# BEEER: Distributed Record and Replay for Medical Devices in Hospital Operating Rooms

Avesta Hojjati
University of Illinois at Urbana-Champaign
hojjati2@illinois.edu

Yunhui Long
University of Illinois at Urbana-Champaign
ylong4@illinois.edu

Soteris Demetriou
Imperial College London
s.demetriou@imperial.ac.uk

Carl A. Gunter
University of Illinois at Urbana-Champaign
cgunter@illinois.edu

## ABSTRACT

Medical devices in hospital operating rooms are getting increasingly interconnected. This enables them to download instructions and report results with less risk of error compared to traditional manual techniques. However, many of these devices are safety critical. Thus, any risks from cyber-attacks can be extremely high. This paper describes BEEER, a distributed record and replay framework suitable for environments where more than one safety critical device is in simultaneous use. A prominent example is a hospital operating room where a number of networked devices work together. In such scenarios, a key step to forensically analyze an incident is understanding the causality of events produced by devices. BEEER orders events during recording by leveraging the fact that it takes significantly more time for a drug's effects to come to prominence on a patient compared to device-to-device communication on a local network. During replay, BEEER uses a newly developed token mechanism to coordinate execution of the events. We implemented and evaluated a prototype of BEEER. We found that synchronization for short medical operations can be achieved using Network Time Protocol (NTP). For longer operations we designed and developed a new event ordering protocol (*projection protocol*) based on vector timestamps. BEEER's replay mechanism is efficient and therefore suitable for forensics analyses in practice.

**ACM Reference Format:**
Avesta Hojjati, Yunhui Long, Soteris Demetriou, and Carl A. Gunter. 2019. BEEER: Distributed Record and Replay for Medical Devices in Hospital Operating Rooms. In *Hot Topics in the Science of Security Symposium (HotSoS), April 1–3, 2019, Nashville, TN, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3314058.3314061

## 1 INTRODUCTION

Digital medical devices have been used in healthcare for over a decade. Such devices transform the way clinical operations are being performed, rendering care both more efficient and more effective. Equipped with advanced sensors and precision electronics, they can collect physiological measurements of patients in real-time and act on the human body in response. For example, a blood pressure cuff can control the rate by which infusion pumps deliver pulses of the infused drug at precision levels in the order of milliliters or even nanoliters. This can be done much more accurately and reliably than a human could do it manually. However, if such a device takes or delivers a wrong measurement or administers the incorrect dosage, it can lead to the death of the patient. Another class of devices are Implantable Cardioverter-Defibrillators (ICDs), which manage the heart with electric pulses. In these cases too, an action or a failure to act can cause death.

Operating Rooms are equipped with many different medical devices. These devices commonly fall under the category of embedded systems, since they either do not employ a user interface or if they do, it is simplistic and tailored to their particular functionality. Furthermore, they tend to run on hardware and software specific to the task the device is commissioned to perform. Manufacturers of such devices tend to focus on the precision and real time guarantees the device can offer in such time-critical environments. Unfortunately, this comes at the expense of security. The main reason for that is because traditionally these devices are accessed only by medical practitioners. As a result, the threat model during the design phase of embedded medical devices does not typically include a remote adversary or a stronger adversary with physical or proximity access to the device. We argue that this premise is fallacious especially with the recent advancements on the Internet of Things (IoT). Increasingly, medical devices become interconnected and interdependent. A prime example of this is the inter-connectivity between medical devices and clinical systems [55].

In essence, such advances introduce new attack surfaces: the software of the devices becomes more complex to support interactions with other devices, and essentially these devices are running commercial operating systems; the device itself needs to implicitly trust the inputs of other devices in the distributed medical operation system. At the same time, there exists the threat of insiders: one with physical or proximity access to those devices could purposefully tamper with them. In fact, researchers have demonstrated that one could inject Radio Frequency (RF) pulses to the Analog to Digital Converter (ADC) component of an infusion pump and render it inoperable [27, 53]. Rushanan et. al. [47] found vulnerabilities with the firmware of popular infusion pumps that are actively being utilized by hospitals across the US, which allowed one to tamper with the dosage levels the pump was administering. Another report stated that 56,000 adverse event reports have been linked to issues with infusion pumps [54]. Halperin et. al. demonstrated vulnerabilities with ICDs [20]. In each of these cases, it is impossible for device vendors or medical personnel to sufficiently reason about the origin of the attack.

**Figure 1: Alaris IV Infusion Pump**

In this paper we develop technology to allow medical devices to be enhanced with record and replay capabilities to enable forensic analyses. Medical devices already log local events. Table 1 illustrates an excerpt of an event log file of the Alaris IV Infusion Pump (Figure 1) at Saint Francis (OSF)

*HotSoS, April 1–3, 2019, Nashville, TN, USA*
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-7147-6/19/04…$15.00
https://doi.org/10.1145/3314058.3314061

**Table 1: Event Log of the Alaris IV infusion pump at OSF Saint Francis Medical Center.**

| Log Date | Description | Details |
|---|---|---|
| 3/13/2015, 10:39:21 AM | EXTERNAL_CONTROL_EVENT | EventID = BINARY_LOG_REQUEST |
| 3/13/2015, 10:39:19 AM | EXTERNAL_CONTROL_EVENT | EventID = BINARY_LOG_REQUEST |
| 3/13/2015, 10:39:19 AM | EXTERNAL_CONTROL_EVENT | EventID = BINARY_LOG_REQUEST |
| 3/13/2015, 10:38:45 AM | ACTIVATE_PAGE | PageID = 2 |
| 3/13/2015, 10:38:45 AM | FORM_REQUEST | Form = NEW_PATIENT; FormRequest = CANCEL_FORM |
| 3/13/2015,10:38:45 AM | FORM_REQUEST | Form = MAINTENANCE_MODE; FormRequest = FORM_REQUEST |

Medical Center. It is evident that events are logged in a coarse granularity (e.g. on second granularity) and the focus is on outcomes (e.g. infusion pump administered drug). The pump also generates a "malfunction" log which lists the *timestamps* of each malfunction on that device. However, a malfunction note is not enough to determine the root cause of problems: a bad reading on another device could have caused a bad action on a patient leading to a cascading effect on all devices connected through the patient. This limits an analyst's capability of determining erroneous system behaviors. In this work we aim to equip such devices with record and replay capabilities that can capture every system call invoked during a medical procedure.

In fact, there are a variety of systems proposed for record and replay [9, 10, 37, 45]. However, we believe that the traditional isolated record and replay capabilities employed on individual devices are not enough since they can only support local queries. Consider for example the following scenario: a Patient Monitor device (*PM*) is connected to a patient in an Operating Room. At the same time, an Infusion Pump (*IP*), is also connected to the patient. A nurse is observing output values of the PM and inputs values to the IP. At a point, the PM reports a bad value which triggers the nurse to provide a wrong input to the IP. In turn the IP administers the wrong dosage to the patient causing they death. Assuming the procedure was captured via current methods (such as video), a forensic investigation would involve interviewing the nurse, watching a video of the operation, and analyzing individual log from each device. This is a cumbersome process that also depends on human factors. With distributed record and replay, an investigator could input the global time of death and trace backwards all the events on all the participating devices. Granted, they could still exist the need to interview the medical staff for more clarity. Nonetheless, distributed record and replay as well as a query system would simplify the investigation and provide more trustworthy results. A good analogy of our approach is the blackbox of an airplane that can be analyzed in the case of an unfortunate event [7]. We aim to design the blackbox of the Operating Room.

We pose that the answer to this problem is a distributed record and replay framework for medical devices. An important distinction in our system is that device interaction happens through the patient's body. In particular, it takes a certain amount of time for a medication to have any physiological effect on a patient, this is known as *Onset* time. Table 2 illustrates typical Onset times for different types of insulin [32]. Onset times are usually in the magnitude of seconds or even minutes because, on average, it takes 13 seconds for the drug to travel from the site of injection (usually the arm) to the brain, where they take effect [1, 52]. Our system leverages these observations to order events across an Operating Room, and guarantees that the order of the events is preserved in a granularity smaller than the minimum Onset time of all drugs.

The term Onset refers to the time that it takes for a medication to have physiological effect on a patient. Ultimately, we are introducing the CPS assumption. That it is, it takes less time for medical devices to communicate compared to a medication to have any type of physiological reaction on a patient in an Operating Room. Table 2 represents an example of Onset time for different types of insulin. In our scenario, we are considering the patient as a non-digital actor of the distributed system.

Our record and replay framework uses a token-based mechanism which leverages the ordering of events at the time of recording to coordinate the execution of events during replay. In this work, we design, implement, and evaluate such a framework. In our framework, each medical device is equipped with local record and replay capabilities, but also synchronizes with the entire distributed system. This allows a replay mechanism to deterministically replay all the devices, capturing their inter-dependencies and order of events. Such a framework would enable (1) doctors and hospital administrators to determine the cause of a non-expected event in the Operating Room at a high-level and (2) device vendors and forensic analysts to determine where a device, and in particular which device, was the culprit.

**Table 2: Onset Action for Insulin.**

| Insulin Comparison Chart | |
|---|---|
| **Insulin Type** | **When does it start working?** |
| Lispro | 15-20 minutes |
| Aspart | 10-20 minutes |
| Glulisine | 5-15 minutes |
| Novolin | 30-60 minutes |
| NPH | 30-120 minutes |
| Glargine | 60-120 minutes |
| Detemir | 60-120 minutes |
| NovoLIN | 30-60 minutes |
| NovoLog Mix | 10-20 minutes |

**Contributions.** Our work makes the following contributions:

- *New Study.* We make a first of its kind study of NTP utilization in safety critical systems in Healthcare. Our findings indicate that synchronization for short medical operations can be achieved using NTP, and we implemented a new event ordering protocol (i.e., *projection protocol*) suitable for longer medical operations which involve single-threaded systems.
- *Functional framework.* We designed and implemented a novel token mechanism for replaying distributed events in an orderly fashion. This mechanism is compatible with both NTP and the introduced projection protocol. We implemented a functional record and replay framework that integrates NTP synchronization, the projection protocol, and the token-based replay mechanism.
- *Implementation and evaluation of a distributed record and replay system.* To the best of our knowledge, we are the first group to design, implement, and evaluate a distributed record and replay system that can be leveraged by doctors, device vendors, and forensic analysts to synchronously replay devices that participated in a medical operation. Our system can be used for training doctors and readily detecting causal events in an Operating Room. Additionally, our system could have applications in other distributed Cyber-Physical environments.

## 2 BACKGROUND

**Cyber-Physical Systems.** It is often appealing to identify which device has caused an issue in a distributed environment with many devices communicating over different channels. Given a wide variety of systems and different architectures, there is no unified solution to identify causality in distributed Cyber-Physical Systems. For example outage in a power-grid is usually reported by consumers who are suffering from no access to the electricity. This essentially could be solved by placing sensors across the grid in every gate. Despite being helpful, this solution still cannot identify the cause of an issue with precise granularity; this is specially a concern when an output of one device could be used for an input of another device. To overcome this challenge, smart grid has evolved using communication and information techniques to provide better situational awareness [34]. For example, periodic modeling or formal verification of the architecture has proven successful to some extent. While this approach is a step towards providing better visibility within the smart grid-system, it still can not assist with identifying the causality of an event in other environments such as an operating room.

**Operating Room**. An advanced Operating Room revolves around the patient.The most important function of such room is to help the patient in different ways. Each Operating Room at minimum includes: a Fetal Monitor (checks on the patient to see if his/her heart is beating), a Cardiac Monitor, a Defibrillator, Oxygen tanks and Nasal Cannulas, an EKG machine (monitors the normal behavior of the patient's heart), an Infusion Pump (infuses fluids), and an Anaesthetic machine (supports the administration of anaesthesia).

**Inside an Operating Room**. An example of device correlation in an Operating Room is as following: after checking the patient in, the first device that needs to be plugged to the patient is the vital sign monitoring system (commonly known as Patient Monitor). This device is critical since it provides the basic yet important information about the patient. Typically, the Infusion Pump is the second important/critical device which needs to be functional. In an advanced Operating Room, the Patient Monitor and Infusion Pump are communicating through the local network, this correlation allows the Infusion Pump to infuse pre-specified drugs to the patient in the case where his/her vital signs are not normal, such action takes place via the readings of Patient Monitor.

For our attack scenario we are considering the following:

**Setup**. A patient with critical conditions has entered the Operating Room, due to certain conditions (e.g., the device has actually been compromised by an adversary via vulnerability in the firmware) the monitoring system is having an abnormal reading of the patient's heart rate. This results in sending a signal to the Infusion Pump to infuse more than normal amount of certain drug (e.g., Adenosine is used to normalize the heart rate). Since the reading was faulty the infused amount will cause the patient to enter the cardiopulmonary arrest (cardiac arrest). In this scenario, a distributed record and replay system can help us to identify the compromised device even if that specific device is not running anymore. Firstly, the ability to replay both of these devices in parallel allows us to identify the causing event and applying more forensics techniques to learn about the specific vulnerability which has caused these issues. Secondly, we can use the replay logs towards an educational purpose. Often in an Operating Room, medical staff do face abnormal situations. This is either due to medical device behavior or the special condition of the patient. Based on our previous research and collaborations, we identified a need for a high level replay of device interaction in an Operating Room in order to be able to train the future medical staff.

**Record and Replay mechanism**. In our experiment settings, we use an eidetic computing platform(referred to as Arnold) to record the states of an operating system. An eidetic computer system refers to a system that provides the ability to recall any past state that existed on the computer and provides the lineage of any byte in a current or past state. An eidetic computer system supports both backward queries and forward queries. A backward query finds out where a particular state comes from and can be used to identify the source of an anomaly event. A forward query traces the outputs and current states that are derived from a particular state and can be used to find out all the measurements influenced by an anomaly event [10].

## 3 SYSTEM OVERVIEW

Here we present a high-level design of our system named **BEEER** (distri**B**uted r**E**cord and r**E**play for the op**E**rating **R**oom). BEEER aims to record an operation in Operating Room and replay it for forensic analysis or educational purposes.

**Design Goals**. BEEER is designed for an Operating Room where safety critical devices operate. Such devices take one of two major roles: active and passive. An active device is responsible to administer drugs or take an action on the patient whereas a passive device is connected to the patient with the purpose of taking measurements. For example, an Infusion Pump is connected to the patient and its main goal is to administer precise dosages of medications to the patient. A Sphygmomanometer or blood pressure cuff applies pressure to the patient's arm that facilitates measurement of the patient's blood pressure. The Infusion Pump and the Sphygmomanometer are examples of active devices in an Operating Room. On the other hand, a heart rate monitor is placed usually on the patient's finger and periodically takes measurements of the patient's heart rate. The latter is an example of a passive device in an Operating Room. In addition, there is a vision of connected medical devices. That is, the industry and academia are now considering situations where such medical devices would be able to communicate with each other [6, 12].

The heterogeneity of such medical devices and the fact that they are manufactured by different vendors, generates the need for solutions that can help vendors identify malfunctioned or compromised devices. Furthermore, physicians are also interested in automated solutions that can help them analyze an operation for educational purposes. BEEER is a distributed system that aims to fulfill these needs. To achieve that, it needs to take precise recordings of system events on each of the medical devices in an Operating Room. More importantly, BEEER needs to be able to synchronize these local events on a global timescale. For example, it should be able to help device vendors and physicians to distinguish and identify events that caused another event. Consider for example a scenario where an active device (A) and a passive device (B) are connected to a patient. An anomalous or unexplained reading by B could be caused by a malfunction of B or more interestingly by an action of A. BEEER is able to help identify events on A that happened before the spurious reading on B. Such information could be critical in identifying the culprit.

**BEEER Architecture**. Figure 2 illustrates the high level design of BEEER. Medical devices in the Operating Room (D1, D2, D3 in the figure), run a BEEER recording session in parallel with their normal operation. The BEEER recording component captures local system events such as system calls and updates a log file. The BEEER local recording component also runs a synchronization algorithm, essential for ordering events in a global timeline. BEEER synchronization can be configured to run either NTP (see section 4) or a `projection protocol` (see section 5) to order the events across the BEEER distributed system. BEEER assumes the presence of a *Master* device which facilitates synchronization. The Master device is also responsible for `merging` the local logs at the end of an operation (see section 6.1) and for replaying the devices for analysis in an interactive manner. BEEER enforces the ordered execution of the distributed events using a novel token mechanism (see section 6.2.) The Master should also be able to answer queries at different semantic levels to be useful for doctors and vendors.

**Implementation**. BEEER's local recording component is an extension of an eidetic system, named Arnold [10]. We have extended Arnold to use NTP synchronization or the projection protocol to generate local timestamps for each event. We extended Arnold's replay mechanism to allow for replay of recordings of multiple devices on a single machine. BEEER's replay will merge the local device logs at the end of an operation. Merging logically orders the events across the whole system based on their local timestamps. Arnold is limited in some cases, for example, its replay mechanism is not interactive. To overcome this issue, we have attached an interactive tool to the replay mechanism [29]. This will essentially assist with displaying each step of the recording during replay.

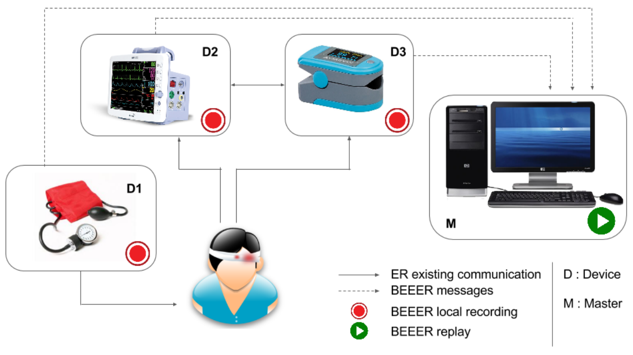


**Figure 2: BEEER high-level architecture**

## 4 DEVICE SYNCHRONIZATION

BEEER requires to be able to synchronize events across multiple medical devices. For example we need to know if a specific action of D1 on the patient happened before D2 took a measurement from the patient. This section will elaborate the device synchronization process on BEEER.

To this end BEEER supports two synchronization approaches: (1) using local NTP and (2) running a *projection protocol* which projects all events to a master timeline.

**Clock synchronization with NTP.** BEEER has the capacity of using NTP (Network Time Protocol). During our experiments (sec 7), we identified that both NTP and PTP (Precision Time Protocol) are not suitable for our environment. NTP is widely used in commercial operating systems, thus we decided to utilize this protocol for clock synchronization between different medical devices in an Operating Room. BEEER employs a master-slave synchronization since there is already a dedicated device for the replay operation. However, naive use of NTP that requires an external NTP root server would entail allowing devices in an Operating Room to access the Internet. This would greatly increase the attack surface. In such a scenario, one would need to include into the adversary model a remote adversary, since even a small bug or other vulnerabilities can be potentially exploited by a remote adversary to get access to the Operating Room's local network and further compromise other devices in the network. To address this, BEEER uses a local NTP where all medical devices synchronize with a local root machine. Our prototype uses the replay machine as the NTP root server, but, in theory, any trusted machine could assume that role.

BEEER wraps Arnold's record command at the clients, to enable NTP synchronization with the Master clock every 10 seconds, which allows it to minimize clock drift across machines. Note that NTP synchronizes a node's clock with the Master by calculating that node's clock offset with respect to that of the Master. In particular, if $o_{\text{real}}$ is the real offset of the client's clock, $o_{\text{ntp}}$ is the one calculated by NTP, $L_1$ is the latency of sending a message from the client to the Master, and $L_2$ is the latency of sending a message from the Master to the client, then it can be shown that the error in the NTP offset calculation has the following upper bound due to network delay:

$$|o_{\text{real}} - o_{\text{ntp}}| < |\frac{L_2 - L_1}{2}|.$$

To guarantee ordering, the following must hold in the NTP case. Let $e_A = |o_{\text{real},A} - o_{\text{ntp},A}|$ be the error in the offset calculation at machine A, $e_B$ the error at machine B, $t_A$ the global time an event is recorded at machine A and $t_B$ the global time an event is recorded at machine B, and $t_A < t_B$. Then NTP can guarantee ordering if:

$$e_A + e_B < t_B - t_A.$$

This ensures that when the clients are synchronized, the worst case combined error is not enough for the two events to be ordered differently. If the difference between the offsets $o_{\text{ntp}}$ on client nodes is large, then multiple recorded distributed system calls might be timestamped in the wrong order. Thus it is critical to synchronize as frequently as possible to limit the damage in the ordering caused by the clients' clock drifts. Furthermore, if the message delay is too large, then the error in the offset calculation at each client can be very high. This also results in unordered events. For reliable emergency room networks, BEEER offers the option of running a `projection protocol` instead of NTP for distributed event ordering. One problem at this stage is the overhead of NTP clock synchronization. NTP has a large overhead for clock synchronization, even when the synchronization is set to 10 second at minimum it might take 120 seconds to update the local clock, this is due to the architecture of NTP and its high dependency on the local network. NTP clients, by default synchronize within a 64s (minimum) interval and gradually increase to 1024s (maximum). However, this could entail a large enough clock drift that would make it hard to confidently order events at the system call granularity. On the other hand, our new event ordering protocol does not introduce any overhead since it doesn't require frequent synchronization during recording.

## 5 PROJECTION PROTOCOL

As we have mentioned, NTP can be used for short operations. In fact, it might be the best choice due to its simple deployment enabled by the availability of helpful libraries in most operating systems. For longer operations, the NTP protocol will require multiple synchronizations during an operation. However, it is enough for doctors, manufacturers, and forensic analysts to know the order that events happened in during an operation. This simply means that there is no need to use NTP for synchronization during recording [28]. For example, it would be sufficient to know that the sphygmomanometer increased the pressure on the patient's arm before the blood oxygen sensor started recording higher values. We take advantage of this observation and use vector timestamps, and developed a *projection protocol* (Algorithm 1) specifically for Operating Rooms to order events on BEEER.

In an Operating Room, causal ordering matters more than total ordering. While it is acceptable to mis-order some unrelated events, the recorded logs should preserve all the potential causalities between the events on different devices for forensic analyses. However, the traditional causal ordering in distributed systems is not enough under the scenario of an Operating Room. In an Operating Room, most devices act independently and as such any local recording would be considered concurrent with another device's local recordings. Although there is no explicit communication, devices can communicate implicitly: a device can perform an action on a patient which can stimulate a physiological reaction captured by a second device. That is, an event taken on one device can cause another event to be taken on a different device without explicit message communications. We name this type of causalities the *implicit causalities*. In the BEEER projection protocol, we would like to be able to reason about both the traditional causalities in distributed systems and the implicit causalities taking place uniquely in an Operating Room. To guarantee the correct ordering, we make the following assumptions about the system:

**Reliable and Ordered Message Delivery.** BEEER assumes that there is no message loss and all the messages in the same channel are delivered in the same order as they are sent. This is a common assumption for distributed systems, and it can be guaranteed using the TCP protocol.

**CPS.** BEEER makes a Cyber Physical System assumption. That is, it assumes the time it takes for a device action on a patient to instigate a physiological reaction on their body ($t_\phi$) and for that reaction to be sensed by another device ($t_s$), is strictly larger than the network latency ($L$) between any two devices in the system. That is, $t_\phi + t_s > L$. This isn't a strong assumption since it often takes several seconds or minutes for a physiological reaction to take place in a human body while the network latency between devices on the same local network is much smaller. Table 2 gives an overview of the time required for different types of insulin to have any type of physiological effect on a patient [32]. The term *Onset* refers to the time that it takes for a medication to have physiological effect on a patient, we take advantage of the long *Onset* time to validate the CPS assumption.

Based on the assumptions we formulate the requirement of our protocol as follows:

**Local Causality.** If $A$ and $B$ are two events on the same device, where $A$ happened chronologically earlier than $B$, then $A$ should be ordered before $B$ in the log.

**Message Causality.** If $A$ denotes the event of sending a message from device $D_1$ and $B$ the event of receiving that message on another device $D_2$, then $A$ should be ordered before $B$ in the log.

**CPS Causality.** If $A$ and $B$ are two concurrent *CPS events* on two different devices $D_1$ and $D_2$, and the time difference between event $A$ and event $B$ is greater than a threshold $t_{\text{cps}}$, then $A$ should be ordered before $B$ in the log.

The local causality and message causality requirements come from the definition of causal ordering in distributed systems. When $t_{\text{cps}}$ equals to the network latency $L$, the CPS causality requirements are enough to capture all the implicit causalities in a system satisfying the CPS assumption.

In the CPS causality requirement, a *CPS event* refers to an event that potentially makes or reacts to physiological or environmental changes.

For example, it could be an injection into the patient or a measure of the patient's heart rate. In the `Projection Protocol`, we categorize all events except for message sending and receiving as CPS events. This categorization can be further elaborated given background knowledge of the applications running on the devices, but categorizing more events as CPS events does not introduce error into the `Projection Protocol`.

To satisfy the local causality requirement and the message causality requirement, BEEER uses vector timestamps to order the events. However, ordering according to vector timestamps does not satisfy the CPS causality requirement because vector timestamps rely on communication across devices to identify causality. BEEER addresses this problem in the following way: all events - (which are in essence all system executions and messages) are projected on a single machine which acts as a Master node. Projections happen through *BEEER* messages which include the event's vector timestamp, the event id, and the device id. Upon the receipt of the message, the Master node appends a Master timestamp to the BEEER message and stores the messages in an append log. By doing this, we ensure that events can be meaningfully ordered when the operation is complete, by merging the individual devices record logs — an operation facilitated by the Master append log. During the merging, the events are first ordered with their vector timestamps. Events that are concurrent according to vector timestamps are ordered with their Master timestamps.

BEEER messages can be sent on every local action, send message, and receive message events. However, since CPS causality only exists between CPS events, we can optimize the protocol in practice by sending BEEER messages only after a CPS event. In this case, the BEEER message should include all previously unprojected local events and their vector timestamps. Upon the termination of a device, a final BEEER message is sent to report all the previously unprojected local events. The high-level logic is summarized in Algorithm 1.

**Algorithm 1:** BEEER Projection (Devices)

```
if a local event $e$ happens at $D_i$ then
    $D_i$ increases its $i$-th vector timestamp by 1.
    if $e$ is a CPS event then
        $D_i$ sends a BEEER message to the Master node.
    end
end
if $D_i$ receives a message from $D_j$ then
    $D_i$ updates its $j$-th vector timestamp according to the message.
    $D_i$ increases its $i$-th vector timestamp by 1.
end
```

The ordering of events given by the `Projection Protocol` satisfies the three causality requirements: local causality, message causality, and CPS causality. The local causality and message causality requirements are satisfied by the vector timestamp, therefore it is sufficient to prove the satisfaction of the CPS causality requirement.

Suppose $A$ and $B$ are two events taking place at different devices, and there is a CPS causality between $A$ and $B$. That is, event $B$ happens due to a physiological reaction caused by event $A$. Let $t_a$ and $t_b$ be the global time that $A$ and $B$ take place, and $\delta_a$ and $\delta_b$ be the network delay of the associated BEEER messages. $L$ is the maximum network latency between devices. Therefore, the master node would receive the BEEER messages from $A$ and $B$ at $t_a + \delta_a$ and $t_b + \delta_b$ separately. According to the CPS assumption, we have

$$\delta_a \leq L < t_{\text{cps}} < t_b - t_a$$

Moreover, because the network latency ($\delta_b$) is always greater than 0:

$$(t_b + \delta_b) - (t_a + \delta_a) = (t_b - t_a) - \delta_a + \delta_b > 0$$

That is, the BEEER message of event $B$ always arrives later than the BEEER message of event $A$. Therefore, the ordering of $A$ and $B$ according to the Master timestamp is always correct, which satisfies the CPS causality requirement.

To demonstrate the effectiveness of the `Projection Protocol`, we study a simple Operating Room setting as shown in Figure 3 with three devices: a Pulse Oximeter (PO), an Oxygen Sensor (OS), and a Blood Pressure Cuff (BPC). The vector timestamp associated with each local event indicates the local causality and message causality between each events, but they cannot reflect the CPS causality. For example, the Pulse Oximeter has no message exchanges with the other two devices, therefore the events taking place on it are considered as concurrent with all other events. However, the Pulse Oximeter can have implicit communications by measuring the physiological reactions on the patient, which are potentially caused by actions of the other two devices. For example, if the Blood Pressure Cuff (event J) applies pressure to the patient's arm via inflating, this would result to have no blood flow to occur through the patient's artery. Consequently, the Oxygen Sensor (event G) and the Pulse Oximeter (event D) will have different readings as a result of physiological changes on the patient's body. Although event J ([0, 0, 3]), G ([0, 3, 1]), and D ([4, 0, 0]) are concurrent according to their vector timestamps, the `Projection Protocol` should guarantee that G and D are ordered after J to correctly record the causalities. Since event G and event D's BEEER messages always arrive later than the BEEER message of J's, the Master timestamp ensures the correct ordering of these three events. However, note that although ordering provided by the `Projection Protocol` preserves all the causalities in the events, not all ordered events are causal. In the example, B and E are concurrent on different devices, but B is ordered after E by the `Projection Protocol`. Since not all events with time difference greater than $t_{\text{cps}}$ are causal, timestamps do not contain enough information to distinguish between CPS causal events and concurrent events occurred at different time. After the logs are merged between the devices, a hospital staff can infer whether there is a real causality between the ordered events based on additional information in the log.

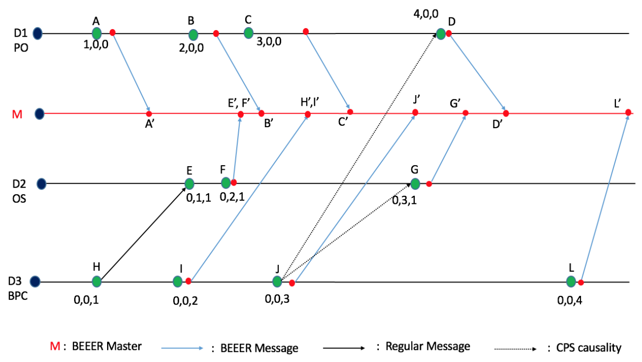


**Figure 3: BEEER Timestamp Projection Example. The green dots in the figure represent the local events happening at each device, including sending messages, receiving messages, and taking actions or measurements. The red dots represent the sending and receipt of BEEER messages. The black solid arrows indicate message causalities, the black dashed arrows indicate CPS causalities, and the blue arrows indicate transmission of BEEER messages.**

Determining the Onset time is challenging as it can vary between humans with different physiological characteristics. However, we can design experiments to measure the communication between different devices in an Operating Room. This can provide us with a conservative estimate of the network latency we can tolerate. In this experiment we do that for a heart rate sensing device. Determining $t_\phi$ is challenging as it can vary between humans with different physiological characteristics. However, we can design experiments to measure $t_s$ for different devices.

In particular, we built a heart rate sensing device that periodically reads the heart rate of a patient. The heart rate sensing device is composed of a commodity heart rate sensor that we connect to an Arduino board, our Arduino software application, and a machine running local report and replay. The sensor periodically sends the measured heart rate signals to the chip, and the chip transmits this data to a USB port of the connected computer.

We further wrote a program on a BEEER recording machine that takes this data and displays it on the screen.

Figure 4 shows the intervals (in milliseconds) between two consecutive system calls in the Heart Rate Sensing Program. The value for System Call Intervals for average case, standard deviation (std), and minimum value are; 57730, 433974, and 251 ms respectively. These values give us a conservative estimation of the network delay ($L$) we can tolerate with the *CPS* assumptions when the `projection protocol` is used for event ordering. Evidently, the time between two consecutive events ($t_s$) is long enough to transmit a message between two devices on a trusted local network and thus our CPS assumption holds in this setting.

With the `Projection Protocol` the resulting Master append log is equivalent with the log merging output from recording with NTP synchronization. Thus, it is compatible with BEEER's replay mechanism. Nevertheless, based on our experiments, we observe that knowing the casual order of events can eliminate the need for NTP protocol. For the purpose of reducing the synchronization overhead in longer operations we do not need to know the actual time an event has happened, this eliminates the need for NTP protocol.

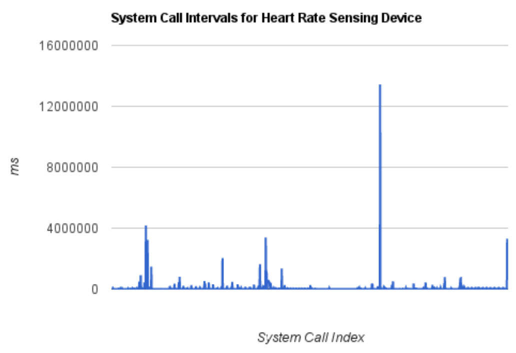


**Figure 4: System Call Intervals for Heart Rate Sensing Sensor**

## 6 REPLAY SCHEDULING

At the end of the medical operation, BEEER collects the local device logs and transmits them to the Master replay machine. There, it performs a log merging operation to create a global ordering of events. BEEER uses this ordering along with a token mechanism to enforce partial ordering of events during the replay operation.

### 6.1 Log Merging

When the logs are collected from the individual devices, BEEER will parse them to generate a `token queue`. The token queue will determine which event (system call) will be replayed in which order. In the case of the projection ordering, the Master will examine its event ordering and, the vector timestamps and event details at the individual device logs to determine a logical start and end time of events. In the case of NTP use, the local times are trusted to delineate the global time.

During log merging, there are four cases, and they are symmetrical in terms of event ordering as depicted in Figure 5. In Case D, events at different machines start and end at the same time. In case C, an event starts and finishes on one machine before any event on another machine starts. However, in Case B, when an event starts on machine 1 and before it finishes, an event on another machine starts and finishes. Also in Case A, when an event starts on machine 1 and before it finishes, an event on another machine starts. The second event finishes after the event on machine 1 finishes. BEEER treats case D as fully concurrent events and will allow machine 1 and machine 2 to execute those system calls in parallel. Note that the execution need not be concurrent as well. This is because there is no loss of causality during replay. In Case C, BEEER will force deterministic execution of events. For example, it will not allow machine

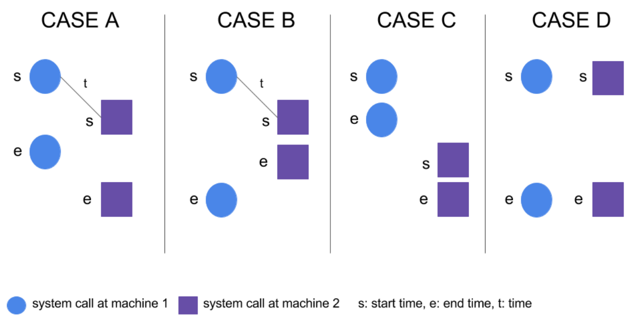


**Figure 5: Log Merging: the different cases BEEER considers**

2 to start replaying before the execution on machine 1 finishes. In cases A and B it is much harder to guarantee that the replay will follow the exact execution timing of the recording. Note that replaying a system call, does not necessarily take the same time as during recording. This is because Arnold caches the input for the system call which is fed during replay. In those cases BEEER guarantees that causal effects will be captured in the following way. BEEER uses a causality threshold ($t_c$) which determines at system call granularity whether a system call on machine 1 could potentially influence a system call on machine 2. In particular if the difference between the start time of the system call on machine 1 and the start time of the system call on machine 2 is less then $t_c$, BEEER reduces case A or B to case D. Otherwise, if the difference is equal or larger than $t_c$, this means that a causal effect could have taken place and as such BEEER needs to show that during replay. Thus, in the latter case, BEEER reduces A or B to C.

More formally, let $t_{s,1}$ be the recorded start time of an event on machine 1, $t_{s,2}$ be the recorded start time of an event on machine 2, $t_{s,2} > t_{s,1}$ since we are considering cases A and B, and $t_c$ is the causality threshold. BEEER reduces case A or B to case D if $t_{s,2} - t_{s,1} < t_c$ and to case C otherwise.

Setting $t_c$ is critical to ensure answering causality queries by doctors and forensic analysts. To capture causality, $t_c$ needs to be greater or equal to the least time it takes for an action by a device on a patient to cause a physiological reaction to her body which will be sensed by a second device. For example, there are guidelines for doctors and nurses that dictate such reaction times per drug, which can be utilized by BEEER to automatically determine $t_c$. However, $t_c$ should also be greater or equal to the minimum network latency observed in the system during recording. The latter is useful to guarantee reflection of causality during replay when medical devices have communicated directly with each other. If there is no such communication the former condition is sufficient.

More formally, let $t_{\mathrm{CPS}}$ be the minimum Onset time of all drugs. Let $t_c$ be the causality threshold and $L$ be the one-way network latency between two medical devices. Then $t_c$ must be: $t_c$

$$t_c \le \min(t_{\mathrm{CPS}}),$$

and

$$t_c \ge L.$$

The first condition is always necessary while the second is needed only when devices communicate with each other. However, in that case, since, according to the CPS assumption, $t_c \le L < \min(t_{\mathrm{CPS}})$, the second condition is enough for this case.

**Log Merging Algorithm.** Our log merging algorithm is illustrated in Algorithm 2. Initially it sets a pointer to the first log line of every log. Then it compares the pointed lines to figure out the order case (see Figure 5). In case D, the algorithm stores sequentially all fully concurrent events that have started and ended before the remaining pointed lines. Their pointers are increased to their next individual log lines. In case C, the oldest event is listed and its line pointer is increased. In the worst case the algorithm will encounter only case C. That means its time complexity would be linear with respect to the sum of all log lines:

$$O(\sum_{i=1}^{N} m_i),$$

where $m_i$ represents number of lines in the $i$th log, and $N$ represents number of logs. The space complexity is always $O(\sum_{i=1}^{N} m_i)$ since we need to store one line per event.

**Algorithm 2:** Log Merging

```
Input: $N$ recording logs
Output: A log with all distributed events ordered
$t_{temp} \leftarrow 0$;
GlobalEventList ← 0;
initLogPointers(pointers [N]);
while not at the end of all logs do
    switch compareLogLines (pointers [N]) do
        case A or B do
            if $t_{temp} < t_C$ then
                foreach line in pointers do
                    GlobalEventList ← line, D pointers [index of line]++
                end
            end
            else
                GlobalEventList ← oldestLogLine (pointers), C pointers [index of oldestLogLine ]++
            end
            break;
        end
        case C do
            GlobalEventList ← oldestLogLine (pointers), C pointers [index of oldestLogLine ]++
              break;
        end
        case D do
            foreach concurrent line in pointers do
                GlobalEventList ← line, D pointers [index of line]++
            end
            break;
        end
    end
end
return GlobalEventList
```

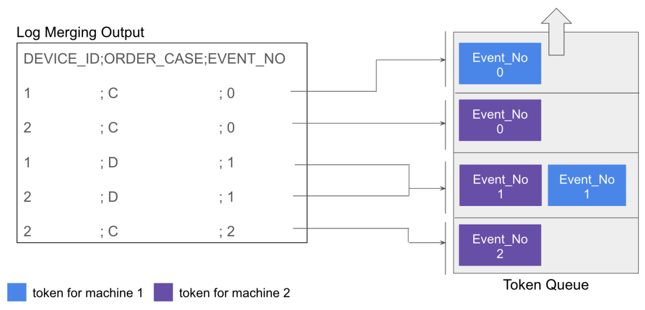
## 6.2 Scheduling with Token Distribution

During replay, a master machine enforces orderly execution of events using a token mechanism. Next, we describe BEEER's `token generation` algorithm and `token scheduling`.

**Token Generation.** The token generation procedure is illustrated in Figure 6. The log merging output is traversed linearly and events are added in a FCFS (First Come First Served) `Token Queue` (TQ). Whenever a case C entry is encountered, a new token is added to the queue. However, when a series of case D events are encountered, a queue node fills up with multiple `sub-tokens`. In our prototype implementation the sub-tokens are stored in a hashmap using the device id as the key. Note that a series of fully concurrent case D events in the log merging output could follow another series of fully concurrent events. BEEER distinguishes between them by assigning a monotonically increasing integer shared between fully concurrent events. Case C events always have unique such ids. The space complexity of the TQ is again $O(\sum_{i=1}^{N} m_i)$ since we store one token (or sub-token) per line in the global log.

**Token Scheduling.** During Replay, `executors` take on the task to replay the recording logs of each medical device. There is a 1:1 relationship between medical devices and executors. The Master node takes the role of the scheduler. In particular, every `executor` is only allowed to execute a system call if it has been granted the appropriate token. Figure 7 depicts the scheduling protocol.

An executor asks the Master scheduler for a token before it attempts to execute a system call. If the request matches the first token in the queue, the scheduler sends the token to the executor which proceeds with the replay of that system call. At the same time, the scheduler moves the token (or sub-token in case D) from the TQ to an `Active Token Queue` (ATQ). As long as the ATQ holds a token, no further tokens are being granted. This ensures replay scheduling of case C (see Figure 5). Note that waiting for the



```
DEVICE_ID;ORDER_CASE;EVENT_NO
1         ; C                ; 0
2         ; C                ; 0
1         ; D                ; 1
2         ; D                ; 1
2         ; C                ; 2
```

**Figure 6: BEEER token generation.**

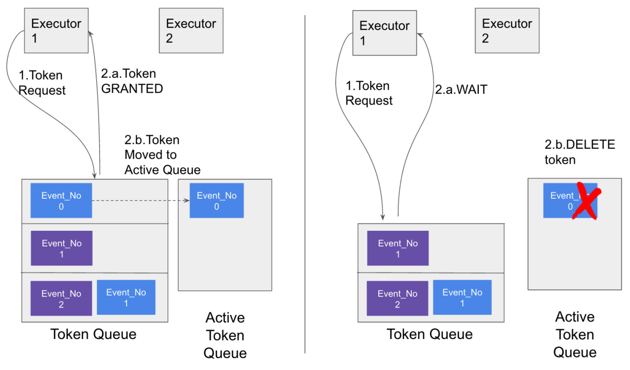


**Figure 7: BEEER Distributed Replay with Token Scheduling.**

end of the previous system call before executing a system call of another machine is important. Consider for example the case where we grant a token to machine 1 and before it finishes execution we grant a token to machine 2. Even though we granted the tokens in order, there is no guarantee that their execution will follow the same order.

To delete a token (or sub-token) from the ATQ, the scheduler needs to know the end time of a replay execution. A straightforward way to do this is to have the executor send a message to the scheduler. Note, however, that when a system call is executed, the next action that an executor takes is to replay the next system call. Thus it will need to send a message to the scheduler requesting a new token. BEEER executors only need to send token request messages since such a message implies the finished execution of the previous system call.

When the TQ has a node with sub-tokens first (case D), requests from multiple executors can be served concurrently. For example, if TQ's first node has a sub-token for Executor 1 and Executor 2, then the scheduler can grant a token to both in whatever order the request is received. This does not violate causality since sub-tokens represent events that are concurrent and as such their order of replay can be arbitrary. Again, the sub-tokens are moved to the ATQ once granted but they remain part of the same ATQ node. Any sub-token not part of the original node or token cannot be granted unless the ATQ node is fully emptied.

**Optimization: Token Batching.** BEEER's token scheduling can guarantee ordering for case C and case D. However, there is a large amount of messages that the scheduler needs to exchange with the executors. To make things worse, the executors need to wait after every system call execution for a new token. Consider for example the scenario where the TQ holds $n$ tokens for machine 1 consecutively, followed by $m$ tokens for machine 2 also consecutively. In this case, the scheduler needs to exchange $2n + 1$ messages with machine 1 and $2m + 1$ messages with machine 2. Assuming delay $d$ to deliver a message, machine spends $(2n + 1) \times d$ time negotiating tokens.

BEEER alleviates this using batching: BEEER traverses the TQ queue and merges consecutive tokens of case C destined for the same device. It further

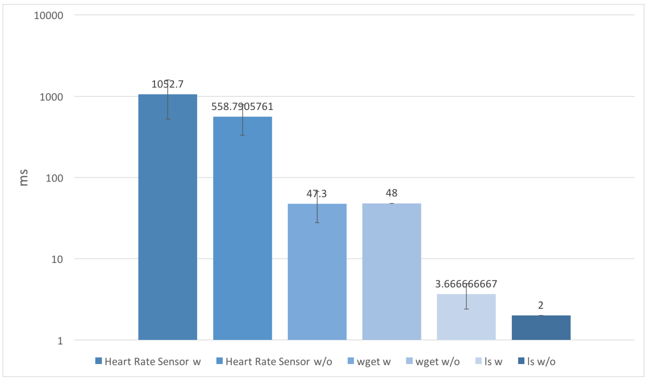


**Figure 8: Application Performance with Arnold recording (w) and without (w/o) recording.**

adds an expiration number equal to the number of tokens concatenated. The latter allows executors to continuously use the token until it expires. In the previous example, this would result in machine 1 exchanging just 3 messages [1] and thus spending $3 \times d$ time negotiating tokens. These are the "Token Request", "Token Granted", and the last "Token Request" indicating the end of replay on that machine.

## 7 EVALUATION

We have implemented BEEER on a number of machines connected on the same network. For our performance experiments, the BEEER clients and the BEEER Master are individual machines. The BEEER Master runs the log merging, token generation, and is responsible for granting tokens to clients. The replay executors are also located on the BEEER clients. The BEEER clients are configured to synchronize using local NTP, we have purposefully used NTP since it is commonly available in current medical devices. Note that with this scenario our replay performance results include the penalty of the network latency paid during the token protocol messages. In reality, the executors will be placed on the same machine as the scheduler. Thus the message latencies will be smaller in practice. Our evaluation is focused on the record and replay performance of the system.

### 7.1 Recording Performance

In this set of experiments we measure the execution time of different programs (heart rate sensor, `ls`, `wget`). For each of the benchmark programs, we measure the duration of each command/program under free settings: running without recording, recording without synchronization, recording using NTP for synchronization, and recording with projection protection. We repeat each experiment 20 times to measure the average and standard deviation of the execution time. For the experiment with the heart rate sensing device, the program measures the patient's heart rate several times within one execution. We estimate the time for a single measurement with $t_S = \frac{t_T}{m}$. $t_S$ is the time for a single measurement; $t_T$ is the total execution time of the program; $m$ is the number of measurements the program takes.

Figure 8 illustrates the average time it takes for the heart rate system to report a measurement when recording is enabled (w) and when it is not (w/o). We compare that with respective experiments on `ls` and `wget`.

It is evident that the heart rate system requires at least one order of magnitude more time to take a measurement compared to a local and a networking program to complete. We also observe that recording (using state of the art technology) is an expensive process. However, this does not have an adverse effect on medical devices that sense or act on the human body. Nevertheless, there might be some devices now or in the future that require more fine-grained recording. BEEER can utilize advancements in single machine recording technology to accommodate such scenarios. Note that we consider single machine recording to be out of scope for our work since we focus on coordinating such recording and replaying across multiple machines.

[1]Note that this is the best case for batching. In the worst case, batching has no effect.

Next, we measure the execution time of program `ls` on a system with no recording capabilities, on a system with single machine recording, on a system with BEEER recording using NTP for synchronization, and on a system with projection protocol recording. In the case of recording with NTP, the devices use NTP to synchronize once before the operation starts. Once NTP finishes synchronization then the devices start their operations which are recorded by BEEER. Figure 9 illustrates BEEER's recording overhead due to NTP. As expected, the performance of BEEER recording with NTP adds an additional initial cost to single machine recording which is added by the NTP synchronization protocol. NTP for clock synchronization might be sufficient for short operations, because it needs to happen once before the operation starts. In the case of longer operations due to higher clock drifts we would require multiple NTP synchronizations. However, as it is evident from the evaluation, the NTP synchronization is long enough (64 seconds on average) to cause missing important interactions. One solution to this issue is utilizing NTP asynchronously, which could be achieved on medical devices where multithreading is supported. Unfortunately neither the medical device in our experiment, nor many medical devices deployed in hospitals are able to support multithreading. To overcome this issue, we developed the projection protocol which is inspired by vector timestamps (see section 5).

This demonstrates that while NTP is an obvious option in synchronizing the devices, yet it is not practical for medical devices since it will be missing a large number of events. However, when the `Projection Protocol` is used instead of NTP, BEEER recording adds no additional overhead compared to single machine recording because sending the projection messages does not interfere with the recording process.

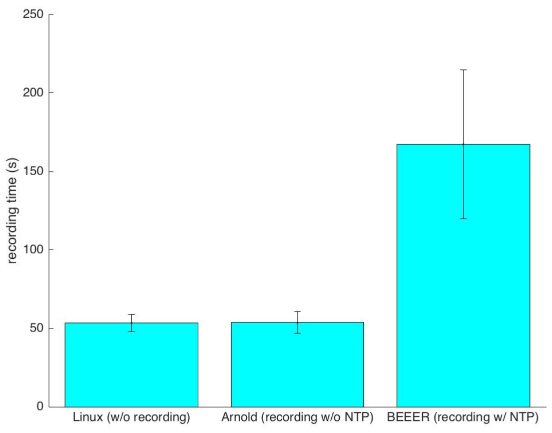


**Figure 9: Application Performance of BEEER recording compared to Arnold and w/o recording.**

### 7.2 Replay Performance

In this set of experiments we demonstrate the overhead imposed by BEEER's token scheduling for replay. In particular, we measure the time it takes to record and replay program `wget` fetching a file of 100MB on a single record and replay machine (Arnold), and compare it with the time it takes to record and replay the same program in BEEER, without batching enabled and with batching enabled. Figure 10 shows that BEEER adds a non-negligible overhead when run without batching since on every system call replay the executor needs to stop and acquire a new token. However, with batching enabled the executor needs to acquire a token once and can replay the program to completion.

To evaluate the effect of BEEER replay overhead on top of Arnold, we focus on the log merging operation. In particular we measure the number of tokens generated for the `ls` program run on two different machines. We first record `ls` with a small time difference between the machines to interleave their global times during log merging. We call this the *common* case. We repeat the above starting the recording of the second `ls` program after the first has been completed. We call this the *deterministic* case. This case is useful to illustrate the maximum benefit of batching for `ls`. The initial record log of `ls` contains 77 events and thus without batching BEEER will generate 77 tokens. Figure 11 illustrates the effect of batching. Intuitively, batching greatly reduces the number of tokens and thus the messages and stop-and-wait times for executors during replay will be reduced as well. This is simply due to collectively sending the tokens where the overhead will be significantly reduced when compared to sending each token individually.

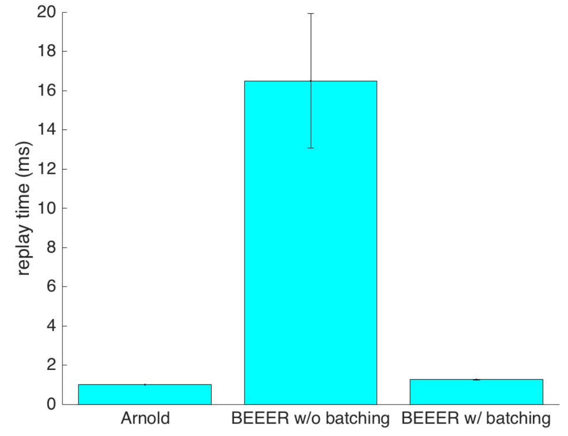


**Figure 10: Performance of BEEER for a single machine, single program w/ and w/o batching when compared to Arnold replay of the same program.**

## 8 RELATED WORK

**Record and Replay.** Recording and replaying program execution has been of interest for many years [5]. Deterministic record and replay has been widely studied on various platforms. Whole-system record and replay allows reproduction of all low level system states such as register and memory addresses, and often relies on processors [4, 9, 10, 11, 13, 22, 36] or virtual machines [14, 50]. Eidetic system [10] can deterministically replay a multiprocessor system. Arnold, divides the system into different groups of replaying processes, records the dependencies between these groups, and uses model-based compression to reduce the overheads. While Mozilla rr provides a comprehensive set of features, it does not support multiprocessor record-replay [42]. ReTrace [50] uses the deterministic replay technology of VMware hypervisor, and captures only non-deterministic events to reduce time and space overheads. Mobile record and replay such as VALERA [23] deals with the complicated environment of smartphones. Instead of focusing on system calls, VALERA records and replays sensor and network input, event schedules, and inter-app communications. Application-level record and replay solutions, record the states of a single application instead of the whole system [2, 18, 40]. For example, Bugnet [40] records the initial architectural state, register, and program counter updates, and all load values used during execution.

Distributed record and replay focuses on the record and replay of applications in a distributed environment. For example, Friday [18] targets the problem of debugging and profiling large-scale distributed applications. It considers distributed problems such as routing consistency in overlay networks and temporal state abnormalities caused by route flaps. Similarly, Jockey [48] and Bugnet [40] achieve the same goal. Most of these systems assume a synchronized clock or a virtual logical clock to achieve consistency in distributed report logs. However, maintaining such a clock is a big challenge in Operating Rooms.

**Synchronization for Safety Critical Systems.** In safety critical systems synchronization between disperse devices is an important task. For example in 2003 the northeast of U.S. suffered a blackout for approximately

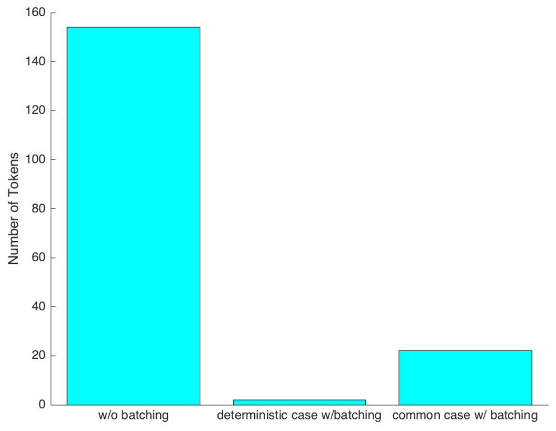


**Figure 11: Number of tokens generated for 2 parallelly recorded ls programs w/ and w/o batching.**

2 days. Despite the fact that the main reason of the blackout wasn't due to synchronization, yet majority of the recovery effort was focused on synchronizing different systems within the grid [33, 41]. This event caused a movement towards GPS clocks and away from NTP for synchronization in distributed safety critical systems. Additionally there is research based on logical synchronization for distributed systems where the focus has been based on modeling the environment and setting time expectations for each device. [43, 49].

**Clock Synchronization in WSN.** Wireless Sensor Networks (WSN) are large-scale networks of small, low-cost, and low-power sensors. Each sensor has a specific task such as observing the surrounding environment. Data collected by each sensor needs to be aggregated to a single, meaningful result. Therefore, synchronization between the sensors is highly desirable [51]. Due to limited energy and bandwidth, clock synchronization among sensors need to be light-weight and efficient. Some works use signals that can be received by all the sensors as a source of synchronization [19, 26, 44]. For example, wearable sensors can use distinctive gestures captured by camera to achieve synchronization [44]. Clock synchronization can be divided into master-slave synchronization and peer-to-peer synchronization [51]. In master-slave synchronization, a time server is available and all sensors synchronize with the time server [17, 21]. In peer-to-peer synchronization, sensor nodes broadcast reference messages with their neighbors [16, 30]. Our clock synchronization is similar to the master-slave synchronization problem in WSN. However, we want to avoid using periodic clock correction with the time server due to the large overhead of such protocol and limited resources on medical devices.

**Synchronization in Other Systems.** In tele-immersion systems, it is important to synchronize multimedia data collected from different sources. Such synchronization includes inter-stream synchronization, and inter-destination synchronization [15, 35]. A lot of work has been done on achieving multi-tier synchronization [24] and on evaluation of synchronization in terms of user experience [38] and quality of service [39].

Clock synchronization is also an important topic in other distributed systems such as power generation systems [8], and distributed real time systems [25]. Probabilistic clock synchronization [3] achieves a bound on the clock skew with a probability of invalidity associated with it. Although this approach is suitable for real time applications with soft deadlines, it is not suitable for an Operating Room environment. Another solution is to use partially-ordered clock vectors [31]. This requires a causality dependency between events happening on different devices. This dependency is not clear for some events on medical devices (e.g. measuring a pulse and measuring the blood pressure). Some other approaches combine continuous synchronization with instantaneous synchronization [25], or need special hardware [46].

## 9 CONCLUSION

We have presented BEEER, a distributed record and replay system for the Operating Rooms. To the best of our knowledge, BEEER is the first and only distributed record and replay system designed specifically for the Operating Rooms. During recording, BEEER supports both clock synchronization with local NTP and logical ordering of events achieved through a newly developed projection protocol. During replay, BEEER employs a novel token mechanism to schedule ordered replay of the recorded distributed events. BEEER further batches tokens to reduce bandwidth requirements and replay overhead. We have evaluated BEEER's token scheduling overhead and found it to be acceptable. The presented system can assist forensic analysts and device manufacturers that want to detect devices that adversely affected a medical operation. At the same time it is useful for medical practitioners that want to replay a medical operation for educational purposes.

Our prototype implementation assumes the BEEER executors are individual machines that communicate through the network with the BEEER scheduler. The presented system can assist forensic analysts and device manufacturers that want to detect devices that adversely affected a medical operation. At the same time it is useful for medical practitioners that want to replay a medical operation for educational purposes.

## ACKNOWLEDGEMENT

This work was supported in part by NSF CNS 13-30491 and NSF CNS 09-64392. The views expressed are those of the authors only.

## REFERENCES

[1] Philip L Altman and Dorothy S Dittmer. *Respiration and circulation.* Tech. rep. Federation of American Societies for Experimental Biology Bethesda MD, 1971.
[2] Silviu Andrica and George Candea. "WaRR: A tool for high-fidelity web application record and replay". In: *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on.* IEEE. 2011, pp. 403–410.
[3] K Arvind. "Probabilistic clock synchronization in distributed systems". In: *Parallel and Distributed Systems, IEEE Transactions on* 5.5 (1994), pp. 474–487.
[4] David F Bacon and Seth Copen Goldstein. *Hardware-assisted replay of multiprocessor programs.* Vol. 26. 12. ACM, 1991.
[5] Robert M Balzer. "EXDAMS: extendable debugging and monitoring system". In: *Proceedings of the May 14-16, 1969, spring joint computer conference.* ACM. 1969, pp. 567–580.
[6] Harald Bauer, Mark Patel, and Jan Veira. "The Internet of Things: Sizing up the opportunity". In: *Retrieved from: McKinsey at https://goo.gl/hkrbgE* (2014).
[7] George Bibel. *Beyond the black box: the forensics of airplane crashes.* JHU Press, 2008.
[8] Frede Blaabjerg et al. "Overview of control and grid synchronization for distributed power generation systems". In: *Industrial Electronics, IEEE Transactions on* 53.5 (2006), pp. 1398–1409.
[9] Nathan Dautenhahn et al. "Nested kernel: An operating system architecture for intra-kernel privilege separation". In: *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems.* ACM. 2015, pp. 191–206.
[10] David Devecsery et al. "Eidetic systems". In: *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14).* 2014, pp. 525–540.
[11] Joseph Devietti et al. "DMP: deterministic shared memory multiprocessing". In: *ACM SIGARCH Computer Architecture News.* Vol. 37. 1. ACM. 2009, pp. 85–96.
[12] Dimiter V Dimitrov. "Medical internet of things and big data in healthcare". In: *Healthcare informatics research* 22.3 (2016), pp. 156–163.
[13] Brendan Dolan-Gavitt et al. "Repeatable Reverse Engineering with PANDA". In: *Proceedings of the 5th Program Protection and Reverse Engineering Workshop.* ACM. 2015, p. 4.
[14] George W Dunlap et al. "ReVirt: Enabling intrusion analysis through virtual-machine logging and replay". In: *ACM SIGOPS Operating Systems Review* 36.SI (2002), pp. 211–224.
[15] John C Eidson et al. "Distributed real-time software for cyber–physical systems". In: *Proceedings of the IEEE* 100.1 (2012), pp. 45–59.
[16] Jeremy Elson, Lewis Girod, and Deborah Estrin. "Fine-grained network time synchronization using reference broadcasts". In: *ACM SIGOPS Operating Systems Review* 36.SI (2002), pp. 147–163.
[17] Bozena Erdmann and David Sanchez Sanchez. *Time synchronization in wireless ad hoc networks of medical devices and sensors.* US Patent App. 11/719,301. 2005.
[18] Dennis Geels et al. "Friday: Global Comprehension for Distributed Replay." In: *NSDI.* Vol. 7. 2007, pp. 285–298.
[19] Lewis Girod et al. "Locating tiny sensors in time and space: A case study". In: *Computer Design: VLSI in Computers and Processors, 2002. Proceedings. 2002 IEEE International Conference on.* IEEE. 2002, pp. 214–219.
[20] Daniel Halperin et al. "Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses". In: *Security and Privacy, 2008. SP 2008. IEEE Symposium on.* IEEE. 2008, pp. 129–142.
[21] Tian Hao et al. "Wizsync: Exploiting wi-fi infrastructure for clock synchronization in wireless sensor networks". In: *IEEE Transactions on mobile computing* 13.6 (2014), pp. 1379–1392.
[22] Derek R Hower and Mark D Hill. "Rerun: Exploiting episodes for lightweight memory race recording". In: *ACM SIGARCH computer architecture news.* Vol. 36. 3. IEEE Computer Society. 2008, pp. 265–276.
[23] Yongjian Hu, Tanzirul Azim, and Iulian Neamtiu. "Versatile yet lightweight record-and-replay for Android". In: *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications.* ACM. 2015, pp. 349–366.
[24] Zixia Huang et al. "SyncCast: synchronized dissemination in multi-site interactive 3D tele-immersion". In: *Proceedings of the second annual ACM conference on Multimedia systems.* ACM. 2011, pp. 69–80.
[25] Hermann Kopetz and Wilhelm Ochsenreiter. "Clock synchronization in distributed real-time systems". In: *Computers, IEEE Transactions on* 100.8 (1987), pp. 933–940.
[26] V Krishnamurthy, K Fowler, and E Sazonov. "The effect of time synchronization of wireless sensors on the modal analysis of structures". In: *Smart Materials and Structures* 17.5 (2008), p. 055018.
[27] Denis Foo Kune et al. "Ghost talk: Mitigating EMI signal injection attacks against analog sensors". In: *Security and Privacy (SP), 2013 IEEE Symposium on.* IEEE. 2013, pp. 145–159.
[28] Leslie Lamport. "Time, clocks, and the ordering of events in a distributed system". In: *Communications of the ACM* 21.7 (1978), pp. 558–565.
[29] Chi-Keung Luk et al. "Pin: building customized program analysis tools with dynamic instrumentation". In: *Acm sigplan notices.* Vol. 40. 6. ACM. 2005, pp. 190–200.
[30] Michael Kevin Maggs, Steven G O'keefe, and David Victor Thiel. "Consensus clock synchronization for wireless sensor networks". In: *IEEE Sensors Journal* 12.6 (2012), pp. 2269–2277.
[31] Friedemann Mattern. "Virtual time and global states of distributed systems". In: *Parallel and Distributed Algorithms* 1.23 (1989), pp. 215–226.
[32] *medlibes.com Types of Insulin.* Accessed: 2017-11-02. URL: https://goo.gl/ET75tw.
[33] JR Minkel. "The 2003 Northeast Blackout–Five Years Later". In: *Scientific American* 13 (2008).
[34] Yilin Mo and Bruno Sinopoli. "Secure control against replay attacks". In: *Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on.* IEEE. 2009, pp. 911–918.
[35] Mario Montagud et al. "Inter-destination multimedia synchronization: schemes, use cases and standardization". In: *Multimedia systems* 18.6 (2012), pp. 459–482.
[36] Pablo Montesinos, Luis Ceze, and Josep Torrellas. "Delorean: Recording and deterministically replaying shared-memory multiprocessor execution ef? ciently". In: *Computer Architecture, 2008. ISCA'08. 35th International Symposium on.* IEEE. 2008, pp. 289–300.
[37] Pablo Montesinos et al. "Capo: a software-hardware interface for practical deterministic multiprocessor replay". In: *ACM Sigplan Notices.* Vol. 44. 3. ACM. 2009, pp. 73–84.
[38] Niall Murray et al. "Subjective evaluation of olfactory and visual media synchronization". In: *Proceedings of the 4th ACM Multimedia Systems Conference.* ACM. 2013, pp. 162–171.
[39] Klara Nahrstedt et al. "QoS and resource management in distributed interactive multimedia environments". In: *Multimedia Tools and Applications* 51.1 (2011), pp. 99–132.
[40] Satish Narayanasamy, Gilles Pokam, and Brad Calder. "Bugnet: Recording application-level execution for deterministic replay debugging". In: *IEEE Micro* 1 (2006), pp. 100–109.
[41] Damir Novosel et al. "Dawn of the grid synchronization". In: *IEEE Power and Energy Magazine* 6.1 (2008), pp. 49–60.
[42] Robert O'Callahan et al. "Engineering Record And Replay For Deployability: Extended Technical Report". In: *arXiv preprint arXiv:1705.05937* (2017).
[43] Jeman Park and Taeho Kim. "A method of logically time synchronization for safety-critical distributed system". In: *Advanced Communication Technology (ICACT), 2016 18th International Conference on.* IEEE. 2016, pp. 356–359.
[44] Thomas Plötz et al. "Automatic synchronization of wearable sensors and video-cameras for ground truth annotation–a practical approach". In: *Wearable Computers (ISWC), 2012 16th International Symposium on.* IEEE. 2012, pp. 100–103.
[45] Gilles Pokam et al. "QuickRec: prototyping an intel architecture extension for record and replay of multithreaded programs". In: *ACM SIGARCH Computer Architecture News* 41.3 (2013), pp. 643–654.
[46] Parameswaran Ramanathan, Dilip D Kandlur, and Kang G Shin. "Hardware-assisted software clock synchronization for homogeneous distributed systems". In: *Computers, IEEE Transactions on* 39.4 (1990), pp. 514–524.
[47] Michael Rushanan et al. "SoK: Security and Privacy in Implantable Medical Devices and Body Area Networks". In: *Proceedings of the 2014 IEEE Symposium on Security and Privacy.* SP '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 524–539. ISBN: 978-1-4799-4686-0. DOI: 10.1109/SP.2014.40. URL: http://dx.doi.org/10.1109/SP.2014.40.
[48] Yasushi Saito. "Jockey: a user-space library for record-replay debugging". In: *Proceedings of the sixth international symposium on Automated analysis-driven debugging.* ACM. 2005, pp. 69–76.
[49] Lui Sha et al. *PALS: Physically asynchronous logically synchronous systems.* Tech. rep. 2009.
[50] MXVMJ Sheldon and Ganesh Venkitachalam Boris Weissman. "Retrace: Collecting execution trace with virtual machine deterministic replay". In: *Proceedings of the Third Annual Workshop on Modeling, Benchmarking and Simulation (MoBS 2007).* 2007.
[51] Bharath Sundararaman, Ugo Buy, and Ajay D Kshemkalyani. "Clock synchronization for wireless sensor networks: a survey". In: *Ad hoc networks* 3.3 (2005), pp. 281–323.
[52] Leonard Tarr, BERNARD S Oppenheimer, and Robert V Sager. "The circulation time in various clinical conditions determined by the use of sodium dehydrocholate". In: *American Heart Journal* 8.6 (1933), pp. 766–786.
[53] Timothy Trippel et al. "WALNUT: Waging doubt on the integrity of mems accelerometers with acoustic injection attacks". In: *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on.* IEEE. 2017, pp. 3–18.
[54] *Wikipedia.com Infusion Pump.* Accessed: 2017-06-20. URL: https://goo.gl/DgT1GK.
[55] Patricia AH Williams and Andrew J Woodward. "Cybersecurity vulnerabilities in medical devices: a complex environment and multifaceted problem". In: *Medical devices (Auckland, NZ)* 8 (2015), p. 305.