Hindawi
Wireless Communications and Mobile Computing
Volume 2018, Article ID 8589343, 13 pages
https://doi.org/10.1155/2018/8589343

*Research Article*

# TTEthernet Transmission in Software-Defined Distributed Robot Intelligent Control System

**Caibing Liu,[1] Fang Li ,[2] Guohao Chen,[1] and Xin Huang[3]**

[1]*South China University of Technology, Guangzhou 510641, China*
[2]*School of Computer Science & Engineering, South China University of Technology, Guangzhou 510641, China*
[3]*Guangzhou Start to Sail Industrial Robot Co., Ltd., Guangzhou 510641, China*

Correspondence should be addressed to Fang Li; cslifang@scut.edu.cn

Received 11 April 2018; Accepted 12 June 2018; Published 17 July 2018

Academic Editor: Dongyao Jia



With the integration of new technologies such as smart technologies and cloud computing in the industrial Internet of Things, the complexity of industrial IoT applications is increasing. Real-time performance and determinism are becoming serious challenges for system implementation in these Internet of Things systems, especially in critical security areas. This paper provides a framework for a software-defined bus-based intelligent robot system and designs scheduling algorithms to make TTEthernet play the role of scheduling in the framework. Through the framework, the non-real-time and uncertainties problem of distributed robotic systems can be solved. Moreover, a fragment strategy was proposed to solve the problem of large delay caused by Rate-Constrained traffic. Experimental results indicate that the improved scheme based on fragmentation strategy proposed in this paper can improve the real-time performance of RC traffic to a certain extent. Besides, this paper made a performance test and comparison experiments of the improved scheme in the simulation software to verify the feasibility of the improved scheme. The result showed that the delay of Rate-Constrained traffic was reduced and the utilization rate of network was improved.

## 1. Introduction

Industrial Internet of Things (IoT) integrates heterogeneous networks such as the Internet, wireless sensor networks [1], and fieldbus networks in traditional industrial networks. With the integration of new technologies such as smart technologies and cloud computing in the industrial Internet of Things, the complexity of industrial IoT applications is increasing. The performance requirements of the system are also constantly increasing [2]. Especially in critical security areas, real-time performance and determinism are very important [3–5]. In terms of "real-time", real-time performance means that the control instruction can reach the control execution node at a specified time delay. In terms of "determination", the determinism refers to the change degree of the delay of the data packet transmitted by the control system, which is the jitter of the data packet [6].

Take a distributed intelligent robot system as an example. The design becomes complicated because the system needs distributed deployment and network integration. Besides, functional requirements are constantly increasing and the sensors and actuators are always from different manufacturers [7]. Also, open architecture is required in achieving the modern robot system for modularity, flexibility, reusability, and reconfigurability. Moreover, because the application of component-based development methods and software bus is implemented in the open architecture, real-time performance and determinism are becoming serious challenges for system implementation in these Internet of Things systems [8–11].

Many researchers have conducted many meaningful studies on real-time performance in IoT. Oliver et al. [12] investigated the computational complexity of incremental scheduling problems and profile performance metrics. Jielin et al. [13] proposed a multiobjective nondominated sorting genetic algorithm based on the mapping matrix. Fuchs [14] et al. used a mixed physical layer and the new real-time Ethernet standard without deteriorating timing in real-time. Danielis et al. [15] summarized the different industrial Ethernet protocols for real-time data transmission. Wollschlaeger et al. [16] investigated the working conditions

of time-sensitive Ethernet. Sestito et al. [17] optimized data extraction and classification of traffic-related features for real-time Ethernet anomaly detection. Carvajal et al. [18] explored the trade-offs of three different solutions for real-time communication. Fuchs et al. [19] proposed a test and online monitoring method for access delay, path delay, and synchronization quality.

Software-defined network (SDN) is an innovative architecture of Emulex network [20]. It is an implementation of network virtualization. Its core technology, OpenFlow, achieves separation of control and data flows. SDN enables flexible control of network traffic, making the network more intelligent as a conduit. SDN is an influential technology for the IoT [21]. Shu et al. [22, 23] described countermeasure techniques that can be used to prevent, mitigate, or recover from some of these attacks.

Increasing number of nodes increases the time delay and jitter of the transmitted data. The nodes of the system include bus nodes, robot nodes, and various device nodes. Specific nodes include depth cameras, gesture recognition devices [24], and recording devices. Especially when it comes to multirobot control, it does not meet the requirement of real-time performance of system from the perspective of task scheduling. Therefore, looking for a suitable real-time network to ensure the "hard" real-time performance of data transmission is particularly important [25–27].

The communication control buses used in traditional robot control systems, such as RS485 and CAN buses, have disadvantages like low transmission rates, short transmission distances, and poor real-time performance [28]. These disadvantages are performance bottlenecks in distributed systems. Real-time Ethernet overcomes the problems of low speed and short transmission distance of the traditional control bus. Moreover, it has certain real-time performance and reliability. It has quickly become one of the important control buses in the safety-critical field [29, 30].

TTEthernet, a new real-time Ethernet technology led by TTech company, has become more and more popular in recent years [31]. TTEthernet is a standard protocol that is compatible with legacy Ethernet and defines three types of data flow: time-triggered (TT), Rate-Constrained (RC), and Best-Effort (BE) Ethernet [32]. Through a global clock synchronization algorithm, a certain accuracy of the synchronization clock is maintained. Therefore, TT traffic achieves real-time performance and certainty to some extent. TTEthernet is based on the exchange of Ethernet technology, which can be extended to any topology and has high scalability and flexibility [33, 34]. Moreover, TTEthernet offers redundant channels and fault tolerance mechanisms that make it able to serve key security areas [35].

Tămaş-Selicean et al. [36] optimized the packaging of messages, the assignment of frames to virtual links, the routing of virtual links, and the time-triggered static schedule in frames so that all frames are schedulable and end-to-end delay of the RC messages is minimized. Aristova et al. [37] discussed Ethernet evolution history and packet assembly in the OSI model, analyzed the limitations of blocking Ethernet applications in industrial automation, and described innovative solutions such as power over Ethernet and embedded Ethernet switches. Selvatici et al. [38] suggested that there is a long-term transition for network. They provided an essential element for the coexistence of network, allowing for transparent communication from the network to another network. Yang et al. [39] proposed a universal control system security model for industrial real-time Ethernet interference. The added security module can effectively prevent hackers from malicious attacks and ensure the security of real-time Ethernet. Tamas-Selicean et al. [40] proposed an optimization algorithm to offline static schedule of TT messages. The result is that the deadline for TT and RC messages becomes more reasonable, and the end-to-end delay of RC traffic is minimized.

Adopting a service-oriented software-defined architecture offers the benefits of loose coupling and flexibility for the system, but at the same time the whole system is not in real-time [41]. To reduce this cost, this paper proposes time-triggered Ethernet as the system's transmission protocol to make the system real-time and deterministic. Time-triggered Ethernet overcomes the problem of low-rate, short transmission distance, and other issues of traditional control bus. Three kinds of traffic of Ethernet meet different real-time requirements of the applications. TT traffic application deployment can be used for high real-time demanding control tasks, BE traffic can be used for general non-real-time applications, and RC traffic is suitable for applications that have certain QoS requirements but do not demand real-time performance.

This paper provides an architecture for a bus-based software-defined intelligent robot system and designs scheduling algorithms to make TTEthernet play the role of scheduling in the architecture. Through the framework, the non-real-time and uncertainties problem of distributed robotic systems can be solved. Moreover, a fragment strategy was proposed to solve the problem that rate limits traffic and thus there was a large delay.

The remaining sections are as follows: Section 2 is TTEthernet-based distributed robot control system framework. Section 3 details the construction of distributed robot control system model based on TTEthernet. Section 4 introduces scheduling technology of the system. Section 5 details the experiments. Section 6 gives a conclusion of this paper.

## 2. Framework

Figure 1 shows the hierarchy architecture of the software-defined TTEthernet-based distributed robot control system, which intuitively presents the application of TTEthernet in the entire system framework. There are three layers in the framework, including service layer, bus layer, and bus management layer. Service layer is an integration of application modules. As a software-defined architecture, data and control are separated in the system. Collaborative control service coordinates the services of various sensors and provides data to the robot. There are three kinds of data services in this layer, including data acquisition service, data transmission service, and data processing service. The data service can collect and calculate information needed by the system, which can save computing resources and the hardware cost can be reduced

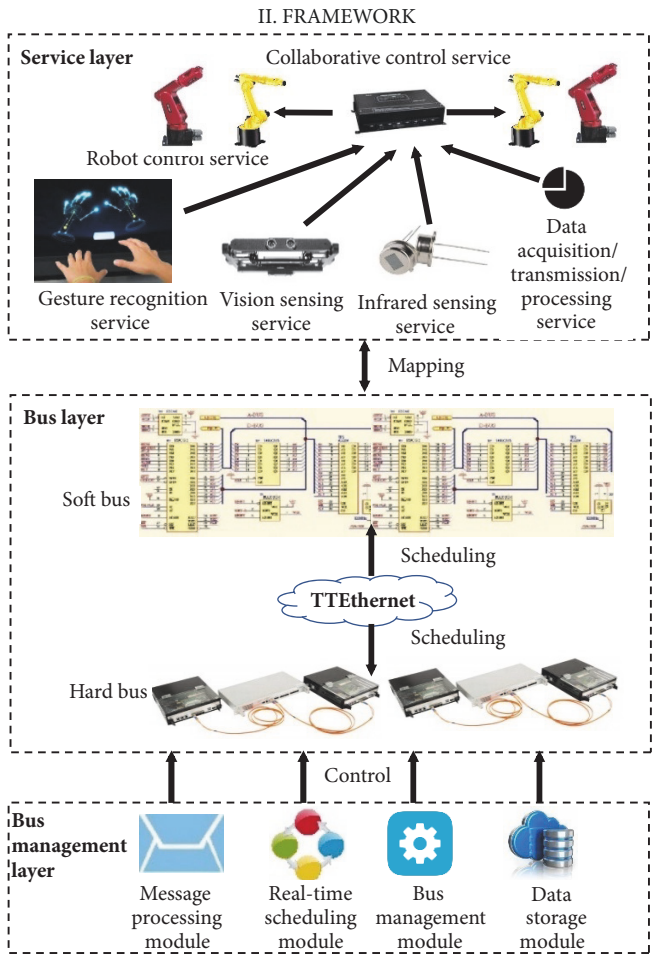


Figure 1: Architecture of software-defined distributed robot control.

accordingly. Therefore, software-defined data processing may provide a more flexible framework. Bus layer includes soft bus and hard bus. TTEthernet is very important because it schedules the bus resource. Bus management layer includes message processing module, real-time scheduling module, bus management module, and data storage module.

To prevent hotspot access from causing excessive load on the single-bus node, [42] proposed a distributed robot control system based on multibus nodes and used a load balancing algorithm to solve the problem of overload. Because of the service-oriented performance of the original control system, its loose coupling characteristics make the system expand into a multinode distributed system with advantages. The early built services can be accessed to the multibus node control system without any changes.

With the widespread use of various sensors in human-robot interaction scenes, the way people interact with robots becomes more and more abundant, such as voice interaction, gesture interaction, and force feedback interaction. A scene is set up based on hybrid sensors to control multiple robots. In the foreseeable future it can even be realized. Therefore, it is more and more important to study the distributed robot control system.

Although the introduction of multiple nodes (including bus nodes, robot nodes, and sensor nodes) has expanded the functions and application scope of the overall system, the introduction of multiple nodes in the system will cause the problem of degrading the real-time performance of message delivery. Because the original system adopts a TCP/UDP-based transmission protocol, the message transmitted by the system adopts a store-and-forward mechanism in the switch. For the high real-time control process, the forwarding of the message is likely to occur in the switch due to a long queue time. The time lag of the arrival of the control command at the control unit and the disadvantages of this type are fatal for the application in the safety-critical field.

Therefore, the proposed structure highlights the leading role of the message in the entire system.

An important feature of the improved service layer is that services can be combined with each other to build new task-solving processes. Service composition can solve the problem of low development efficiency and high repetition rate. Developers do not repeat coding; they just need to combine the existing service modules. Through this control system, both the old robot application and the new application can seamlessly interact with each other on this platform. Software

Table 1: Service description document structure.

| Element | Definition |
|---|---|
| <portType> | Services related operations, implementation functions |
| <message> | The message used by the service |
| <types> | The data type used by the service |
| <binding> | Message bus and protocol bounded to the Service |
| <QoS> | The quality of service provided by the service |

developed in different periods is available in this platform.

Table 1 describes the structure of the service description document. <types> defines the data type that service will use, whereas data types are defined with XML Schema. In doing so, XML is platform independent. <message> defines the data elements of an operation; each message can be composed of one or more components. The components in the message can be seen as the formal parameters required to write a function in the procedure. <binding> defines the protocol and format details used to provide the service. <portType> describes the operations that the service can perform and details about the parameters and messages involved. <QoS> describes the service quality of a service.

The bus layer provides various message encapsulations to the service layer. Two layers have mappings. Application services send a specific message to the bus according to their own needs. Google protocol buffers are a protocol defined by Google's internal messaging. Because the message is platform independent and based on binary, it is ideal for platforms that require high performance. According to the published test results, Google protocol buffers are 3 to 10 times smaller than traditional XML serialized messages, with 20-100 times better performance than XML on serialization and deserialization. These performance improvements are achieved with the cost of readability like the XML format. However, for the proposed system, it is worth sacrificing some readability for better performance.

Each message consists of three parts: the length of the header, the body of the header, and the body of the message. The length of the header is a 4-byte digital representation of the unsigned integer, which is read as small-end storage when fetched. The body of the header indicates the description attribute provided by the message packet. Among headers, in addition to the keyword "type" and "msgLength", the rest of the keywords are not necessary. Relevant fields are filled according to the role of the message. According to the service registration process, firstly, the service sends a message to the bus whose type is SERVICE_REGIESTER, and then the bus returns the registration service result with the message type SERVICE_RESULT.

The bus layer requires maintaining communications between the software bus and TTEthernet and regular communications between TTEthernet and hard bus. Finally, the service layer can greatly reduce the complexity of the development of the upper application, enabling service providers to focus on writing service-specific function codes and dramatically increase production efficiency.

Since the bus management layer of the prototype system requires frequent calls to the data storage module, the bus management layer and data storage module of the prototype system should be integrated in consideration of the improvement of overall system efficiency. Because each bus independently assumes all the functions in the distributed bus control system architecture, each bus node has a corresponding data storage module. Data redundancy improves, which also improves the data safety. The bus control layer of the prototype system needs to make frequent calls to the data storage layer for the registration information of the service, the routing information of the message, and the management information of the bus. So consider the efficiency of the overall system and integrate the bus control layer and data storage layer of the prototype system. And because, in the distributed bus control system structure, each bus independently assumes all the functions, each bus node has a corresponding data storage module equivalent to data for redundant backup, which also improves data security.

## 3. Model Building

*3.1. Model Complementation.* Figure 2 is a schematic diagram of the TTEthernet terminal node simulation model. It can be seen from the figure that the terminal node simulation model mainly includes four parts: a traffic application part, a sending part, a receiving part, and a scheduling and time synchronization part.

Figure 3 shows the relationship of the scheduler, sender, and receiver. The scheduling and time synchronization part is composed of the scheduling trigger module and the time synchronization module. System scheduling trigger module mainly depends on the scheduler. In scheduler part, the scheduler mainly consists of three modules: timer, oscillator, and period. The timer module is the core module of the scheduler. Timer maintains an event queue, which is used to trigger other events in the terminal node related to time modules (such as TTApp). Event triggered on the terminal node needs to be registered in the timer, like the TT sending queue, receiving queue, and so on. Oscillator is a local clock module that provides a time reference for the timer. Period provides the number of periods to the timer. That is, if the next period arrives (TT period), it will send a message to the timer to update the period. The timer will update the parameter according to this message and handle it accordingly (for example, discard the events of the previous period and issue a false warning).

The time synchronization module consists of a Sync module. Sync forwards the clock synchronization signal from the master node to the timer and forwards the clock synchronization signal to the synchronization master node during the synchronization period.

The processing of the timer is as follows:

(1) Initialize the event list in timer module, oscillator module, and period module, and get the global clock signal.

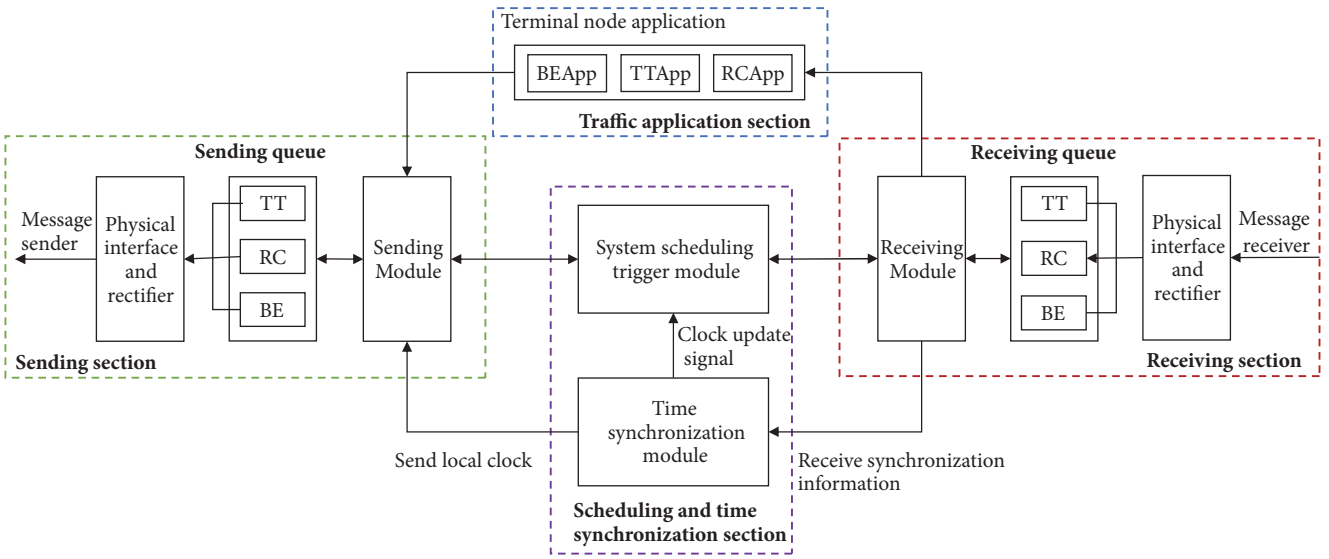


Figure 2: TTEthernet terminal node simulation model.

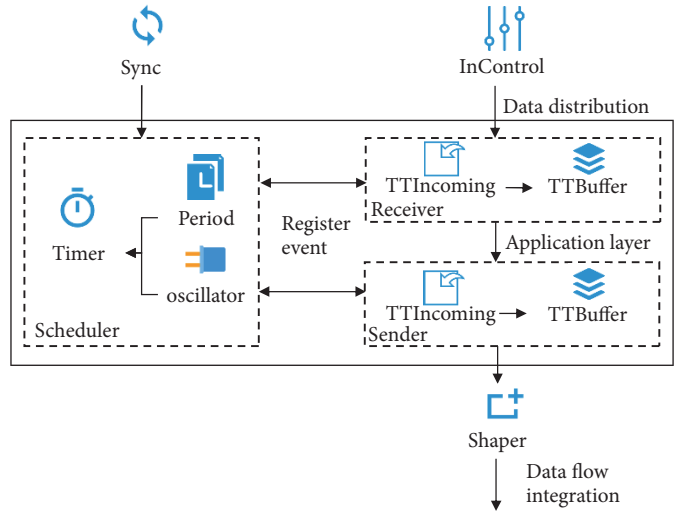


Figure 3: Relationship of scheduler, sender, and receiver.

(2) Monitor trigger message cyclically; if trigger message is received, go to step C. Otherwise, continue monitoring.

(3) Start to search the event list and send the event trigger message with the trigger time shorter than or equal to the current time to the corresponding module receiving unit. If the trigger time is less than the current time, output the event alarm signal. Otherwise, go to (2) to continue monitoring.

Take TT message as an example; in the receiver part, InControl module is used for receiving data, and the main function is to complete the diversion of different data messages. It is in the data link layer in the physical receiving layer. As can be seen in the figure; InControl sends the received TT data to the receiver that handles the TT data.

Receiver has two modules, TTIncoming and TTBuffer. The main function of TTIncoming is to ensure that the TT message can be received within the time range specified in the system schedule. If it exceeds the received time range, the message needs to be discarded and returns the alarm. Otherwise, TTIncoming sends an event to the scheduler. The event is that TT messages are forwarded to TTBuffer based on the time specified in the system's schedule. Scheduler triggers TTIncoming to send a message to the TTBuffer queue based on the time of the registration event. TTBuffer cyclically registers sending events in the scheduler according to the configuration cycle from initialization. Finally, when the TTBuffer sending event is triggered, the scheduler triggers the TTBuffer to send the TT message from the queue to the application layer.

The processing of receiver is as follows:

(1) Initialize TTIncoming and TTBuffer, and then TTBuffer registers sending trigger events according to the system's time schedule to the scheduler.

(2) TTIncoming monitors the receiving port, and if the received message is within the allowed time range, send a trigger event to the scheduler registration and go to c. Otherwise, discard the message and alarm, and then continue monitoring the port.

(3) Scheduler sends a trigger message to TTIncoming; TTIncoming forwards the received message to TTBuffer and goes to (4).

(4) TTBuffer puts the message forwarded by TTIncoming to the message queue. If TTBuffer receives the trigger message sent from the scheduler, it pops the first node in the message queue and forwards it to the corresponding APP according to the configuration file parameters. If no trigger message is received, then wait for the next scheduler trigger message to arrive.

In sending part, the functions of TTIncoming and TTBuffer are the same as those described in the receiver. Shaper module is in the link layer to deal with different data streams module. The main function of Shaper module is to integrate the three data streams and forward the data to the physical link without causing conflict according to the priority of the data stream.

The processing of the sending module is as follows:

(1) Initialize TTIncoming and TTBuffer, and TTBuffer registers sending trigger events according to the system's time schedule to the scheduler.

(2) TTIncoming monitors the received port from the application layer and registers a sending event to the scheduler if the received message is within the allowed time range and goes to (3). Otherwise, discard the message and alarm, and continue monitoring the port.

(4) Scheduler sends a trigger message to TTIncoming, and TTIncoming forwards the received message to TTBuffer and goes to (4).

(5) TTBuffer puts the message forwarded by TTIncoming into the message queue. If the TTBuffer receives the trigger message from the scheduler, then it pops the message in the message queue and forwards it to the Shaper module in the link layer and goes to (5). If no trigger message is received, then wait for the next scheduler trigger message to arrive.

(6) According to the priorities in the queue, the Shaper module forwards the messages in the queue to the physical ports from a high priority to a low priority until the message queue is empty.

As shown in Figure 4, the RC traffic application needs to periodically schedule its scheduler registration to the system because the RC traffic needs to periodically send RC traffic data according to the RC parameters (sending interval time).

The processing of RC traffic application is as follows:

(1) Initialize the application based on the system network configuration, set the rate of the RC traffic and the maximum size of the sent data packet, and register the initial triggering event in the scheduler.

(2) The news arrives. If the event is scheduler-triggered, go to (4). Otherwise, go to (3).

(3) Discard the message, go to b and wait for the next message to arrive.

(4) Generate eligible data packets according to the configuration parameters.

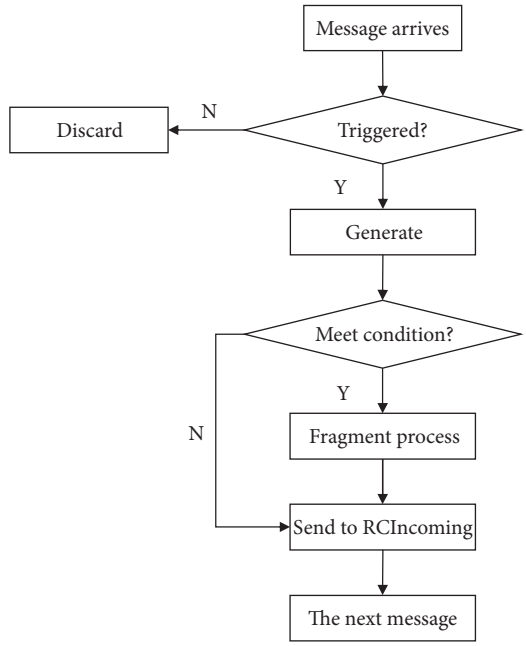


Figure 4: Process RC application.

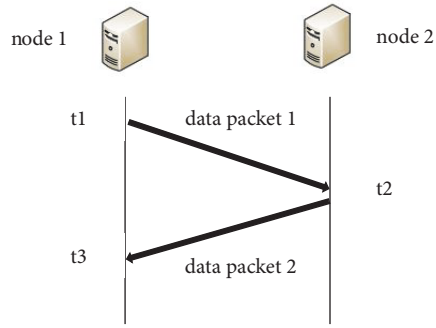


Figure 5: Network end-to-end time delay.

(5) Determine whether the current sending condition meets the condition of the fragmentation. If the condition is satisfied, go to (6). Otherwise, go to (7).

(6) Fragment the message to be sent according to the fragmentation policy.

(7) Send all messages in the buffer to RCIncoming in the sending module.

(8) According to the parameters of the rate limit, register a new trigger message in the scheduler, return to (2), and wait for the next message to arrive.

*3.2. Performance Index.* The network performance indicators in the experiment are as follows:

(1) Time delay: it is the time from the data sending the node to the data receiving node. The experimental time delay used in this paper is the one-way transmission delay from the sending node to the receiving node. As shown in Figure 5, the end-to-end time delay of packet 1 is: $Delay_{12} = t_2 - t_1$. The delay of packet 2 is $Delay_{21} = t_3 - t_2$.

(2) Network jitter: it is the transmission delay difference between the largest packet and the smallest packet over a measurement interval. The factors that affect jitter are generally related to network congestion. In simulation experiments, the closer T to zero, the smaller the network jitter and the more stable the overall network operation. *Tjitter* is calculated as follows: *Tjitter=Tdelay- TlastDelay*, where *Tdelay* is the current data stream transmission delay, and *TlastDelay* is the last data stream transmission delay.

(3) Network channel utilization: it includes the proportion of the total time that the channel is sent to the overall channel. It can be obtained through the OMNeT ++ API.

(4) Network packet loss: it is the loss of data units that cannot reach the receiving node in the transmission process. It is caused by the physical nature of physical links or network-driven defects.

## 4. Scheduling Technology

Due to the large RC data packet, the transmission cannot be completed before the arrival of the next TT data packet. The timely blocking algorithm is triggered and the RC data packet will delay sending. Because TT data traffic occupies data link bandwidth periodically, it is much likely that an RC data packet is clogged in the node's buffer zone and cannot be sent. In this case, if an RC traffic burst occurs in an application sending the RC packet, a large part of the traffic burst will be discarded by the buffer due to the size limitation of the buffer, resulting in a packet loss problem. To solve this problem, this paper presents a fixed size RC packet segmentation strategy. Figure 6 shows the flow chart of the fragment strategy algorithm. A specific description of a fixed size RC packet segmentation policy is as follows:

(1) Determine whether the packet is greater than the maximum transmission unit virtual link limit when RC data packet arrives. If the packet is bigger than the maximum transmission unit limit, the packet is divided into several data packets by the maximum transmission unit.

(2) Add all groups to the preparation queue. If the preparation queue is empty, then the process is over, or jump to the next step.

(3) Take a packet from preparation queue and obtain the time *Ttt_arrival* next TT data stream arrives. Calculate packet sending end time, and the formula is as follows:

$$\begin{aligned} &SendDuration \\ &= \frac{(msize + INTERFRAME + ((PREAMBLE + SFD) * 8))}{txRate} \\ &SendTime = TotalTime + SendDuration \end{aligned} \quad (1)$$

where *SendDuration* is the time required to transmit the packet, *SendTime* is the end of the packet transmission, *msize* is the number of bits for sending packets. INTERFRAME, PREAMBEL, and SFD are the required fields for the network interface layer whose sizes are 96 bits, 56 bits, and 8 bits, respectively. *TotalTime* is the system's current time.

(4) Determine if *Ttt_arrival* is greater than *SendDuration*; if it is, then the packet can be directly sent to the RC packet buffer queue. If not, go to the fifth step.

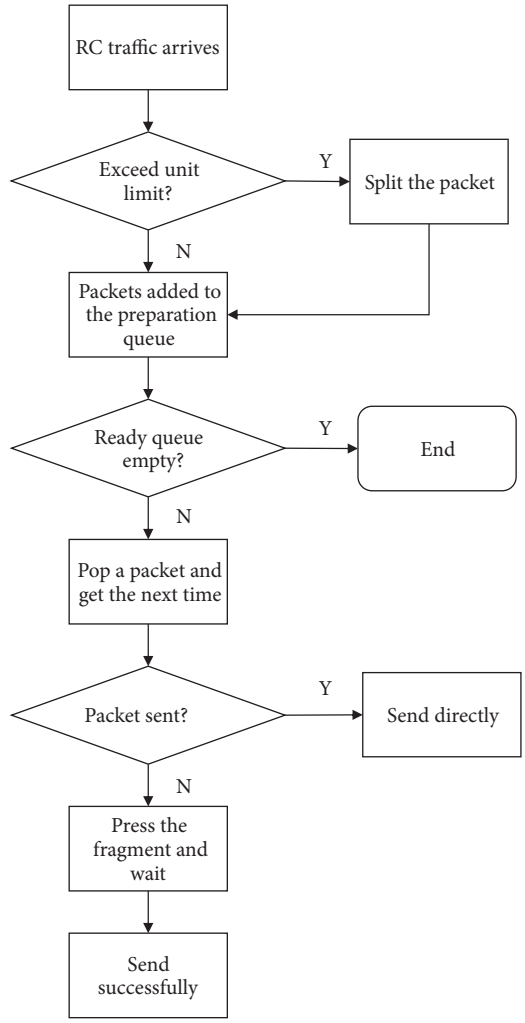


Figure 6: Process of fragment strategy algorithm.

(5) Split the packet. The current link can be used to transfer the largest unit as the basis of the fragment to avoid the problem of information redundancy.

$$\begin{aligned} &MAXUNIT \\ &= \left(TotalTime - T_{tt\_arrival} - T_{safety\_margin}\right) * txRate \\ &N = \left\lceil \frac{msize}{MAXUNIT} \right\rceil \end{aligned} \quad (2)$$

*MAXUNIT* is the maximum size of fragments that can be transmitted, and N is the number of fragments that are grouped. *Tsafety_margin* is used to guarantee that the fragment can complete the transmission.

(6) Divide the packet into N fragments according to the formula in the fifth step and place it into the buffer queue to wait for the transmission.

(7) After all the fragments have been sent, the execution process of a fixed size RC packet will go to the second step.

Table 2: TT traffic application.

| Send node | Application type | Destination | Period (ms) | Traffic size (byte) |
|---|---|---|---|---|
| co_controller | TTSource | Robot 1 | 100 | 72 |
| co_controller | TTSource | Robot 2 | 100 | 72 |
| co_controller | TTSource | Robot 3 | 100 | 72 |
| co_controller | TTSource | Robot 4 | 100 | 72 |
| sensor 3 | TTSource | co_controller | 50 | 80 |

Table 3: RC traffic application.

| Send node | Application type | Destination | Period (ms) | Bandwidth distribution slots (us) | Traffic size (byte) |
|---|---|---|---|---|---|
| Robot 1 | RCSource | co_controller | 0.8 | 350 | 390 |
| Robot 2 | RCSource | co_controller | 0.8 | 350 | 390 |
| Robot 3 | RCSource | co_controller | 0.8 | 350 | 390 |
| Robot 4 | RCSource | co_controller | 0.8 | 350 | 390 |
| Sensor 1 | RCSource | co_controller | 0.8 | 350 | 390 |
| Sensor 2 | RCSource | co_controller | 0.8 | 350 | 390 |
| Robot 3 | RCSource | co_controller | 0.8 | 350 | 390 |
| Robot 4 | RCSource | co_controller | 0.8 | 350 | 390 |

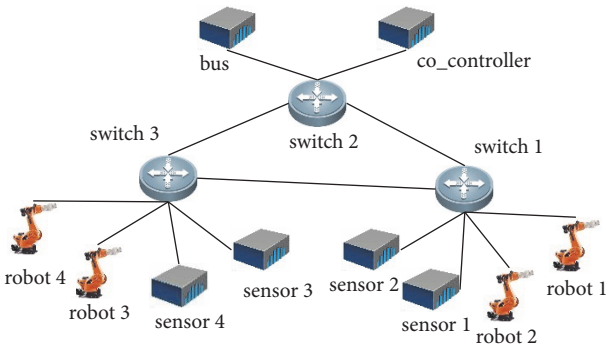


Figure 7: Simulation experiment network topology.

## 5. Experiment

*5.1. System Performance Verification Experiment.* The first experiment is a TTEthernet distributed robot system simulation verification. The main purpose is to verify whether TTEthernet-based distributed robot system can meet the real-time and deterministic requirements. The simulation topology is shown in Figure 7. There were four robots, sensors, cocontrollers, and bus nodes used to represent the TTEthernet host node. Robot 1, robot 2, sensor 1, and sensor 2 are connected to switch 1. Similarly, robot 3, robot 4, sensor 3, and sensor 4 are connected to switch 3. Bus nodes and cooperative controller are connected to switch 2. Three TTEthernet switches form the core switching network. All channels have a link length of 100Mb/s and a length of 20 meters. Four time-triggered applications are deployed in the coordinated controller. One control instruction is sent to all the four robots. That is, a total of four TT data streams are sent from the coordinated controller. The time spent from the origin device to the destination device is 20us. The entire network has a cluster period of 100ms. The switch schedules the received messages in 100us. The size of the time-triggered data is 72 Bytes. In addition, sensor 3 deployed a time-triggered traffic and was sent to the cooperative controller. The switch transmits the traffic in 200us, which sends the monitored operating status of the whole system to the coordinated controller. The detailed TT traffic application is shown in Table 2.

In addition to the TT message sending time in the cluster cycle, the remaining time channels can transmit RC messages and normal Ethernet messages. The rate limit message has higher priority than the normal Ethernet message if not in the time-triggered period; the switch will forward the rate limit message with priority. Therefore, in this experiment, a rate limiting application was deployed in sensor 1-sensor 4. The allocated bandwidth is 350us, the traffic is 390 Bytes, and message generation interval is 0.8ms. In addition, four robots are also deployed with a RC traffic, the bandwidth is 350us, the message size is 390 Bytes, and message generation interval is 0.8ms. Detailed RC traffic applications are shown in Table 3.

The four robots nodes deployed an application that provide BE service. The sending data time interval of this application is in the range of 100$\mu$s and 500$\mu$s obeying uniform distribution. The data size obeys uniform distribution in the range of 46 Bytes and 1500 Bytes. Detailed RC traffic applications are shown in Table 4.

The second experiment mainly verifies real-time performance and other related indicators about RC traffic fragmentation strategy, including time delay, jitter, network packet

Table 4: BG traffic application.

| Send node | Application type | Destin-nation | Period (ms) | Traffic size (byte) |
|---|---|---|---|---|
| Robot 1 | BGSource | bus | uniform(100,500) | uniform(46, 1500) |
| Robot 2 | BGSource | bus | uniform(100,500) | uniform(46, 1500) |
| Robot 3 | BGSource | bus | uniform(100,500) | uniform(46, 1500) |
| Robot 4 | BGSource | bus | uniform(100,500) | uniform(46, 1500) |
| Robot 1 | BGSource | Robot 2 | uniform(100,500) | uniform(46, 1500) |
| Robot 2 | BGSource | Robot 3 | uniform(100,500) | uniform(46, 1500) |
| Robot 3 | BGSource | Robot 4 | uniform(100,500) | uniform(46, 1500) |
| Robot 4 | BGSource | Robot 1 | uniform(100,500) | uniform(46, 1500) |

Table 5: Time delays and average jitter of three types of traffic.

| Time delay (us) | TT traffic (200us) | TT traffic (100us) | RC traffic | BE traffic |
|---|---|---|---|---|
| max time delay | 213.827 | 109.529 | 405.009 | 109593 |
| min time delay | 195.182 | 108.538 | 100.000 | 19.700 |
| average time delay | 201.205 | 109.037 | 142.004 | 797.909 |
| average jitter | 3.153 | 0.231 | 29.162 | 118.035 |

loss rate, and data link utilization. The fragmentation strategy was adapted based on the TTEthernet simulation model at HAW Hamburg University in Germany [43]. This experiment mainly compares two fragmentation strategies and the non-fragmentation strategy under the application environment of three different RC sending data sizes.

The time distribution of the time-triggered traffic is mainly in the interval of 100us to 200us, and the delay distribution of RC traffic is mainly distributed in the interval of 200 to 600, while the delay distribution of BE traffic is dispersed over the entire time axis. In conclusion, the time-triggered traffic has the advantage of low average latency over the remaining two traffic types, followed by RC traffic, and the BE traffic is the worst.

Table 5 shows the statistics of the maximum delay and the minimum delay for three types of traffic. The jitter time is the time delay difference of two consecutive data packets, reflecting the stability and certainty of the transmission of the different traffic data streams. The time-triggered traffic (100us) is the time-triggered traffic from the coordinated controller to the robot node. The average delay of triggered traffic is 109.037us with the average jitter of 0.231us, which is the least jitter among the three traffic types that meets the requirements of the motion control. The requirements are that the maximum delay is not higher than 1ms and the maximum time certainty is not higher than 1us [6]. Moreover, as the monitoring signal, the time-triggered traffic is 200us; the average delay is 201.205us with the average jitter of 3.153us, which follows the control, and has met the time certainty of monitoring. Due to TTEthernet's regulation, the average delay and average jitter of RC traffic and BE traffic are lower than the time-triggered traffic. RC traffic can transmit video and other data that is not deterministic to the data flow. Transfer bus service management data.

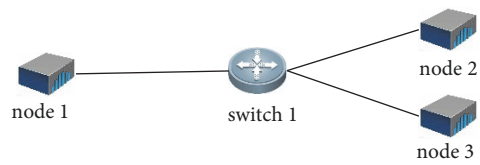


Figure 8: Simulation experiment network topology.

Through the analysis of the experimental data, it can be verified that the TTEthernet distributed robot control system meets the robot control requirements. From the above data, it can be obtained that, in time-triggered Ethernet simulation, time-triggered traffic has an advantage of lower latency and higher certainty. Time-trigger traffic is suitable for real-time and deterministic data. The BE data can be used to transfer application data with low requirements of real-time performance. RC traffic is a compromise between time-triggered and BE data.

*5.2. Fragmentation Strategy Verification Experiment.* The second experiment, the fragmentation strategy verification experiment, mainly verifies the relevant indicators such as real-time performance of the RC traffic fragment strategy, including time delay, jitter, network packet loss rate, and data link utilization. Figure 8 shows the topological structure of the simulation application, which contains three end nodes and one switch node. Each terminal node is connected with the switch node; the data link data transfer rate is 100Mbits/s. The simulation time of the experiment is 10s. In this experiment, two TT cycles, C1 and C2, are defined, the period of C1 is 100us, the period of C2 is 1000us, and there is a bias of 30us (that is, every C2 period starts from ($i * 1000 + 30$, $i = 2...$) us). If the least common multiple of C1 and C2 as a

Table 6: TT traffic application.

| Terminal node | Application type | Destination | Period | Traffic size (Byte) |
|---|---|---|---|---|
| Node 1 | TTSource | Nodes 2, 3 | $C_1$ | 46 |
| Node 1 | TTSource | Node 2 | $C_2$ | 46 |
| Node 2 | TTSource | Node 1 | $C_2$ | 46 |

Table 7: Comparison of three environmental parameters.

| scene | Application type | Traffic size (Byte) | SI(us) | VL(us) |
|---|---|---|---|---|
| 1 | RCSource/RCSource_frag | Uniform(46,357) | 5000 | 1000 |
| 2 | RCSource/RCSource_frag | Uniform(357,1232) | 5000 | 1000 |
| 3 | RCSource/RCSource_frag | Uniform(1232,1500) | 5000 | 1000 |

SI: sending interval; VL: virtual link.

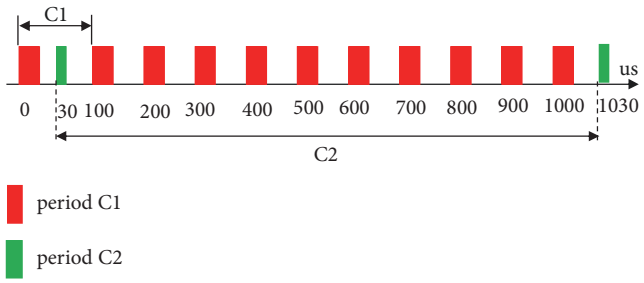


Figure 9: Network TT cycle integration.

Table 8: Average time delay of RC traffic in three scenarios.

| Average time delay (s) | scene 1 | scene 2 | scene 3 |
|---|---|---|---|
| RCSource | 4.73288E-05 | #NaN | #NaN |
| RCSource_frag | 4.52014E-05 | 0.000354399 | 0.00062752 |

complete period from the perspective of a data link is taken, the TT traffic occupancy in the data link can be obtained as shown in the figure. In addition, the configuration file also defines that the length of buffer queues should be no longer than 2500. For bursts of large data streams, it is likely that due to the limitation of buffer queue size a higher packet loss rate will result.

Figure 9 shows the result of integrating two TT cycles into the whole network. It can be noticed that the smallest interval in the network occurs in the interval of 1000us increments (0us, 30us and 1000us, 1030us). If the data size transmitted in the data link of the TT cycle is the smallest data unit (46 Bytes), i.e., the smallest Ethernet frame (64 Bytes), the maximum data frame transmitted in this interval is about 375 Bytes in size (which should be less than 375 Bytes in actuality). Similarly, in the interval (100us, 200us) and other increments of 1000us, the maximum size of the data is about 1250 Bytes (actually less than 1250 Bytes).

Table 6 describes the application of TT traffic deployed in the terminal node in this lab. TT data stream in the distributed robot control system mainly controls signals such as real-time performance and high deterministic signal.

This experiment mainly verifies the performance of RC traffic subcontracting strategy, so this paper compares the performance of the three applications. The application of the RC traffic is deployed on Node 2, and the RCSource and RCSource_frag correspond to the RC traffic application without fragmentation strategy and with fragmentation strategy, respectively. In addition to the two applications with or without fragmentation strategy, the other parameters are the same. Table 7 lists the three application scenarios for comparison. The main difference is that the traffic is differentiated according to the load. The other parameters are the same. The traffic volume of the three environments follows a uniform distribution. The parameters are designed according to the maximum and minimum transmittable data frame sizes in the integrated TT cycle mentioned above.

Table 8 shows the average delay of RC traffic in three scenarios. It can be seen from the data that in all three scenarios the average delay of RC traffic with fragmentation strategy is better than that of the original model. It can be seen from the simulation results that the maximum experimental latency with fragmentation strategy under scene 1 is 107.37us, the minimum is 19.59us, and the difference is 87.78. The maximum experimental delay without fragmentation strategy was 135.03us, the minimum was 19.59us, and the difference was 115.44. In scene 1, from the perspective of average delay, it can be concluded that the performance of the fragmented strategy is improved by 4.5% compared with the nonfragmented strategy.

The appearance of #NaN in the table indicates that the simulation platform does not detect the related traffic; that is, in scene 2 and scene 3, the RC application data stream without the fragment strategy fails to reach the target node. It can be found in the log file through the simulation platform that, as the data generated by the RC application at the beginning of the TT period gap cannot be successfully sent (i.e., greater than the maximum transmittable data frame size, about 1250 Bytes), the data frame has been clogged in buffer queue. Consequently, the following data frames cannot be sent.

From the data of scene 2 and scene 3, it can be found that the RC application of the original model in TTEthernet is not robust to the larger RC data frames, resulting in a waste of

Table 9: Average jitter of RC traffic in three scenarios.

| Average time delay (s) | scene 1 | scene 2 | scene 3 |
|---|---|---|---|
| RCSource | 2.08913E-05 | #NaN | #NaN |
| RCSource_frag | 1.8761E-05 | 0.000307254 | 0.000649226 |

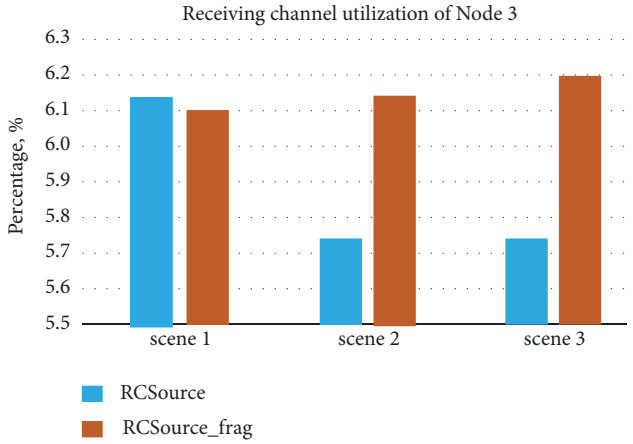


Figure 10: Receiving channel utilization of Node 3.

data link bandwidth. This is also the reason for the proposed RC data fragmentation strategy.

Table 9 shows the average jitter of RC traffic in three scenarios. It can be seen from the data that, in all three scenarios, the average jitter time of the RC traffic application with fragmentation strategy is due to the RC application of the original model. In scene 1, the average RC jitter of the original model is 20.89$\mu$s, that of the RC application with fragmentation strategy is 18.76$\mu$s, and that of the RC model with fragmentation strategy is 10.2% higher than the jitter performance of the original model. Similarly, due to the problem of congestion in the buffer queue, the RC traffic without the original model RC application is successfully received.

The utilization of Node 3 receiving channel in the simulation network is shown in Figure 10. Node 3 is the only node that receives the RC traffic in the simulation network. The channel utilization of original model and improved model can be analyzed by examining the receiving channel utilization of the node. It can be obtained from scene 1 that the channel utilization of the original model is 6.12% and the channel utilization of the improved model is 6.10%, which is considered to be consistent. In scene 2 and scene 3, the channel utilization rate of the original model dropped sharply to 5.76% due to the failure that sends the RC traffic. In scenarios 2 and 3, the channel utilization of the improved model is 6.12% and 6.20%, respectively. The reason for the higher channel utilization in scene 3 lies in the fact that the size of the data to be sent is concentrated on larger data frames (1232 Bytes, 1500 Bytes). Moreover, due to the use of substrategy, channel utilization is improved.

Table 10: Packet loss in simulation network in three scenarios.

| Packet loss | scene 1 | scene 2 | scene 3 |
|---|---|---|---|
| RCSource | 0 | 0 | 0 |
| RCSource_frag | 0 | 0 | 662 |

Finally, we acquire the packet loss by examining the simulation network, which is shown in Table 10. For scene 1 and scene 2, the result indicates that both models have zero packet loss. However, in scene 3, packet loss occurs in applications which use the fragment strategy, accounting for 16.02% of the total data frames sent. The reason for packet loss is that a large data frame is divided into several data frames due to the fragment strategy. If the buffer queue length is short, data packet loss occurs. Therefore, the use of fragment strategies needs to consider the size of the buffer and transmit the data with fewer certainty requirements.

## 6. Conclusion

This paper provides a framework for a bus-based intelligent robot system and designs scheduling algorithms to make TTEthernet play the role of scheduling in the framework. TTEthernet is proposed as the system's transmission protocol which has improved the real-time and deterministic performance in the system. TTEthernet overcomes the problems of low speed and short transmission distances and has certain real-time and reliability performance.

A bus-based intelligent robot system framework is put forward and a scheduling algorithm is designed. The framework can achieve real-time performance and deterministic performance for distributed robot systems. In addition, because of the problem of rate limiting traffic which causes a large delay, a fragment strategy is proposed to solve this problem.

Experimental results indicate that the improved scheme based on fragmentation strategy proposed in this paper can improve the real-time performance of RC traffic to a certain extent but due to network node buffer queue restrictions will lead to packet loss of fragmentation strategy when there is a large data flow of the problem; the application of certainty is not applicable. In addition, the original simulation model is not optimized for the virtual link routing. For the simulation of large networks, there can be an improvement in the TTEthernet virtual link routing problem. These issues need further analysis and resolution.

## Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

This work is partially supported by National Natural Science Foundation of China (no. 61602182), Science and Technology

Planning Project of Guangzhou, China (no. 201604046029), Guangdong Natural Science Funds for Distinguished Young Scholar (2017A030306015), Pearl River S&T Nova Program of Guangzhou (201710010059), Guangdong Special Projects (2016TQ03X824), and the Fundamental Research Funds for the Central Universities (no. 2017JQ009).

## References

[1] X. Li, D. Li, J. Wan, A. V. Vasilakos, C.-F. Lai, and S. Wang, "A review of industrial wireless networks in the context of Industry 4.0," *Wireless Networks*, vol. 23, no. 1, pp. 23–41, 2017.

[2] G. Du, P. Zhang, and D. Li, "Human-manipulator interface based on multisensory process via kalman filters," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 10, pp. 5411–5418, 2014.

[3] M. FRUSTACI, P. PACE, G. ALOI, and G. FORTINO, "Evaluating critical security issues of the IoT world: Present and Future challenges," *IEEE Internet of Things Journal*, 2017.

[4] M. Asplund and S. Nadjm-Tehrani, "Attitudes and Perceptions of IoT Security in Critical Societal Services," *IEEE Access*, vol. 4, pp. 2130–2138, 2016.

[5] H. B. Salameh, S. Almajali, M. Ayyash, and H. Elgala, "Security-aware channel assignment in IoT-based cognitive radio networks for time-critical applications," in *Proceedings of the 4th International Conference on Software Defined Systems, SDS 2017*, pp. 43–47, Spain, May 2017.

[6] M. Felser, "Real-time ethernet - Industry prospective," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1118–1129, 2005.

[7] G. Du and P. Zhang, "Online serial manipulator calibration based on multisensory process via extended kalman and particle filters," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 12, pp. 6852–6859, 2014.

[8] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[9] M. S. Branicky, S. M. Phillips, and W. Zhang, "Scheduling and feedback co-design for networked control systems," in *Proceedings of the 41st IEEE Conference on Decision and Control*, pp. 1211–1217, Las Vegas, Nev, USA, December 2002.

[10] S. Dai, H. Lin, and S. S. Ge, "Scheduling-and-control codesign for a collection of networked control systems with uncertain delays," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 1, pp. 66–78, 2010.

[11] W. Jie and L. Wei-dong, "Control and scheduling co-design of networked control system," in *Proceedings of the 2011 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, pp. 1–4, Xi'an, Shaanxi, China, September 2011.

[12] R. Serna Oliver, S. S. Craciunas, and G. Stoger, "Analysis of Deterministic Ethernet scheduling for the Industrial Internet of Things," in *Proceedings of the 2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, CAMAD 2014*, pp. 320–324, Greece, December 2014.

[13] J. Li and M. Chen, "Multiobjective Topology Optimization Based on Mapping Matrix and NSGA-II for Switched Industrial Internet of Things," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1235–1245, 2016.

[14] S. Fuchs, H.-P. Schmidt, and S. Witte, "Test and on-line monitoring of real-time Ethernet with mixed pysical layer for Industry 4.0," in *Proceedings of the 21st IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2016*, Germany, September 2016.

[15] P. Danielis, J. Skodzik, V. Altmann et al., "Survey on real-time communication via ethernet in industrial automation environments," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pp. 1–8, Barcelona, Spain, September 2014.

[16] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0," *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 17–27, 2017.

[17] G. S. Sestito, A. C. Turcato, A. L. Dias et al., "A Method for Anomalies Detection in Real-Time Ethernet Data Traffic Applied to PROFINET," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2171–2180, 2018.

[18] G. Carvajal, C. W. Wu, and S. Fischmeister, "Evaluation of communication architectures for switched real-time ethernet," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 63, no. 1, pp. 218–229, 2014.

[19] S. Fuchs, A. Gercikow, and H. Schmidt, "Monitoring of real-time behavior of industrial ethernet for industry 4.0," in *Proceedings of the 2017 International Electrical Engineering Congress (iEECON)*, pp. 1–4, Pattaya, Thailand, March 2017.

[20] X. Li, D. Li, J. Wan, C. Liu, and M. Imran, "Adaptive Transmission Optimization in SDN-Based Industrial Internet of Things With Edge Computing," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1351–1360, 2018.

[21] J. Wan, S. Tang, Z. Shu et al., "Software-Defined Industrial Internet of Things in the Context of Industry 4.0," *IEEE Sensors Journal*, vol. 16, no. 20, pp. 7373–7380, 2016.

[22] Z. Shu, J. Wan, D. Li, J. Lin, A. V. Vasilakos, and M. Imran, "Security in Software-Defined Networking: Threats and Countermeasures," *Mobile Networks and Applications*, vol. 21, no. 5, pp. 764–776, 2016.

[23] Z. Shu, J. Wan, J. Lin et al., "Traffic Engineering in Software-Defined Networking: Measurement and Management," *IEEE Access*, vol. 4, pp. 3246–3256, 2016.

[24] G. Du and P. Zhang, "A markerless human-robot interface using particle filter and kalman filter for dual robots," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 4, pp. 2257–2264, 2015.

[25] H. Fischer, P. Vulliez, J.-P. Gazeau, and S. Zeghloul, "An industrial standard based control architecture for multi-robot real time coordination," in *Proceedings of the 14th IEEE International Conference on Industrial Informatics, INDIN 2016*, pp. 207–212, France, July 2016.

[26] A. C. Kapoutsis, S. A. Chatzichristofis, L. Doitsidis et al., "Real-time adaptive multi-robot exploration with application to underwater map construction," *Autonomous Robots*, vol. 40, no. 6, pp. 987–1015, 2016.

[27] P. Das, S. Pradhan, H. Tripathy, and P. Jena, "Improved real time A*-fuzzy controller for improving multi-robot navigation and its performance analysis," *International Journal of Data Science*, vol. 2, no. 2, p. 105, 2017.

[28] Z. Lukasik, B. Pniewska, R. Pniewski, and Z. Łukasik, *Rozproszony system sterowania robotem mobilnym*, IEEE, 2009.

[29] D. Kim, Y. Doh, and Y.-H. Lee, "Table driven proportional access based real-time Ethernet for safety-critical real-time systems," in *Proceedings of the Pacific Rim International Symposium on Dependable Computing, PRDC 2001*, pp. 356–363, Republic of Korea, December 2001.

[30] S. YANG, "Safety Critical Real-Time Networks Based on Ethernet Technology," *Journal of Software* , vol. 16, no. 1, p. 121, 2005.

[31] A. Ademaj, H. Kopetz, P. Grillinger, K. Steinhammer, and M. Prammer, "Integration of Predictable and Flexible In-Vehicle Communication using Time-Triggered Ethernet," in *Proceedings of the SAE 2006 World Congress & Exhibition*.

[32] W. Steiner, G. Bauer, B. Hall, M. Paulitsch, and S. Varadarajan, "TTEthernet dataflow concept," in *Proceedings of the 2009 8th IEEE International Symposium on Network Computing and Applications, NCA 2009*, pp. 319–322, USA, July 2009.

[33] A. Loveless, "On TTEthernet for integrated Fault-Tolerant spacecraft networks," in *Proceedings of the AIAA SPACE Conference and Exposition, 2015*, USA, September 2015.

[34] T. Steinbach, H.-T. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz, "Tomorrow's in-car interconnect? a competitive evaluation of IEEE 802.1 AVB and time-triggered ethernet (AS6802)," in *Proceedings of the 76th IEEE Vehicular Technology Conference, VTC Fall 2012*, Canada, September 2012.

[35] B. Chen, J. Wan, L. Shu, P. Li, M. Mukherjee, and B. Yin, "Smart Factory of Industry 4.0: Key Technologies, Application Case, and Challenges," *IEEE Access*, vol. 6, pp. 6505–6519, 2018.

[36] D. Tămaş–Selicean, P. Pop, and W. Steiner, "Design optimization of TTEthernet-based distributed real-time systems," *Real-Time Systems*, vol. 51, no. 1, 2014.

[37] N. I. Aristova, "Ethernet in industrial automation: Overcoming obstacles," *Automation and Remote Control*, vol. 77, no. 5, pp. 881–894, 2016.

[38] M. Selvatici, M. Ruggeri, and L. Dariz, "Advanced gateway services for real-time in-vehicle ethernet network," in *Proceedings of the 2015 IEEE International Conference on Industrial Technology, ICIT 2015*, pp. 1826–1831, Spain, March 2015.

[39] J. C. Yang et al., "Research towards IoT-oriented universal control system security model," *Journal of Communications*, 2012.

[40] D. Tamas-Selicean, P. Pop, and W. Steiner, "Synthesis of communication schedules for TTEthernet-based mixed-criticality systems," in *Proceedings of the the eighth IEEE/ACM/IFIP international conference*, p. 473, Tampere, Finland, October 2012.

[41] G. L. Du, P. Zhang, and X. Liu, "Markerless humanmanipulator interface using leap motion with interval Kalman filter and improved particle filter," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 2, pp. 694–704, 2016.

[42] P. F. Jin, VxWorks-based Software Bus and Its Autonomous Recovery Technology,.

[43] T. Steinbach, H. Dieumo Kenfack, F. Korf, and T. Schmidt, "An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy," in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, Barcelona, Spain, March 2011.