. On the other hand, the ventilator can neither block oxygen flow for too long, or the patient will *suffocate* due to too low blood oxygen level (SpO_2), e.g., below threshold $\Theta_{SpO_2} = 95\%$.

The system model for airway-laser MDPnP system is shown in Fig. 4. Simply, we can find there are five hazards in the system: (1) surgical fire h_1: $(state^{\text{O2Sen}} = [O_2 > 30\%]) \wedge (state^{\text{Laser}} = \textit{Emit})$, (2) potential surgical fire h_2: $(state^{\text{O2Sen}} = [O_2 > 30\%]) \wedge (state^{\text{Laser}} = \textit{NoEmit})$, (3) surgical fire h_3: $(state^{\text{Vent}} = \textit{Oxy\_Open}) \wedge (state^{\text{Laser}} = \textit{Emit})$, (4) potential surgical fire h_4: $(state^{\text{Vent}} = \textit{Oxy\_Open}) \wedge (state^{\text{Laser}} = \textit{NoEmit})$, (5) suffocation h_5: $(state^{\text{Oxim}} = [SpO_2 < 95\%]) \wedge (state^{\text{Vent}} = \textit{Oxy\_Closed})$.

Hazard h_2 and h_4 are called *potential* surgical fire, because the surgeon is free to initiate laser request. Even though in h_2 and h_4, surgical fire does not happen, surgeon's request will possibly come in any time, causing the laser scalpel to switch from *NoEmit* to *Emit* state. Then, h_2 and h_4 turn to be h_1 and h_3, respectively. Therefore, h_2 or h_4 are forbidden system states as well. To prevent those five hazards, the supervisor runs a set of control rules. Due to space limit, we only show three of them here. i) Rule r_1: $(state^{\text{Vent}} = \textit{Oxy\_Open}) \wedge (state^{\text{Oxim}} = [SpO_2 > 95\%]) \wedge (state^{\text{Laser}} = \textit{Req}) \Longrightarrow mode^{\text{Vent}} = \text{BLOCKED}$. ii) Rule r_2: $(state^{\text{Vent}} = \textit{Oxy\_Closed}) \wedge (state^{\text{Oxim}} = [SpO_2 > 95\%]) \wedge (state^{\text{O2Sen}} = [O_2 < 30\%]) \wedge (state^{\text{Laser}} = \textit{Req}) \Longrightarrow mode^{\text{Laser}} = \text{ALLOWED}$. iii) Rule r_3: $(state^{\text{Laser}} = \textit{NoReq}) \wedge (state^{\text{Oxim}} = [SpO_2 \leq 95\%]) \wedge (state^{\text{Vent}} = \textit{Oxy\_Closed}) \Longrightarrow mode^{\text{Vent}} = \text{NONBLOCKED}$.

IV. Plug-and-Play Service in VirLoop

Over the past decade, we have been pursuing PnP technologies for devices/services in the pervasive environment, resulting in a variety of PnP middleware technologies, such as UPnP, Jini, Salutation [1]. In spite of using distinct protocols and platforms, they are composed by similar features, e.g., description, discovery, control, and eventing. Now we are promoting PnP concept to the hospital environment. Likewise, we should build similar PnP features as well. However, due to the uniqueness of MDPnP systems, we should take into account the specific requirements and knowledge from clinical domain. In a MDPnP system, the supervisor is a central controller which coordinates the medical devices' behavior. It is usually more powerful than the medical devices running embedded software. This inspires us to make the supervisor a central point for device registration. The role that the supervisor plays is analogous to the control point in UPnP and the lookup service in Jini [1]. The messages used for PnP service are listed in Table I.

TABLE I: A list of PnP messages

Message Type	Usage
ADVERTISE	Advertise the existence of a device
DISCOVER	Discover the device that satisfies the requirements
SUBSCRIBE	Subscribe to receive the interested device information
NOTIFY	Notify the subscriber of the needed device information
GET	Obtain the value of a device attribute
SET	Update the value of a device attribute
ACTION	Guide a medical device to make mode transition
REPLY	The message is used to give positive/negative ACK

Device Description. Whether two devices are interoperable depends on the consistency of the data exchanged in the application layer as well as the lower communication protocols they use. In healthcare domain, a variety of standards have been developed to increase interoperability, including ISO/IEEE 11073 standard family [14][15] and HL7 [16], etc. So it is likely that different devices adopt different standards. Our approach to solve this heterogeneity problem is to create a standard *profile* for each medical device. By interpreting a device's profile, the supervisor will understand the messages from the device semantically and know how to interact with it. Specifically, we obtain a lot of hints from the ISO/IEEE 11073 Domain Information Model (DIM) [15], and categorize the device profile information into five groups. i) *Basic Attribute*: is_busy, device type, manufacture information, clock, device model, device health status, application layer standard, medical nomenclature, etc. ii) *Sensing*: measurement parameter, sampling period, unit, etc. iii) *Actuation*: mode transition and state transition function. iv) *Trigger*: event-trigger function, time-trigger function. v) *Communication*: network interface, communication protocol, protocol version, etc. As pointed out in [7], DIM is too complicated to implement. As a result, very few devices in the market implemented the full version of DIM. So in our design, we also make certain simplifications without hurting device interoperability under our device coordination model.

Device Registration and Discovery. While a medical device joins in the network, it automatically broadcasts an ADVERTISE message to announce its existence. The ADVERTISE message contains the profile of itself. On reception of a advertisement message, a supervisor will record the device in the registry. On the other hand, a supervisor is able to initiate a medical device discovery procedure by broadcasting a DISCOVER message. Detailed requirements related to the target medical device, e.g., device type, should be contained in

DISCOVER. If a medical device receiving DISCOVER finds itself matching the requirements, it immediately replies with a REPLY message along with its profile.

Eventing. Another service that the PnP component provides is subscription and notification. The supervisor can subscribe to receive two types of information: *sensor measurement* and *device state transition*. To subscribe the interested information, the supervisor will send a SUBSCRIBE message to the intended device. Acknowledgement by sending a REPLY message is mandatory to confirm the successful subscription. Afterwards, the intended devices will spontaneously send the subscribed information, encapsulated in NOTIFY message, to the subscriber. Furthermore, in the SUBSCRIBE message, the subscriber has to specify the notification frequency. To be specific, there are two types of notification frequency: *event-trigger* and *time-trigger*. The former defines a triggering condition for the notification procedure. Only in the case when the condition is satisfied, NOTIFY message is allowed to be sent. For example, laser scalpel sends NOTIFY to the supervisor every time when its state switches from *NoReq* to *Req*. The latter determines when the NOTIFY message should be sent by time, e.g., the subscriber can request a device to send NOTIFY message at a specific time point, while a timer fires, or in a periodic manner.

Device Manipulation. Device manipulation includes *device configuration* and *device control* services. Traditionally medical devices enable configuration by offering an on-board GUI/panel to the caregivers. By opening up the medical device interoperability, remote configuration and control of medical devices will be possible. The PnP module permits the supervisor to send GET and SET message to obtain and update the attributes of remote devices, respectively. Meanwhile, ACTION message can be used by the supervisor to remotely control the delivery devices, i.e., update them with new supervisory modes.

V. Device Composition and Synchronization Services in VirLoop

By exchanging the PnP messages, eventually the supervisor $\mathcal{S}$ will have a list of network-reachable medical devices, along with their features, capabilities and interaction mechanisms. If the device list contains all the devices $\mathcal{D}$ required for the clinical procedure, and meanwhile they are not involved in other MDPnP systems (i.e., it_busy=false), then the supervisor will initiate to compose those devices to build an integrated MDPnP system and run the synchronization service.

A. Device Composition Service

In our design, the supervisor initiates the composition process, because it relies on the state feedback from medical devices to adjust its control decision. Let us denote $\mathcal{F}$ as the set of devices that appear in the condition fields of the rule set $\mathcal{R}$. The state update from $\mathcal{F}$ may influence the supervisor's decision. So it is reasonable for the supervisor to establish feedback paths from all the devices in $\mathcal{F}$, so as to close the control loop. If the control loop is closed, we consider device composition is finished. Then the integrated system operate in an closed-loop manner. To establish a feedback path from a device, the supervisor should send a SUBSCRIBE message with the following subscription requirements.

- For a monitoring device $d_i \in \mathcal{F}$, the supervisor chooses *time-trigger* manner to subscribe d_i's sampling result $state^{d_i}$. The notification period equals to the sampling period, which is defined in d_i's profile.
- For a delivery device $d_j \in \mathcal{F}$, the supervisor subscribes d_j's state in *event-trigger* manner. That is, it expects to receive NOTIFY message, once d_j switches to a state that appears in the condition fields of rule set $\mathcal{R}$.

On reception of supervisor's SUBSCRIBE message, a medical device will send positive REPLY to the supervisor if the subscription is accepted. We say a feedback path from $d \in \mathcal{F}$ has been *established* if the supervisor's subscription is confirmed by d. The control loop in the MDPnP system is *closed* if all the feedback paths from $\mathcal{F}$ to the supervisor has been established. However, because of the unreliable network, both SUBSCRIBE and REPLY messages may get lost. The supervisor will retransmit the SUBSCRIBE messages until the subscription is confirmed. Note that the variability of the network condition may lead some feedback paths to be established earlier than others. Then, the supervisor will receive NOTIFY messages, containing device state information, from the successful paths. In this case, control rules may be triggered. In order to ensure safety, the device composition service has to enforce an additional safety rule: *if there exists any unsuccessful feedback paths, no control rules are allowed to be executed*. With this additional rule, we can easily conclude that all the medical devices must stay in their initial modes before the control loop is completely closed. Since it is hazard-free for multiple devices to stay in the initial modes, safety is thus guaranteed in the device composition stage.

B. Synchronization Service

In the open clinical environment, the communication is generally *unstable* and *asynchronous*, namely, message transmission may fail or suffer unbounded delay. As we know, in the asynchronous distributed systems, people often construct certain level of synchrony to reduce software complexity. Reduction of complexity leads to simpler logic and less latent bugs, and thus increases software reliability. This is especially important for safety-critical systems. A typical example is Time-Triggered Architecture (TTA) [17], which has been widely used by the safety-critical real-time distributed systems in the industry. TTA constructs a very precise global clock for the distributed system. Since MDPnP systems also fall into the category of time- and safety-critical systems, we follow this design philosophy and provide time synchronization service in the VirLoop middleware to ease application development. In the real-time distributed system area, there exist a great deal of work on time synchronization that can be adopted by VirLoop, such as [18], [19]. So we consider synchronization protocol design beyond our focus in this paper.

VI. Virtual Network Maintenance Service in VirLoop

Because of the asynchronous and unreliable communications, a mode update message may not be received by a delivery device. Missing critical modes endangers patient's life. To deal with this problem, our basic idea is to construct a Safety Assurance module in the supervisor (see Fig. 3), which can plan a sequence of safety-assured future modes for each delivery device, i.e., a *time-augmented mode transition path* (TMP) which will be defined later. Therefore, in the case of missing modes from the supervisor, the delivery device is still able to follow the sequence to take mode transition, and meanwhile maintains safety.

By introducing TMP, Fig. 3 shows the *real interaction* beneath the middleware layer. Specifically, the real interaction involves following three types of messages. i) *ACTION message from supervisor to delivery device* d_i, which contains a TMP $tmp^{d_i}_{t_s}$, generation time of the TMP (t_s), and the sequence number (SEQ); ii) *NOTIFY message from delivery device* d_i *to supervisor*, which contains the subscribed device state $state^{d_i}_{t_s}$, the occurrence time of the state transition (t_s), most recently received TMP ($tmp^{d_i}_{t_m}$), and the sequence number (SEQ); iii) *NOTIFY message from monitoring device* d_j *to supervisor*, which contains the up-to-date sampling result ($state^{d_j}_{t_s}$), the sampling time (t_s), and the sequence number (SEQ). Note that the sequence number here is used to deal with the message out-of-order problem during asynchronous communications. A message that arrives later than the messages with higher SEQ from the same source will be discarded. Suppose a message is received at time t, actually $t-t_s$ is the message transmission delay which may vary under different network conditions.

A. Time-augmented Mode/State Transition Path

Now we give the definition of TMP. For a delivery device d, a mode $mode^d$ actually defines a state space in which device d is allowed to operate. A *time-augmented mode* $\langle mode^d, t\rangle$ associates a time t with mode $mode^d$; it restricts device d to be in $mode^d$ starting from time t. A mode transition path $mp^d = mode^{d_0} \rightarrow mode^d_1 ... \rightarrow mode^d_l$ is a sequence of mode transitions, where each transition from $mode^d_i$ to $mode^d_{i+1}$ is valid. By associating each mode in the path with a time, we obtain a *time-augmented mode transition path* (TMP) $tmp^d_{t_0}$.

$$\begin{aligned} tmp^d_{t_0} \quad = \quad & \langle mode^d_{t_0}, t_0\rangle \rightarrow \langle mode^d_{t_1}, t_1\rangle \rightarrow \ldots \rightarrow \\ & \langle mode^d_{t_i}, t_i\rangle \rightarrow \ldots \rightarrow \langle mode^d_{t_l}, t_l\rangle, \end{aligned}$$

where $t_i < t_{i+1}$ for all $0 \leq i < l-1$, and l is the length of the path. A TMP specifies not only the order of mode transition for a device, but also when each transition should take place. We use $tmp^d_{t_0}[t]$ to indicate the specific mode in the path at time t, where $t \geq t_0$. When delivery device d receives $tmp^d_{t_0}$ (contained in ACTION message) from the supervisor at time t, it checks whether $t_0 \leq t < t_1$ hold or not. If yes, d will immediately switch to $mode^d_{t_0}$ and accept $tmp^d_{t_0}$ as its TMP. Otherwise, it implies that d misses certain mode transition deadline required by the supervisor, and thus the received $tmp^d_{t_0}$ is not accepted.

Now let us consider d is a monitoring device. Since d always operates in the SAMPLING mode, we are more interested in its state transition, i.e., the variation of its sampling results, which reflects how the monitored parameter changes. A *time-augmented state transition path* (TSP) for d from time t_0, denoted by $tsp^d_{t_0}$, is a sequence of sampling result and sampling time pairs. More formally,

$$tsp^d_{t_0} = \langle state^d_{t_0}, t_0\rangle \rightarrow \ldots \langle state^d_{t_i}, t_i\rangle \ldots \rightarrow \langle state^d_{t_l}, t_l\rangle,$$

where t_i is the sampling time of result $state^d_{t_i}$. Similarly, $tsp^d_{t_0}[t]$ represents the d's state in the path at time t.

The following introduces two basic operations on TMP/TSP. The first operation is called *truncation*, denoted by $trunc()$. A t-truncation of $tmp^d_{t_0}$ (or $tsp^d_{t_0}$), where $t \geq t_0$, is a sub-path of $tmp^d_{t_0}$ (or $tsp^d_{t_0}$) starting from t. Suppose $t_{i-1} \leq t < t_i$, we have

$$\begin{aligned} trunc(tmp^d_{t_0}, t) \quad = \quad & \langle mode^d_{t_{i-1}}, t\rangle \rightarrow \langle mode^d_{t_i}, t_i\rangle \rightarrow \ldots \\ & \rightarrow \langle mode^d_{t_l}, t_l\rangle. \end{aligned} \tag{3}$$

The second one is called *appending*, denoted by $append()$, which adds a time-augmented mode (or state) at the end of a TMP (or TSP). For example,

$$\begin{aligned} tmp^d_{t_0}.append(\langle mode^d_{t_{l+1}}, t_{l+1}\rangle) = \langle mode^d_{t_0}, t_0\rangle \rightarrow \ldots \\ \rightarrow \langle mode^d_{t_l}, t_l\rangle \rightarrow \langle mode^d_{t_{l+1}}, t_{l+1}\rangle. \end{aligned} \tag{4}$$

As a result, the length of the TMP/TSP is incremented by 1.

B. Safety Assurance Mechanism

Recall that we introduced TMP in the real interaction. A key question is: how does the supervisor generate TMP for delivery devices? In the following, we will answer this question.

Because of the unreliable network, the TMP sent to the delivery device may not be received and thus may not take effect. In this case, the supervisor does not know the exact TMP that is being used by the delivery device. However, it is able to estimate all the possible TMPs the delivery device is using, based on the past delivered TMP. Meanwhile, to reduce the estimation space, we require each delivery device to report their current TMP while they are sending NOTIFY message to the supervisor (see Fig. 3). For a delivery device $d_i \in \mathcal{D}$, the Safety Assurance module in the supervisor maintains a set of TMPs, denoted as $\mathcal{TP}^{d_i}(t)$, which includes all the possible TMPs used by d_i at time t according to the supervisor's estimation. $\mathcal{TP}^{d_i}(t)$ is initialized to $\{\langle mode^{d_i}_0, t_e\rangle\}$ while the closed-loop control setup is accomplished at time t_e. Here, $mode^{d_i}_0$ is d_i's initial mode. Afterwards, there are two conditions when the supervisor should update $\mathcal{TP}^{d_i}(t)$. First, when the supervisor generates a new TMP $n\_tmp^{d_i}_t$ for d_i (see Alg. 1), then

$$\mathcal{TP}^{d_i}(t) = \mathcal{TP}^{d_i}(t) \cup \{n\_tmp^{d_i}_t\}. \tag{5}$$

Second, once the supervisor receives a NOTIFY message from d_i, which includes d_i's most recently received TMP $tmp^{d_i}_{t_m}$,

Algorithm 1: Algorithm for calculating new TMP

Input: $\mathcal{TP}^d(t)$ or tsp_t^d for all $d \in \mathcal{D}$
Output: A new TMP $n\_tmp_t^{d_r}$ for device d_r

1 **for** $t_h = t$ **to** $infinity$ **do**
2 **foreach** d *is a delivery device in* $\mathcal{D}$ **do**
3 $State_{t_h}^d = \bigcup_{tmp_t^d \in \mathcal{TP}^d(t)} \left(Space^d(tmp_t^d[t_h])\right)$;
4 **foreach** d *is a monitoring device in* $\mathcal{D}$ **do**
5 $state_{t_h}^d = tsp_t^d[t_h]$;
6 **if** *there is a control rule* r *enabled at time* t_h *&& no hazards will be generated after executing* r **then**
7 **if** $execute\_once$ **then**
8 $n\_tmp_t^{d_r} \leftarrow NULL$;
9 **foreach** $d_i \in \mathcal{D} \backslash \{d_r\}$ **do**
10 Freeze $\mathcal{TP}^{d_i}(t)$;
11 $execute\_once \leftarrow false$;
12 $n\_tmp_t^{d_r} \leftarrow n\_tmp_t^{d_r}.append(\langle mode_r^{d_r}, t_h \rangle)$;

then $\mathcal{TP}^{d_i}(t)$ is updated to contain the t-truncation of all the TMPs generated no earlier than t_m, i.e.,

$$\mathcal{TP}^{d_i}(t) = \{trunc(n\_tmp_{t_k}^{d_i}, t) | t_m \leq t_k \leq t\}. \quad (6)$$

On the contrary, for a monitoring device $d_j \in \mathcal{D}$, the Safety Assurance module maintains a worst-case estimation of d_j's future state, i.e., TSP $tsp_t^{d_j}$. Updating $tsp_t^{d_j}$ happens once a NOTIFY message is received from d_j which contains d_j's up-to-date feedback $state_{t_s}^{d_j}$. At first, we use the worst-case prediction function to generate a TSP $tsp_{t_s}^{d_j}$ as the estimation of d_j's state starting from time t_s. Then, by removing the states from t_s to t, we can obtain $tsp_t^{d_j}$, i.e., $tsp_t^{d_j} = trunc(tsp_{t_s}^{d_j}, t)$. Note that the worst-case prediction function is dependent on the medical devices' states and the patient's characteristics. Actually, the assumption of having the worst-case state prediction is not unreasonable, because it is common practice that the surgeon should estimate patient's future state according to the past treatment and drugs delivered. However, our previous work found that long-term estimation of future state is generally impossible [8]. So, once beyond our estimation range, we assign the *worst valid sampling value* of d_j, denoted as $WORST^{d_j}$, into $tsp_{t_s}^{d_j}$. For example, for pulse oximeter, $WORST^{\text{Oxim}} = 0\%$ and our estimation range for SpO_2 is next 6 seconds. Then, a TSP starting from 10 second could be $tsp_{10}^{\text{Oxim}} = \langle 98\%, 10\rangle \rightarrow \langle 95\%, 12\rangle \rightarrow \langle 93\%, 14\rangle \rightarrow \langle 0\%, 16\rangle$.

As the system execution progresses, the Safety Assurance module will continuously update $\mathcal{TP}^{d_i}(t)$ and $tsp_t^{d_j}$ for each delivery device d_i and monitoring device d_j. Meanwhile, the module will check whether there is any control rule enabled. If the execution of a rule r will not yield hazards by considering all the combinations of r's resultant state space $Space(mode_r^{d_r})$ with other devices' possible states, the supervisor will execute the rule and continue to generate the future modes for d_r, i.e., obtain a new TMP $n\_tmp_t^{d_r}$. Alg. 1

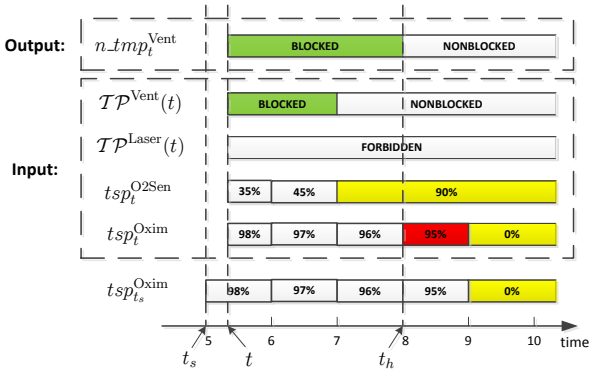


Fig. 5: An example of executing Alg. 1 in the airway-laser MDPnP system

shows the detailed steps for calculating $n\_tmp_t^{d_r}$. Line 2-4 describes how to determine a medical device's state space at future time t_h. Procedure from line 7-11 is executed only once, which freezes $\mathcal{TP}(t)$ for the devices except d_r, and hence only allows to execute control rules to generate d_r's future modes. Line 12 means that once another rule is executed, the resultant mode is appended into $n\_tmp_t^{d_r}$. Alg. 1 terminates until the length of $n\_tmp_t^{d_r}$ is fixed. After the algorithm finishes, the generated $n\_tmp_t^{d_r}$ will be immediately sent to d_i so as to update d_i's TMP. Note that there is a possibility that multiple control rules can be enabled if their condition fields are satisfied and no hazards yield after their executions. In this case, we randomly choose one rule to execute, and proceed to generate the complete TMP.

Fig. 5 illustrates how Alg. 1 works in the context of airway-laser surgery. Suppose pulse oximeter notifies the supervisor of its latest sampling result ($state_{t_s}^{\text{Oxim}} = 98\%$) at time $t_s = 5$ sec, and the supervisor receives this NOTIFY message at time $t = 5.3$ sec. Obviously, $t - t_s$ is the transmission delay of the NOTIFY message. While receiving NOTIFY message from the pulse oximeter, the first step for the supervisor is to calculate $tsp_{t_s}^{\text{Oxim}}$ by the worst-case prediction function. Then, tsp_t^{Oxim} is obtained by t-truncation over $tsp_{t_s}^{\text{Oxim}}$. Since at time t there is a control rule r_1 enabled (refer to Section III-D for the control rules), the supervisor will start running Alg. 1 to generate a new TMP. Because $d_{r_1} = \text{Vent}$, the supervisor freezes $\mathcal{TP}(t)$ for laser scalpel, and generates a new TMP for ventilator. Executing r_1 results in $n\_tmp_t^{\text{Vent}} = \langle \text{BLOCKED}, t\rangle$ temporarily. Then, the algorithm continues to generate the future modes. At time $t_h = 8$ sec, another rule r_3 is enabled. Executing rule r_3 forces the ventilator to switch to NONBLOCKED mode because of low patient's SpO_2 (i.e., $\leq 95\%$). Thus, we append $\langle \text{NONBLOCKED}, t_h\rangle$ to $n\_tmp_t^{\text{Vent}}$. Finally, we have $n\_tmp_t^{\text{Vent}} = \langle \text{BLOCKED}, t\rangle \rightarrow \langle \text{NONBLOCKED}, t_h\rangle$, which is the output of Alg. 1.

Now that the supervisor knows how to calculate TMP for delivery devices, in the Appendix we prove that safety is ensured when the TMPs are used by delivery devices.

C. Maintaining Virtually Perfect Network

At last, we introduce how the Safety Assurance module maintains a virtually reliable network for the application layer.

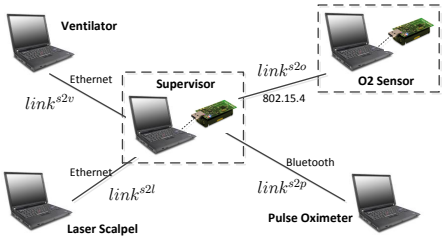


Fig. 6: The layout of the experiment

On the supervisor side, the Safety Assurance module filters the received NOTIFY messages from the medical devices, and only forwards the device state to the application layer. In addition, whenever the Safety Assurance module finds there is a mode transition for delivery device d_r, which corresponds to the execution of control rule r, it forwards the enabling condition field to the application layer. Thus, the application layer will execute the rule as well. On the delivery device side, the Safety Assurance module continuously retrieves the mode from the most recent TMP received from the supervisor, and forwards the mode to the application layer. Since executing r has been verified to be safe by the supervisor's Safety Assurance module, from the application layer's point of view, the whole system's safety is guaranteed as well.

VII. Evaluations

This section evaluates the safety property and performance of VirLoop middleware. Due to the fact that most of existing medical devices are not programable and do not support remote control, we use other equipment, such as laptops and wireless sensors, to simulate the real medical devices. The simulated medical devices are called *virtual medical devices*. We implement VirLoop on the virtual medical devices, and run a well-known MDPnP scenario, airway-laser surgery.

A. Experiment Settings

Three laptops are used to simulate a ventilator, a laser scalpel, and a pulse oximeter, respectively. Meanwhile, we use another two laptops, each associated with a TelosB sensor, to simulate a supervisor and a O_2 sensor. In the supervisor and O_2 sensor, the laptop is connected with the TelosB sensor via USB port. We implement all the modules of VirLoop in each simulated device. Additionally the supervisor runs the control rules for airway-laser surgery. Firstly, by exchanging PnP messages, these devices are able to recognize each other. Then, Synchronization module and Closed-loop Setup module enable the devices to be synchronized and integrated as a MDPnP system whose layout is shown in Fig. 6. The link names and communication types are depicted in the figure as well. The rationale under such setting is that we try to test VirLoop's PnP capability and safety property in the case of diverse communication interfaces and wireless environments.

B. Trace Analysis

In the experiment, we collect real-world traces to serve as the measurements for pulse oximeter and O_2 sensor. However, we confront one problem. That is, due to the limitation of existing devices and traces, we are not able to collect traces about airway O_2 concentration. Fortunately, CO_2 data can be found, and thus we use CO_2 data to replace O_2 measurements because of the negative correlation between O_2 and CO_2 measurements in patient's trachea [8], i.e., high (low) O_2 concentration implies low (high) CO_2 concentration. To be specific, the trace we use is called *HKPolyU Trace*, retrieved from an emulated airway-laser surgery [8]. Note that the time span of the HKPolyU Trace is 1200 seconds, i.e., the airway-laser surgery lasts 1200 seconds. Because of the negative correlation between O_2 and CO_2 measurements, in the sequel, we will replace all the parameters regarding O_2 by CO_2 values. For example, the O_2 threshold ($\Theta_{O_2} = 30\%$) in the control rules is replaced by CO_2 threshold ($\Theta_{CO_2} = 15$ mmHg).

Recall that the Safety Assurance module relies on the worst-case prediction to obtain the TSPs of the monitoring devices. A reasonable way of constructing the worst-case prediction functions for SpO_2 and CO_2 is to obtain the *worst-case variations in one sampling period*. Then we can reason out the whole TSP by considering how many sampling periods have passed. For pulse oximeter, the lower SpO_2 implies worse patient's physiological condition. So we are interested in its *maximum decrease in one sampling period*. Similarly, we should figure out the *maximum decrease in one sampling period* for constructing CO_2's TSP. These results are obtained through trace analysis (see Tab. II). Note that the worst-case prediction function depends on the states of medical devices.

TABLE II: Trace analysis results for the worst-case prediction

Parameter	Unit	Sampling Period	Maximum Decrease in One Sampling Period
SpO_2	%	2 sec	1 ($state^{\text{Vent}} = Oxy\_Open$)
			1 ($state^{\text{Vent}} = Oxy\_Closed$)
CO_2	mmHg	2 sec	29 ($state^{\text{Vent}} = Oxy\_Open$)
			10 ($state^{\text{Vent}} = Oxy\_Closed$)

C. Evaluation Results

MDPnP systems not only help to enhance patient safety, but also have to provide satisfiable quality of medical services. So in the following, we will evaluate VirLoop's safety property, as well as its service metrics, including *clinical efficiency* and *average response time*. In the context of airway-laser surgery, clinical efficiency means the percentage of time when the laser scalpel is in the ALLOWED mode, in which the surgeon can use the laser to cut patient's trachea. Average response time means the average delay from the time of surgeon's request to use laser to the time when the request is met. Meanwhile, we also evaluate the *communication cost* of VirLoop, i.e., the total number of messages that have been sent, regardless of whether message reception is successful or not. Note that VirLoop is compared with NASS [5] in the experiment.

In the experiment, we create two types of unliable communications. First, we adjust the *distance* and *line-of-sight condition* for link $link^{s2p}$ and $link^{s2o}$ to achieve different levels of wireless link quality. To be specific, we set up four placements for them: (1) **1M**: 1-meter distance with line-of-

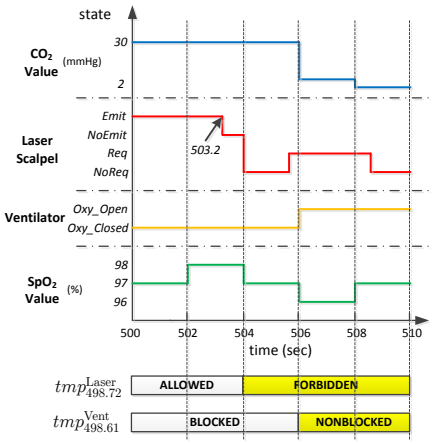


Fig. 7: System state diagram ($link^{s2p}$=1M, $link^{s2o}$=1M)

sight, (2) **3M**: 3-meter distance with line-of-sight, (3) **5M**: 5-meter distance with line-of-sight, (4) **5M_W**: 5-meter distance with a wall in the middle. The link quality becomes worse from placement 1 to 4. Second, we intentionally disconnect link $link^{s2l}$ or $link^{s2v}$, so that we can investigate the system's behavior in the case of network disconnection. Specifically, $link^{s2l}$ is disconnected from 500-510 second and 1000-1010 second; $link^{s2v}$ is disconnected from 800-810 second. Aside from the network setting, we also need to model the surgeon's behavior during the surgery. As real surgeon's behavior is very complex, in our experiment we choose a simple model, that is, surgeon decides whether to initiate a request in every D_i seconds, where D_i follows a uniform distributed in $[2, 5]$. The probability of initiating a request is 0.5 at the beginning of D_i, and the request is not released until D_i ends. For each following test, we will run 20 times to cancel out the randomness generated by surgeon's behavior.

1) Safety Property: Along the 1200-second surgery, we collected the states of all the virtual medical devices to check whether hazards happened or not. The results show that no hazards had happened in any circumstances, which validates the safety property of VirLoop. Due to space limit, we only show one system state diagram (Fig. 7) between 500-510 second (disconnection) in the case of **1M** placements for pulse oximeter and O_2 sensor. We can find from the figure that, no hazards happen (refer to Section III-B for the hazard list). Furthermore, even though laser scalpel is disconnected from the supervisor at 500 second, it is still able to allow surgeon to use laser, i.e., $state^{\text{Laser}} = \textit{Emit}$ from 502-503.2 second. This is because laser scalpel receives from the supervisor a TMP $tmp^{\text{Laser}}_{498.72}$ at 498.72 second, which allows the laser scalpel to continue operating in the ALLOWED mode till 504 second.

2) Clinical Efficiency: Fig. 8 shows the clinical efficiency diagram in different settings. As we can see, the clinical efficiency decreases as the link quality becomes worse, i.e., the placement changes from 1M to 5M_W. Because the supervisor conservatively uses the worst-case prediction to estimate future SpO_2 and CO_2 values, more severe message loss leads the supervisor to underestimate the real SpO_2 and CO_2 values more frequently, and give less time for the laser scalpel to operate in ALLOWED mode. However, under the moderate message loss condition, i.e., $link^{s2o}$=3M and $link^{s2p}$=3M, VirLoop still achieves comparable efficiency value (24.20%) in contrast with the best case scenario where $link^{s2o}$=1M and $link^{s2p}$=1M (29.20%). This result proves VirLoop's capability of tolerating message loss.

3) Average Response Time: Response time affects the surgeon's experience and clinical efficiency. Fig. 9 plots the results of average response time for VirLoop and NASS when $link^{s2p}$ and $link^{s2o}$ vary. In the figure, NASS 200 and NASS 400 refer to the NASS framework with cycle length equal to 200ms and 400ms, respectively. Besides, mode vector length is set to a large value 20 for NASS 200 and NASS 400 to eliminate the impact of vector length. We can see that VirLoop has much shorter response time compared to NASS. This difference is mainly due to the periodic interaction mechanism used by NASS. In NASS, supervisor's response to device state change has to be deferred to the next cycle, resulting in longer delay, no matter how long the cycle length is. However, no such delay is posed by VirLoop. In fact, the response delays in NASS 200 and 400 are 5-10 times of that of VirLoop, in our experimental cases.

4) Communication Cost: As NASS needs periodic message exchange between the supervisor and medical devices, we can easily determine the total message number for the cases of NASS 200 and 400. To be specific, in each cycle, there are 8 messages exchanged. So in total, NASS 200 geneates $1200 * \frac{1000ms}{200ms} * 8 = 48000$ messages within the 1200-second surgery. Similarly, we can figure out the total message number for NASS 400, that is, $1200 * \frac{1000ms}{400ms} * 8 = 24000$. The results of communication cost for VirLoop are depicted in Fig. 10. As network condition becomes worse the total message number decreases slightly, because worse network condition results in less chances of control rule execution. Hence, mode/state update messages are generated. In contrast with NASS, VirLoop maintains significantly lower communication cost (around a factor of 100 less).

VIII. Conclusion and Future Work

In this paper we proposed VirLoop middleware to support safe medical device coordination. Through VirLoop, devices are able to discover and recognize each other, and integrate together to enable device coordination. By using VirLoop services, unreliable and asynchronous communication problems will not compromise patient safety. Instead, VirLoop is able to hide the device heterogeneity and network issues, so that device coordination logic can be much simplified. Furthermore, VirLoop achieves high clinical efficiency and low delay to respond user's request, which enables its application for time-stringent clinical scenarios. Our future work includes investigating the impact of synchronization error on safety, and supporting run-time joining and leaving of medical device during device coordination.

References

[1] S. Helal, "Standards for service discovery and delivery," *IEEE Pervasive Computing*, vol. 1, no. 3, pp. 95–100, 2002.

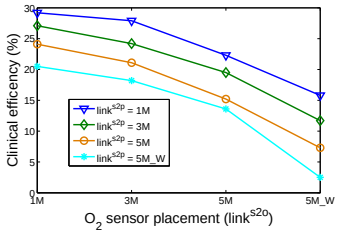


Fig. 8: Clinical efficiency

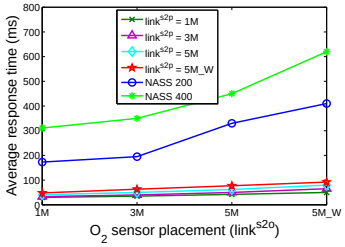


Fig. 9: Average response time

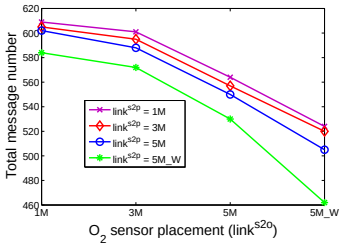


Fig. 10: Communication cost

[2] L. T. Kohn *et al.*, *To err is human: building a safer health system*. Washington, D.C.: National Academy Press, 2000.

[3] MD PnP Interoperability Program. [Online]. Available: http://www.mdpnp.org/

[4] *Medical Devices and Medical Systems — Essential Safety Requirements for Equipment Comprising the Patient-Centric Integrated Clinical Environment (ICE), Part 1: General Requirements and Conceptual Model*, ASTM International STAM F2761-2009, 2009.

[5] 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 *ICCPS'10*, 2010, pp. 149–158.

[6] C. Kim, M. Sun, H. Yun, and L. Sha, "A medical device safety supervision over wireless," *Autonomic Systems*, pp. 21–40, 2010.

[7] R. M. Hofmann, "Modeling medical devices for plug-and-play interoperability," Master's thesis, MIT, 2007.

[8] T. Li *et al.*, "From offline toward real-time: A hybrid systems model checking and cps co-design approach for medical device plug-and-play (mdpnp)," in *ICCPS'12*, 2012, pp. 13–22.

[9] A. King, D. Arney, I. Lee, O. Sokolsky, J. Hatcliff, and S. Procter, "Prototyping closed loop physiologic control with the medical device coordination framework," in *Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care (SEHC)*, 2010, pp. 1–11.

[10] D. Arney, M. Pajic, J. M. Goldman, I. Lee, R. Mangharam, and O. Sokolsky, "Toward patient safety in closed-loop medical device systems," in *ICCPS'10*, Stockholm, Sweden, Apr. 2010, pp. 139–148.

[11] I. Lee *et al.*, "Challenges and research directions in medical cyber-physical systems," *Proceedings of the IEEE*, vol. 100, pp. 75–90, 2012.

[12] V. Garg and B. Waldecker, "Detection of weak unstable predicates in distributed programs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, pp. 299–307, 1994.

[13] Y. Huang *et al.*, "Concurrent event detection for asynchronous consistency checking of pervasive context," in *PerCom'09*, 2009, pp. 1–9.

[14] *Health informatics — Point-of-care medical device communication — Part 20101: Application profile — Base standard*, ISO/IEEE 11073 Std.

[15] *Health informatics — Point-of-care medical device communication — Part 10201: Domain information model*, ISO/IEEE 11073 Std.

[16] Health level 7. [Online]. Available: http://www.hl7.org/

[17] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[18] H. Kopetz and W. Ochsenreiter, "Clock synchronization in distributed real-time systems," *IEEE Transactions on Computers*, vol. C-36, no. 8, pp. 933–940, 1987.

[19] U. Schmid, "Synchronized universal time coordinated for distributed real-time systems," *Control Engineering Practice*, vol. 3, no. 6, pp. 877–884, 1995.

APPENDIX

For a monitoring device $d \in \mathcal{D}$, suppose there are two states $state^d$ and $state^d_*$, and $state^d$ is no worse than $state^d_*$, from clinical point of view. For example, $state^{\text{Oxim}} = [SpO_2 = 96\%]$ is no worse than $state^{\text{Oxim}} = [SpO_2 = 94\%]$. Apparently, we have the following lemma.

Lemma 1. *If a control rule $r \in \mathcal{R}$ is safe to be executed when d is in $state^d$, then r is also safe to be executed in the case when d's state is $state^d_*$.*

Now we check the safety property of Alg. 1.

Theorem 1. *Alg. 1 makes sure that all the generated TMPs are safe to use by delivery devices at any time.*

Proof: Suppose current time is t_c, and a delivery device d receives an ACTION message from the supervisor, which contains TMP $tmp^d_{t_m}$. Apparently we have $t_m < t_c$. If $tmp^d_{t_m}$ is accepted by d, the implication is that none of the TMPs generated by the supervisor between t_m and t_c have been received by d. In the following, we will prove that it is safe for d to use $tmp^d_{t_m}$ at time t_c. Three cases are possible to happen during $[t_m, t_c]$ period.

Case 1: The most recent time that the supervisor runs Alg. 1 is t_m. On the one hand, for all delivery device $d_i \in \mathcal{D}\backslash\{d\}$, any TMPs accepted by d_i during $[t_m, t_c]$ period must be contained in $\mathcal{TP}^{d_i}(t_m)$. At time t_m, Alg. 1 utilizes $\mathcal{TP}^{d_i}(t_m)$ to calculate new TMP for d and safety is ensured while executing control rules in the calculation. Therefore, the resultant TMP for d is safe for d to use, no matter which TMP is finally used by d_i at time t_c. On the other hand, for all monitoring device $d_j \in \mathcal{D}\backslash\{d\}$, at time t_m the supervisor uses worst-case prediction to generate d_j's TSP. The real sampling results of d_j during $[t_m, t_c]$ period must be no worse than the values in the TSP. Lemma 1 implies that the generated $tmp^d_{t_m}$ must be safe to d. In summary, regardless of the real TMP used by d_i and the real sampling results of d_j, $tmp^d_{t_m}$ generated by Alg. 1 is safe for d to use.

Case 2: After t_m, the supervisor runs Alg. 1 to update d's TMP again. Since the new TMP has not been received by d and has not take effect, there is no need to check its safety.

Case 3: After t_m, there exists a delivery device $d_i \in \mathcal{D}\backslash\{d\}$ whose TMP is updated by the supervisor at time t_h. Here, $t_m < t_h \leq t$. Because of Equation 5 and 6, $\mathcal{TP}^d(t_h)$ always includes t_h-truncation of $tmp^d_{t_m}$, no matter whether the supervisor generates new TMPs for d or not during $(t_m, t]$ period. Let us denote by $n\_tmp^{d_i}_{t_h}$ the new TMP generated by Alg. 1 for d_i at t_h. While the supervisor calculates $n\_tmp^{d_i}_{t_h}$, it already considers $tmp^d_{t_m}$ to check the safety of control rule execution. Therefore, even though $n\_tmp^{d_i}_{t_h}$ will be accepted by d_i during $(t_h, t_c]$, safety is still ensured.

To summarize the above cases, we ensure the safety when delivery device d uses $tmp^d_{t_m}$ for its future operation. Because of the arbitration of d and t_c, we prove Theorem 1. ■