

Distributed Real-Time Systems on Responsive Link

Hiroyuki Chishiro and Nobuyuki Yamassaki

Department of Information and Computer Science, Keio University

3-14-1, Hiyoshi, Kouhoku-ku Yokohama Kanagawa, Japan, 223-8522 Phone: +81-45-560-1060, FAX: +81-45-560-1061

Received: Month dd, yyyy / Accepted: Month dd, yyyy / Published: Month dd, yyyy.

Abstract: Responsive Link is a communication standard as specified in ISO/IEC 24740:2008 for distributed real-time systems. Responsive Link has multiple features to meet both hard and soft real-time requirements. Unfortunately, current real-time operating systems do not support the main features of Responsive Link. In addition, distributed real-time systems in noisy environments require revealing the tolerance of Responsive Link to achieve correct communication. We present Application Program Interfaces (APIs) for Responsive Link in RT-Est real-time operating system. Using the APIs, users can make use of the main features of Responsive Link easily. Experimental evaluations perform the voltage tolerance test against noise and reveal the tolerance of Responsive Link.

Key words: Responsive Link, Responsive Multithreaded Processor, Distributed Systems, Real-Time Communication, Tolerance.

1. Introduction

Robots [1] in distributed real-time systems have many nodes and require various topologies. In distributed real-time environments such as deep sea and outer space, there is much noise to interrupt correct communication. In addition, packets must be arrived by their deadlines in such noisy environments. However, existing communication standards have various limitations such as real-time requirement, topology and tolerance.

For example, wireless communication standards such as WirelessHART [2] reduce tolerance compared to wired communication standards. On the other hand, wired communication standards such as Controller Area Network (CAN) [3] and FlexRay [4] suffer from the limitations of topologies. Since there are various shapes of robots, we require a topology-free communication standard. Also, these communication standards have priority inversion problems [5] because higher priority packets cannot overtake lower priority packets. Due to priority inversion problems, higher priority packets may miss their deadlines. In order to

overcome the weakness of these communication standards, we have developed Responsive Link [6].

Responsive Link is a communication standard as specified in ISO/IEC 24740:2008 for distributed real-time systems. Responsive Link has multiple features to meet both hard and soft real-time requirements. For example, Responsive Link has two communication links: event link and data link. An event link is used to transmit packets with hard real-time requirements. On the other hand, a data link is used to transmit packets with soft real-time requirements. By the separate transmission of the event link and the data link, Responsive Link is easier to support hard and soft real-time communication than other communication standards. Unfortunately, current real-time operating systems do not support the main features of Responsive Link. In addition, distributed real-time systems in noisy environments require revealing the tolerance of Responsive Link to achieve correct communication.

We present Application Program Interfaces (APIs) for Responsive Link in RT-Est real-time operating system [7]. Using the APIs, users can make use of the main features of Responsive Link easily. Experimental evaluations perform the voltage

Corresponding author: Hiroyuki Chishiro (1985-), Ph.D., visiting researcher, research fields: real-time systems. Email: chishiro@ny.ics.keio.ac.jp.

tolerance test against noise and reveal the tolerance of Responsive link.

The contribution of this paper is to implement APIs for Responsive Link and to build the experimental environments for tolerance. We believe that the APIs for Responsive Link help users to build distributed real-time systems easily and the method to evaluate the tolerance of communication standards is widely used.

The remainder of this paper is organized as follows: Section 2 introduces the detail of Responsive Link. Section 3 presents the implementation of the APIs for Responsive Link. The effectiveness of Responsive Link is evaluated in Section 4. Section 5 compares our work with related work in distributed real-time systems. Finally we offer concluding remarks in Section 6.

2. Responsive Link

In this section, we introduce the detail of main features in Responsive Link [6].

2.1 Packet Overtaking

Real-time systems require real-time scheduling, which guarantees completing real-time tasks by their deadlines. In real-time scheduling, higher priority tasks can preempt lower priority tasks. In order to achieve this preemption in real-time communication, higher priority packets require overtaking lower priority packets in each node. Therefore, each packet in Responsive Link has a priority. If packets have the same priority, then the round-robin rule is applied. By the technique of the packet overtaking, real-time scheduling such as fixed-priority scheduling [8] and semi-fixed-priority scheduling [9,10] can be adapted to packet scheduling.

2.2 Format of Packet

Responsive Link supports two kinds of packets: event packet and data packet. Fig. 1 shows the formats of the event packet and the data packet. An event

packet is 16-byte length consisted of a 4-byte header, an 8-byte payload and a 4-byte trailer. The main use of the event packet is to transmit control commands and operations with hard real-time requirements. A data packet is 64-byte length consisted of a 4-byte header, a 56-byte payload and a 4-byte trailer. The main use of the data packet is to transmit image and sound data with soft real-time requirements.

Header		Trailer	
Source Node Address 2 bytes	Destination Node Address 2 bytes	Payload 8 bytes	Control and status 4 bytes
Source Node Address 2 bytes	Destination Node Address 2 bytes	Payload 56 bytes	Control and status 4 bytes

Fig. 1 Formats of event packet (upper) and data packet (lower).

Source Node Address		Destination Node Address	
Priority[7-4] 4 bits	Source Address 12 bits	Priority[3-0] 4 bits	Destination Address 12 bits

Fig. 2 Format of header.

Fig. 2 shows the format of the header. A packet has a header that includes a 16-bit source node address and a 16-bit destination node address. The source node address includes a 4-bit priority[7-4] and a 12-bit source address. On the other hand, the destination node address includes a 4-bit priority[3-0] and a 12-bit destination address. The priority[7-4] has the 7-4 bits of the 8-bit priority and the priority[3-0] has the 3-0 bits of the 8-bit priority. Larger values have higher priorities in packets. The highest priority is 255 (0xff) and the lowest priority is 0 (0x00).

Byte	0	1	2	3	4	5	6	7
0	UD	Full	Data Length					
1	Dirty0	Dirty1	Dirty2	Dirty3	Dirty4	Dirty5	Dirty6	Dirty7
2	Dirty8	Dirty9	Dirty10	Dirty11	Dirty12	Dirty13	Dirty14	Dirty15
3	Start	End	Int.	Fatal	Correct	Serial Number		

Fig. 3 Format of trailer.

Fig. 3 shows the format of the trailer. The trailer includes the following parameters.

- User Defined (UD): sets this bit to the user defined data.

- Full: sets this bit to 1 if all payload data are valid (data packet: 56 bytes, event packet: 8 bytes). Otherwise, set this bit to 0.
- Data Length: indicates the length of the valid payload data (data packet: 0-56, event packet: 0-8).
- Dirty[0-15]: indicates which word (4 bytes) in the packet has an error in case of a data packet or which byte in the packet has an error in case of an event packet. For example, if the 3rd word of the data packet has an error, then set the Dirty2 bit to 1. If the 4th word of the event packet has an error, then set the Dirty3 bit to 1.
- Start: sets this bit to 1 if this packet is the start packet. Otherwise, set this bit to 0.
- End: sets this bit to 1 if this packet is the end packet. Otherwise, set this bit to 0.
- Interrupt (Int.): generates an interrupt when this packet arrives at the destination node if this bit is set to 1. Otherwise, set this bit to 0.
- Fatal: sets this bit to 1 by hardware if one byte in the packet has an unrecoverable fatal error. Otherwise, set this bit to 0.
- Correct: sets this bit to 1 by hardware if this packet has an error that has been corrected. Otherwise, set this bit to 0.
- Serial Number: indicates the serial number. The start packet has the serial number 0 and the serial number is incremented in the following packets. The serial number returns 0 after it arrives 7 and the sequence is repeated.

2.3 Routing

Responsive Link achieves an end-to-end connection by setting the routing tables of all nodes along the transmission path from a source node to a destination node. Fig. 4 shows the routing table of Responsive Link. Each node has a routing table to control the route of the packet and the priority exchange function. Each ID in the routing table has the 32-bit reference part and the 16-bit referent part. The 32-bit reference

part is the same as the header of the packet, as shown in Fig. 2. On the other hand, the 16-bit referent part includes the following parts.

- Event Enable (EE): sets this bit to 1 if the event link is valid. Otherwise, set this bit to 0.
- Data Enable (DE): sets this bit to 1 if the data link is valid. Otherwise, set this bit to 0.
- Priority Exchange (PE): indicates whether the priority exchange function is valid.
- Priority[7-0] (P[7-0]): includes the new priority level (8 bits) which is valid if PE is set to 1.
- Link[4-0] (L[4-0]): indicates the output link numbers. The L[4-1] bits indicate the output link numbers of four physical ports in Responsive Link. On the other hand, the L[0] bit indicates the output link number to Dual Port Memory (DPM) in a processor. If multiple bits are set to 1, then multicast is indicated. If all bits are set to 1, then broadcast is indicated.

Since the EE bit and the DE bit can be set independently, the event link and the data link may have different routes to the same destination.

Responsive Link supports the layer 1 to the layer 4 in OSI reference model. Using Responsive Link, we can implement communication protocols such as TCP with low overhead, compared to software implementation using other communication standards.

Unfortunately, current real-time operating systems do not support the main features of Responsive Link. Therefore, the APIs for Responsive Link are required to build distributed real-time systems easily.

3. Implementation

We implement APIs for Responsive Link in RT-Est real-time operating system [7] in C language. Using the APIs, users can make use of the main features of Responsive Link easily.

3.1 Initialization

We implement the device driver of Responsive Link in `init_resplink` function. This function

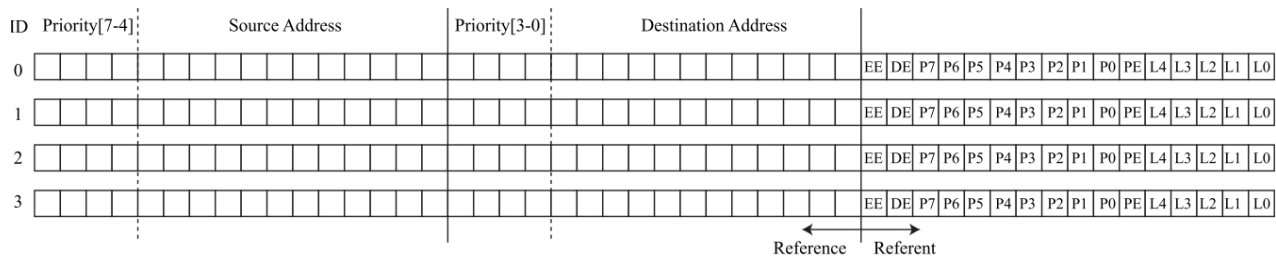


Fig. 4 Routing table.

initializes the following parameters.

- SDRAM: sets the size of SDRAM for the packet overtaking by the priority of each packet selected with [None, {8,16,32,64,128,256}MB].
- Speed: sets the speed of Responsive Link selected with [{50,100,200,400,800}Mbps].
- Switch: initializes the switch, the encoder and the decoder of Responsive Link. The switch of Responsive Link supports the following modes.
 - Cut Through Mode: The switch starts to forward a packet before the whole packet has been received, as soon as the destination address is processed. This technique has the advantage of reducing latency and the disadvantage of decreasing tolerance.
 - Store and Forward Mode: The switch starts to forward a packet to a node where the packet is kept and sent at a later time to the destination node or another intermediate node.
- Interrupt: clears all interrupts of both the event link and the data link and enables a proper interrupt by users. For example, Responsive Link supports the following End Of Packet (EOP) interrupts.
 - Down: The cable of Responsive Link is unplugged.
 - Wakeup: The cable of Responsive Link is plugged.
 - Fatal: A fatal error occurs.
 - Data-Out EOP: occurs if sending data packets in the range of the specified DPM.
 - Data-In EOP: occurs if receiving data packets in the range of the specified DPM.
 - Data Packet-In: occurs if receiving data packets, which enable their interrupt bits.
 - Event-Out EOP: occurs if sending event packets in the range of the specified DPM.
 - Event-In EOP: occurs if receiving event packets in the range of the specified DPM.
 - Event Packet-In: occurs if receiving event packets, which enable their interrupt bits.
- Link[4-0] (L[4-0]): indicates the output link numbers, as described in Subsection 2.3.
- Serial/Parallel[4-0]: indicates serial/parallel connection in each output link number. If an output link is parallel connection, set this bit to 1. If an output link is serial connection, set this bit to 0.
- DPM: initialize event in/out and data in/out registers.
 - Transmission Mode: If the transmission mode is Mode0, set this bit to 0. In Mode0, each packet includes each header and trailer. If transmission mode is Mode1, set this bit to 1. In Mode1, first of all, Responsive Link transmits the payload of each packet at the head of the DPM sequentially. After the payload of each packet, Responsive Link next transmits the header and the trailer of each packet sequentially. Therefore, all packets arrive at the same destination node address.
 - Interrupt: If this bit is set to 1, an EOP interrupt occurs.
 - Dreq: If this bit is set to 1, perform Direct Memory Access (DMA) transfer of the packet by the DMA counter.

- From_Addr/To_Addr: These control registers have same features. For ease of comprehension to users, the name of each control register is different.

Users can configure the above parameters for target distributed real-time applications. Therefore, Responsive Link can support more various types of distributed real-time applications than other communication standards.

3.2 Packet

Now we explain the implementation of the event packet and the data packet.

```
struct resplink_header {
    unsigned int high_priority:4;
    unsigned int src_addr:12;
    unsigned int low_priority:4;
    unsigned int dst_addr:12;
} __attribute__((packed));
```

Fig. 5 struct resplink.

Fig. 5 shows struct resplink_header. As shown in Fig. 2, struct resplink_header includes a 4-bit high_priority, a 12-bit src_addr, a 4-bit low_priority and a 12-bit dst_addr. In addition, struct resplink_header has the packed type attribute.

```
struct eventlink_packet {
    union {
        uint32_t addr;
        struct resplink_header header;
    } u;
    uint8_t payload[8];
    uint32_t trailer;
};
```

Fig. 6 struct event_packet.

Fig. 6 shows struct event_packet. As shown in Fig. 1, struct event_packet has a 4-byte header, a 8-byte payload and a 4-byte trailer. In struct

eventlink_packet, union u has a 4-byte address and struct resplink_header. For example, this 4-byte address is used to write header to the reference part in routing tables and this header is used to set the priority, the source address and the destination address.

```
struct datalink_packet {
    union {
        uint32_t addr;
        struct resplink_header header;
    } u;
    uint8_t payload[56];
    uint32_t trailer;
};
```

Fig. 7 struct data_packet.

Fig. 7 shows struct data_packet. As shown in Fig. 1, struct data_packet has a 4-byte header, a 56-byte payload and a 4-byte trailer. Like struct event_packet in Fig. 6, struct data_packet also has union u.

```
void send_eventlink(unsigned int mode,
    struct event_packet *ep);
void send_datalink(unsigned int mode,
    struct data_packet *dp);
int recv_eventlink(uint8_t *buf);
int recv_datalink(uint8_t *buf);
```

Fig. 8 send/receive functions.

Fig. 8 shows send/receive functions in the event link and the data link. The first argument of send_eventlink and send_datalink functions is mode, which selects Mode0 or Mode1 if the first argument is 0 or 1 respectively. The second argument of send_eventlink is the pointer of struct event_packet. Also, the second argument of send_datalink is the pointer of struct data_packet. The first argument of recv_eventlink and recv_datalink functions is the pointer of the 1-byte character. This

pointer indicates the head of the buffer receiving event packets and data packets respectively.

3.3 Routing Table

```
void add_rtable(uint32_t reference,
               uint16_t referent);
void delete_rtable(unsigned int id);
```

Fig. 9 add/delete functions.

Fig. 9 shows add/delete functions to add/delete routing tables respectively. `add_rtable` function adds a rule to both reference and referent parts in the routing table, as shown in Fig. 4. Then the rule is set to the unique identifier. On the other hand, `delete_rtable` function deletes the rule selected by `id`.

Note that if registering the same rules of routing tables, the rule with the smallest `id` in the same rules is applied to routing packets.

3.4 Example

In this subsection, we explain an example of usage with APIs for Responsive Link.

Fig. 10 shows an example of sending/receiving data packets in `usermain` function. First of all, struct `datalink_packet` sets the header, the payload and the trailer of the data packet. The priority of the header is 0x55, the source address is 0xa5a and the destination address is 0xa5a. Since this is an example of the loopback transmission, the source address is the same as the destination address. The trailer of the data packet sets Full bit to 1 because all payload data are valid. Next we set `new_priority` to 0x11. Then `init_resplink` function is called to initialize the parameters of Responsive Link, as described in Subsection 3.1. Now we call `add_rtable` function to set the routing table. The first call of `add_rtable` function sets the priority, the source address and the destination address of `dp`. The priority exchange function is valid and changes the priority of data packet from 0x55 to 0x11. The output link of the data

```
void usermain(void)
{
    unsigned int cur_priority, src, dst,
               new_priority;
    /* initialize data packet */
    struct datalink_packet dp = {
        .u.header = {
            .high_priority = 0x5,
            .src_addr = 0xa5a,
            .low_priority = 0x5,
            .dst_addr = 0xa5a,
        },
        .payload = {0},
        .trailer = 0x40000000,
    };
    uint8_t buf[0x40];
    new_priority = 0x11;
    /* initialize Responsive Link */
    init_resplink();
    /* add rule to routing table */
    add_rtable(dp.u.addr,
               (0x3 << 14) |
               (new_priority << 6) |
               (0x1 << 5) | 0x1);
    cur_priority = new_priority;
    new_priority = 0xff;
    /* add rule to routing table */
    add_rtable(((cur_priority & 0xf0) << 24) |
               (dp.u.addr & 0xffff0fff) |
               ((cur_priority & 0xf) << 12),
               (0x3 << 14) |
               (new_priority << 6) |
               (0x1 << 5) | 0x0);
    /* send data packet */
    send_datalink(0, &dp);
    /* wait until receiving packet */
    while (recv_datalink(buf) == 0)
        ;
}
```

Fig. 10 Example of sending/receiving data packets.

packet is `L[1]`. The second call of `add_rtable` function sets `new_priority`, the value of which is `0xff`. The source address and the destination address are set to those of `dp`. The priority exchange function is also valid and changes the priority of data packet from `0x11` to `0xff`. The output link of the data packet is `L[0]`. We call `send_dataLink` function to send the data packet with `Mode0`. After sending the data packet, the program calls `recv_dataLink` function repeatedly until receiving the data packet. The return value of `recv_dataLink` function is the length of received data. If receiving the data packet, the busy loop is finished.

Note that the priority of the data packet is finally set to `0xff` so that the header of the data packet is changed from `0x5a5a5a5a` to `0xfa5afa5a`. If the second call of `add_rtable` sets `new_priority`, the value of which is `0x55`, the packet is not transmitted to `L[0]` because the original priority of the data packet is `0x55`. As a result, the header of the data packet is changed to `0x5a5a5a5a` and the data packet is transmitted to `L[1]` repeatedly.

4. Experimental Evaluations

4.1 Experimental Setups

The experimental evaluations use Responsive Multithreaded Processor (RMTP) [11] which has prioritized SMT architecture with MIPS and RMTP-specific instructions. Tab. 1 shows the specification of RMTP. We implement APIs for Responsive Link [6] in RT-Est real-time operating system [7] on RMTP. In order to transform RMTP-specific instructions to machine language, we have developed a cross-compiler for RMTP, which extends gcc version 3.4.3 for MIPS.

Fig. 11 shows our experimental environments. There are two RMTPs, which are connected by Responsive Link. The noise generator covers the cable of Responsive Link. The baud rate of Responsive Link is selected within $\{50, 100, 200, 400, 800\}$ Mbps. In `init_reslink` function, the switch mode is set to

the store and forward mode because the store and forward mode has higher tolerance than the cut through mode. In addition, the transmission mode is set to `Mode0` and the connection method is serial. We use the data link in experimental evaluations.

Tab. 1 Specification of RMTP.

Clock frequency	31.25MHz
SDRAM	64MB
SRAM	256KB
I-Cache/D-Cache	Each 32KB (Harvard)
Fetch width	8
Issue width	4
Integer register	32-bit x 32-entry x 8-set
Integer renaming register	32-bit x 64-entry
FP register	64-bit x 8-entry x 8-set
FP renaming register	64-bit x 64-entry
ALU	4 + 1 (Divider)
FPU	2 + 1 (Divider)
64-bit ALU	1
FP vector unit	1 (4 FPU x 2 line)
Branch unit	2
Memory access unit	1



Fig. 11 Experimental environments.

Tab. 2 Specification of noise wave.

Frequency	12MHz
# of bursts	5
Wave shape	Triangle

We assume the actual noise in noisy environments where a humanoid robot [1] runs. Tab. 2 shows the specification of the noise wave.

In order to evaluate the tolerance of Responsive Link, we perform the voltage tolerance test. First of all,

we set the voltage of the noise wave to 1.0V in the noise generator. Next, we check whether sending/receiving data packets in the noisy connection. If Responsive Link can send/receive data packets, we raise the voltage of the noise wave by 0.1V. If Responsive Link cannot send/receive packets due to noise, the value subtracting 0.1V from the current voltage is that of the voltage tolerance against noise in the environments. We perform this experiment by 10 times in each baud rate and measure the minimum, average and maximum values of the voltage tolerance.

4.2 Overhead

Tab. 3 Overhead of APIs.

function	# of cycles
init_resplink	1,681
add_rtable	91
send_dataLink	27
recv_dataLink	344

Tab. 3 shows the overhead of APIs for Responsive Link. `init_resplink` function sets the various parameters of Responsive Link. As a result, this function has the most cycles in the evaluated functions. `add_rtable` function has low overhead because this function only writes a 32-bit reference part and a 16-bit referent part and sets the identifier of the rule which is the smallest in empty ids. `send_dataLink` function also has low overhead because this function sets the control registers of the data link and the Direct Memory Access Controller. We measure the overhead of `recv_dataLink` function from entering to exiting this function. `recv_dataLink` function checks whether receiving data packets. If receiving data packets, write the data packets to the specified memory. Therefore, `init_resplink` function has higher overhead than `add_rtable` and `send_dataLink` functions.

4.3 Voltage Tolerance Test

Fig. 12 shows the result of the voltage tolerance in the loopback transmission. If the baud rate of

Responsive Link is faster and faster, the voltage tolerance is usually lower and lower.

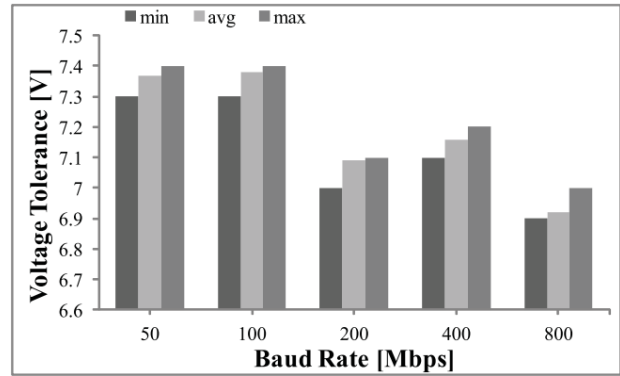


Fig. 12 Loopback transmission.

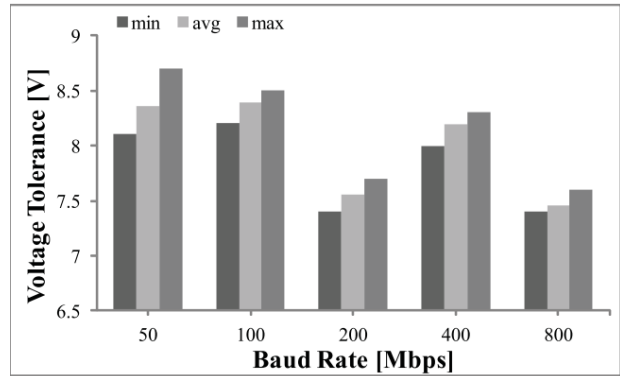


Fig. 13 Point-to-point transmission.

Fig. 13 shows the result of the voltage tolerance in the point-to-point transmission. Each voltage tolerance in the point-to-point transmission has slightly higher voltage tolerance than that in the loopback transmission in Fig. 12. That is to say, the point-to-point transmission improves the tolerance compared to the loopback transmission. Like the loopback transmission, if the baud rate of Responsive Link is faster and faster, the voltage tolerance is also usually lower and lower.

5. Related Work

We compare our work with other work in distributed real-time systems.

The ACE ORB (TAO) [12] is a middleware, which is compliant with Real-Time Common Object Request Broker Architecture (RT-CORBA) [13]. In addition,

TAO implements Internet Inter-ORB Protocol (IIOP) protocol, which is a TCP/IP implementation of General Inter-ORB Protocol (GIOP) protocol specified in CORBA [14-16] by Ethernet. However, it is difficult to avoid packet collision by Ethernet.

Except Ethernet, CAN [3] and FlexRay [4] are also wired communication standards. CAN supports only bus topology and FlexRay supports both bus and star topologies. Distributed control robots [1] require various topologies so that topology-free communication standards such as Responsive Link have more compatibility than CAN and FlexRay. In contrast, Responsive Link supports layer 1 to layer 4 in OSI reference model and has four physical ports to avoid packet collision.

In wireless communication standards, real-time packet scheduling over WirelessHART [2] is proposed [17]. Like Responsive Link, wireless communication standards are also topology-free. In general, wireless communication standards frequently occur packet collision/loss compared to wired communication standards so that the analysis of the worst case arrival time in each packet is difficult. Therefore, Responsive Link is better than wireless communication standards in distributed real-time systems.

6. Conclusion

We presented the APIs for Responsive Link in RT-Est real-time operating system. In particular, we explain how to use the APIs for the data link. Using the APIs, users can build distributed real-time systems easily. In addition, we introduce a method to evaluate the tolerance of communication standards. Experimental evaluations reveal the voltage tolerance against noise in Responsive Link. Using the results, users can select a proper baud rate to communicate between nodes correctly in noisy environments.

In future work, we will implement the APIs for packet scheduling with fixed-priority scheduling [8] and semi-fixed-priority scheduling [9,10] over

Responsive Link. The analysis of the worst case arrival time in each packet is intriguing.

Acknowledgement

This research was supported in part by CREST, JST. This research was also supported in part by Grant in Aid for the Global Center of Excellence Program for "Center for Education and Research of Symbiotic, Safe and Secure System Design" from the Ministry of Education, Culture, Sport, and Technology in Japan. This research was also supported in part by Grant in Aid for the JSPS Fellows.

References

- [1] I. Mizuuchi, Y. Nakanishi, Y. Sodeyama, Y. Namiki, T. Nishino, N. Muramatsu, J. Urata, K. Hongo, T. Yoshikai, and M. Inaba, Advanced Musculoskeletal Humanoid Kojiro, In Proceedings of the 2007 IEEE-RAS International Conference on Humanoid Robots, Pittsburgh, USA (2007) 294-299.
- [2] Available at <http://www.hartcomm.org/>.
- [3] ISO 11898, Road vehicles - Interchange of digital information - Controller area network (CAN) for high-speed communication, International Organization for Standardization, (1993).
- [4] Available at <http://www.flexray.com/>.
- [5] L. Sha, R. Rajkumar, and J. P. Lehoczky, Priority Inheritance Protocols: An Approach to Real-Time Synchronization, IEEE Transactions on Computers, 39 (1990) 1175-1185.
- [6] N. Yamasaki, Responsive Link for Distributed Real-Time Processing, In Proceedings of the 10th International Workshop on Innovative Architecture for Future Generation High- Performance Processors and Systems, Hawaii, USA (2007) 20-29.
- [7] H. Chishiro and N. Yamasaki, RT-Est: Real-Time Operating System for Semi-Fixed-Priority Scheduling Algorithms, In Proceedings of the 2011 International Symposium on Embedded and Pervasive Systems, Melbourne, Australia (2011) 358-365.
- [8] C. Liu and J. Layland, Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, Journal of the ACM, 20 (1973) 46-61.
- [9] H. Chishiro, A. Takeda, K. Funaoka, and N. Yamasaki, Semi-Fixed-Priority Scheduling: New Priority Assignment Policy for Practical Imprecise Computation, In Proceedings of the 16th IEEE International Conference

- on Embedded and Real- Time Computing Systems and Applications, Macau SAR, (2010) 339-348.
- [10] H. Chishiro and N. Yamasaki, Global Semi-fixed-priority Scheduling on Multiprocessors, In Proceedings of the 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Toyama, Japan (2011) 218-223.
 - [11] N. Yamasaki, Responsive Multithreaded Processor for Distributed Real-Time Systems, Journal of Robotics and Mechatronics, 17 (2005) 130-141.
 - [12] D. C. Schmidt, B. Natarajan, A. G. N. Wang, and C. Gill, TAO: A Pattern-Oriented Object Request Broker for Distributed Real-time and Embedded Systems, IEEE Distributed Systems Online, 3 (2002).
 - [13] O. M. Group, Real-time CORBA Specification, Object Management Group, formal/05-01-04 edition (2005).
 - [14] O. M. Group, Common Object Request Broker Architecture (CORBA) Specification, Object Management Group, Version 3.1 Part 1: CORBA Interfaces, formal/08-01-04 edition (2008).
 - [15] O. M. Group, Common Object Request Broker Architecture (CORBA) Specification, Object Management Group, Version 3.1 Part 2: CORBA Interoperability, formal/08-01-07 edition (2008).
 - [16] O. M. Group, Common Object Request Broker Architecture (CORBA) Specification, Object Management Group, Version 3.1 Part 3: CORBA Component Model, formal/08-01-08 edition (2008).
 - [17] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, Real-Time Scheduling for WirelessHART Networks, In Proceedings of the 31st IEEE Real-Time Systems Symposium, San Diego, USA (2010) 150-159.