Tsinghua Science and Technology

Volume 21 | Issue 5 Article 10

2016

Probabilistic Modeling and Optimization of Real-Time Protocol for Multifunction Vehicle Bus

Lifan Su
the School of Software, Tsinghua University, Beijing 100084, China.

Min Zhou
the School of Software, Tsinghua University, Beijing 100084, China.

Hai Wan
the School of Software, Tsinghua University, Beijing 100084, China.

Ming Gu
the School of Software, Tsinghua University, Beijing 100084, China.

Follow this and additional works at: https://tsinghuauniversitypress.researchcommons.org/tsinghua-science-and-technology

Part of the Computer Sciences Commons, and the Electrical and Computer Engineering Commons

Recommended Citation

Lifan Su, Min Zhou, Hai Wan et al. Probabilistic Modeling and Optimization of Real-Time Protocol for Multifunction Vehicle Bus. Tsinghua Science and Technology 2016, 21(5): 561-569.

This Research Article is brought to you for free and open access by Tsinghua University Press: Journals Publishing. It has been accepted for inclusion in Tsinghua Science and Technology by an authorized editor of Tsinghua University Press: Journals Publishing.

TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 10/11 pp561-569
Volume 21, Number 5, October 2016

Probabilistic Modeling and Optimization of Real-Time Protocol for Multifunction Vehicle Bus

Lifan Su, Min Zhou, Hai Wan, and Ming Gu*

Abstract: In this paper, we present the modeling and optimization of a Real-Time Protocol (RTP) used in Train Communication Networks (TCN). In the proposed RTP, message arbitration is represented by a probabilistic model and the number of arbitration checks is minimized by using the probability of device activity. Our optimized protocol is fully compatible with the original standard and can thus be implemented easily. The experimental results demonstrate that the proposed algorithm can reduce the number of checks by about 50%, thus significantly enhancing bandwidth.

Key words: train communication network; probabilistic modelling; protocol optimization

1 Introduction

Train Communication Network (TCN) is a hierarchical combination of two field bus systems for digital train operation. It comprises a Multifunction Vehicle Bus (MVB) inside each coach and a Wire Train Bus (WTB) to connect MVBs with the train control system. Both MVB and WTB have been standardized in IEC 61375[1]. In practice, device status (for instance, status of the brakes and the air-conditioning units) is reported to the Central Control Unit (CCU) of the train, and various devices are controlled by the CCU via MVBs. The MVB network transmits two types of data using Real-Time Protocol (RTP): process data and message data. Process data are small pieces of important data that are required to be transmitted periodically within hard time limits. Message data, by contrast, is usually sporadic information that is transmitted based on demand, and message size may be large. In this study, we consider the optimization of MVB message data transmission.

The physical layer is a wired bus, where each frame is actually visible to all devices[1]. If multiple devices transmit frames simultaneously, the signals interleave with each other and Manchester encoding is broken. Thus, no valid frame can be transmitted. To mitigate this problem, MVBs are coordinated by a bus master. All devices, including the device that serves as the bus master, transmit frames under coordination of the bus master. In general, the bus master sends a master frame, which includes the instructions required to be followed by slave devices. Then, the corresponding slave devices transmit slave frames. For process data, the protocol ensures that there is a unique responding slave device. However, there may be multiple responding slave devices in the arbitration phase of message data transmission.

Message data is transmitted using sliding window protocol[1] which is similar to TCP/IP. The content of a long message is first divided into multiple smaller pieces, and each piece is transmitted within an MVB frame. The bus master first sends a master frame to check whether any device on an MVB wants to send a message data frame. There is no problem if the number of such devices is 0 or 1, i.e., there are no slave frames or only one slave frame, which means no collision occurs. If there are more than one devices, their frames collide with each other (the bus master can detect such a collision), and the master then needs to re-send the master frame by refining its query to a smaller set of slave devices until there is one response at most. This

• Lifan Su, Min Zhou, Hai Wan, and Ming Gu are with the School of Software, Tsinghua University, Beijing 100084, China. E-mail: thssld@gmail.com; zhoumin03@gmail.com; wanhai@tsinghua.edu.cn; guming@tsinghua.edu.cn.
∗ To whom correspondence should be addressed.
Manuscript received: 2015-09-04; accepted: 2015-10-22

process is called arbitration.

On an MVB with a large number of devices, the efficiency of arbitration significantly influences the bandwidth of message data transmission. *Pending devices* are those that want to send message data frames. IEC 61375 suggests a binary tree-like arbitration protocol, which is suitable for a network in which only a small proportion of devices are active. The aforementioned standard arbitration protocol requires $O(\log(n))$ arbitration checks at most to locate a single device, where n is the total number of devices. If there are only two active devices, $1 + \log(n)$ arbitration checks are needed at most. However, if all n devices are active, the number of arbitration checks increases to $2n - 1$. Surprisingly, this outcome is worse than that of the naive round-robin algorithm, which queries all devices one by one, requiring $n + 1$ arbitration checks. As explained above, the probability of device activity influences the optimal arbitration strategy. The standard strategy prescribed in IEC 61375 is far from optimal. In the worst case, it is twice as worse as the optimal strategy.

We propose a scheme for optimizing MVB message data arbitration in this paper and harness a probabilistic model to characterize device activeness. We improve arbitration by skipping certain intermediate nodes on the binary structure of the device tree. Such probability can be obtained from priori knowledge or approximated based on the frequency of activities within a period of time before computation. Compared to the standard-defined reference algorithm, our method can decrease the number of arbitration checks by 50% in the best case. Furthermore, the method preserves compatibility with the standard. The only modification is in the arbitration algorithm of the bus master. Furthermore, our method is effective because it has linear time complexity compared to the number devices.

The remainder of this paper is organized as follows. Section 2 describes the problem and formally introduces the current algorithms. Section 3 illustrates our method. Section 4 describes the performance evaluation of our method. Section 5 presents related work and comparison, and our concluding remarks are given in Section 6.

2 A Motivating Example

The IEC 61375 standard specifies the concept of *group address* to facilitate MVB performance optimization[1]. A group is presented by a pair $\{M, C\}$, where C is the value of the pairs common suffix in binary form and M is the length of the unfixed prefix. In IEC 61375, the pair $\{M, C\}$ is also called *group address*. For example, if we assume that the address space ranges from 0 to 7, all device addresses are represented by 3 binary digits. Devices with addresses 2 (010_2) and 6 (110_2) are in group $\{M = 1, C = 2\ (10_2)\}$ because they share the common suffix 10_2 in their binary representation. They are also in group $\{M = 2, C = 0\}$, along with two other devices 0 (000_2) and 4 (100_2). The standard requires $0 < M <$ digits_of_address, but in practice, a single device with address A can be viewed as a generalized group $\{M = 0, C = A\}$. All devices are in group $\{M = \text{digits\_of\_address}, C = 0\}$, which simplifies further discussion. It is obvious that a group can be divided into two smaller groups. Therefore, all generalized groups form a binary tree, with the root being is the group with all devices and the leaves representing single devices. A non-leaf node of the tree, whose corresponding group is $\{M, C\}$, has two children. Devices in the left child share the common suffix of "0" appended at the left of the common suffix of the current node, so the left child has the group address $\{M - 1, C\}$. The right child, whose devices share the common suffix of a "1" appended at the left, has the group address $\{M - 1, C + 2^{\text{digits\_of\_address}-M}\}$.

During arbitration, the bus master checks the number of pending devices in a generalized group by sending a master frame. Each pending device in the group that has not sent an event will send a slave frame that contains its device address. The response can be classified into three cases: silence is detected, a correct frame is detected, or a collision is detected[1]. They correspond to no, just one, and two or more devices pending in the group, respectively.

An arbitration round starts with a check of all devices, where a device with pending events shall send a response slave frame, and only these devices are allowed to respond until the current round ends. During arbitration, once the bus master gets a correct response, it re-sends the response as a master frame and the corresponding device can send a frame of event data. This procedure is called *event read*. In the rest of the round, this device should not respond to upcoming request frames. Once all devices are checked, the bus master rechecks all devices. Normally, all pending events are sent, so the bus is expected to be silent[1].

A recursive Depth First Search (DFS) algorithm is used for arbitration management. In the case that the bus master acts on a single device, the bus master checks that device. If the device is pending, the bus master obtains a correct response and then reads the event.

In the case that the bus master acts on a group of devices, the bus master checks all devices in the group. If the bus master obtains silence, it does nothing. If the bus master obtains a correct frame, it reads the corresponding event; else, the bus master obtains a collision, which indicates that there are two or more pending devices. The bus master acts on the two subgroups. A round must start with the execution of the DFS algorithm on the group of all groups[1]. An example of a three-bit address system is shown in Fig. 1 and summarized in Table 1.

This conventional method is referenced as the *basic DFS* arbitration algorithm in the rest of this paper.

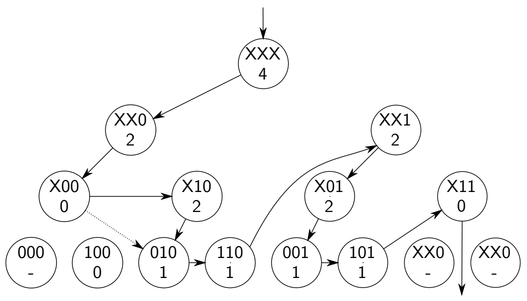


Fig. 1 Procedure of DFS algorithm on binary tree. Each node is labeled with its address and the number of pending devices within the address. We use X as a wildcard for the group address. The solid arrows in the tree denote the sequence of checks performed with basic DFS, while the dashed arrows represent the difference with the reference DFS.

Table 1 Actions and responses using DFS algorithm.

Step	Group being checked	Result	Read event of device address
1	XXX	Collision	—
2	XX0	Collision	—
3	X00	Silence	—
4	X10	Collision	—
5	010	Correct(2)	2
6	110	Correct(6)	6
7	XX1	Collision	—
8	X01	Collision	—
9	001	Correct(1)	1
10	101	Correct(5)	5
11	X11	Silence	—
12	XXX	Silence	—

Notice node X10: when arbitrating its parent group, XX0, two devices 010 and 110 respond; hence, a collision occurs. Then, the bus master checks group X00, and silence is obtained this time. Therefore, the bus master knows that there are at least two devices in group X10. As a result, the bus master skips step 4 in Table 1 without changing the remaining process. Thus, the number of checks during this round is reduced by 1. This idea is used in the standard, which specifies a reference implementation[1] called the *reference DFS* arbitration algorithm. The method that skips the check of a group if the group is known to have at least two pending devices slightly reduces the average number of checks per round.

Furthermore, if the bus master skips checking groups XX1 and X01, the number of checks is further reduced. Skipping certain devices influences the number of arbitration rounds required, and this number may change. However, if we know the probability of each pending device, the bus master can compute the expected number of checks for both possible decisions. In what follows, a probability-based optimization scheme is proposed.

3 Our Proposed Method

We propose a novel probabilistic algorithm that follows the same framework as depth first search. However, when the bus master executes the algorithm on a group, it can determine whether to perform the check for the group or to skip the check and execute the algorithm on the two subgroups. The decision is made by comparing the expected number of checks resulting from the two choices. The expected numbers are computed from the probability of each pending device and the check history. This method employs prior probability to approximate posterior probability. We assume that events such as devices being pending are independent.

3.1 The algorithm

We use the abbreviations specified in Tables 2 and 3 to represent different types of arbitration frames issued by the bus master and the possible responses obtained by it.

Algorithm 1 is the pseudocode of our algorithm. It can be unrolled to a non-recursive style with explicit state in implementation, especially if the bus master is deployed on hardware or is designed as a single-threaded implementation.

In Algorithm 1, in an arbitration round, the bus

Table 2 Request frame types.

Type	Abbr.	Arguments	Explanation	Condition for a device to send a response frame
General	GB	Priority of the round	Start of a round of specified priority	Pending
	GE	None	End of the current round	Still pending
Group	MR	Group address		Belonging to the group and still pending
Single	SR	Device address		Address matching and still pending

Table 3 Response of an arbitration check.

Type	Arguments	Condition	Implied number of pending devices in the corresponding group of request
SILENCE	None	If no device response	0
CORRECT	Device address of responsing device	If single device responses	1
COLLISION	None	Other cases	$\geqslant 2$

master first loads the probability of an event such that each device is pending by using the function *LoadProbability* on line 2. The *Preparation* function computes the decisions for all combinations of groups and the known minimal numbers, and stores the decisions in global data skipDecision in line 3. The global data *skipDecision* function accepts a group address and the known least pending devices in the group, and returns the decision about whether the check for the group should be skipped. The bus master checks the group of all devices to start a new round in line 4. If a collision occurs, the function *PdfsGroup* is called. This function reads all pending events in a group. It first checks whether the current check should be skipped according to *skipDecision* in line 17. If this is the case, it recursively checks the two subgroups. Else, it executes as a DFS algorithm. The known minimal number of pending devices is always 0 for the left subgroups because of the lack of adequate information. For the right subgroups, this number is computed by subtracting the actual number of pending devices in the left subgroup from the known minimal number for the current node group.

3.2 Probabilistic modeling

The essence of the algorithm is to compute *skipDecision*. The following notations are used. Let G be a string of "0" and "1" characters. Group G contains all devices that share a common suffix G. In this notation, group $0G$ and $1G$ are the two subgroups of group G. $\hat{G}$ is the binary code of G. p_i is the probability of an event such that the device at address i is pending to send in the current round. Let $x \oplus y = \min(x+y, 2)$, and $x \ominus y = \max(x-y, 0)$. $P_{G,i}$ is the probability of an event such that the number of pending devices in group G is i if i is 0 or 1, but equal to or greater than i if i is 2. Let A_G be a random variable of the number of arbitration checks. To send all events inside group G, provided that the check for group G is skipped, and the rest is performed according to our algorithm. Let U_G be the expected number of A_G. Let s_G be the known least number of pending devices. Let t_G be the minimum number of 2 and the actual number of pending devices. $E^1_{G,i,j} = U_G|(s_G = i, t_G = j)$, $E^1_{G,i} = U_G|(s_G = i)$, where $0 \leqslant i \leqslant j \leqslant 2$. B_G and V_G are defined in the same way as A_G and U_G, respectively, except that the check for group G is performed. $E^2_{G,i,j} = V_G|(s_G = i, t_G = j)$, $E^2_{G,i} = V_G|(s_G = i)$. C_G is defined as A_G and B_G, and W_G is defined in the same way as U_G and V_G, except that the check of group G is performed according to our algorithm. $E_{G,i,j} = W_G|(s_G = i, t_G = j)$, $E_{G,i} = W_G|(s_G = i)$.

We consider the case that a general group consists of only one device. Each probability can be derived directly from the probability of the event that the corresponding device is pending. The probability of an event such that one device is pending in group G is the probability that device $\hat{G}$ is pending, and it is impossible to have two or more devices pending in group G. Moreover, the expected number of arbitration checks is always 1 because group G should always be tested.

$$\begin{cases} P_{G,0} = 1 - p_{\hat{G}}; \\ P_{G,1} = p_{\hat{G}}; \\ P_{G,2} = 0; \\ E_{G,i,j} = 1, \quad 0 \leqslant i \leqslant j \leqslant 2; \\ E_{G,i} = 1, \quad 0 \leqslant i \leqslant 2 \end{cases} \tag{1}$$

Consider a group that contains a greater number of devices. We first calculate the probabilities of the group based on the probabilities of its internal subgroups.

Algorithm 1 Optimized Algorithm, Pdfs

```
Function PdfsRound is
    Data: probability
    Data: skipDecision
    LoadProbability()
    Preparation()
    busState←send(GB)
    if busState=COLLISION then
        minimal←2
        leftActual←PdfsGroup(LeftChild(groupAll), 0)
        rightMin← max(minimal−leftActual, 0)
        rightActual←PdfsGroup(RightChild(groupAll),
        rightMin)
        busState←send(GE)
    else if busState = CORRECT(addr) then
        eventRead(addr)
        busState←send(GE)
    end
end
Function pdfsGroup is
    Data: skipDecision
    Input: address: address, group or single
    Input: minimal: known minimal number of pending
            devices in the group
    Output: actual number of pending devices in the group
    if not skipDecision(address, minimal) then
        if address is group then
            busState←send(MR(address))
        else
            busState←send(SR(address))
        end
        if busState = SILENCE then
            return 0
        else if busState=CORRECT(addr) then
            eventRead(addr)
            return 1
        end
    end
    leftActual←PdfsGroup(leftChild(address), 0)
    rightMin← max(minimal−leftActual, 0)
    rightActual← PdfsGroup(rightChild(address),
    rightMin)
    return leftActual + rightActual
end
```

Given that the events in which each device is pending are independent, the numbers of pending devices in each subgroup are independent as well. The probability of an event in which there is no pending device in group G is the product of the probabilities of the events in which there is no pending device in each subgroup.

$$\begin{cases} P_{G,0} = P_{0G,0}P_{1G,0}; \\ P_{G,1} = P_{0G,0}P_{1G,1} + P_{0G,1}P_{1G,0}; \\ P_{G,2} = 1 - P_{G,0} - P_{G,1} \end{cases} \tag{2}$$

When the arbitration check is omitted, the bus master simply executes the process on the two subgroups. Thus, $A_G = C_{0G} + C_{1G}$. We can calculate the expected number of checks as follows:

$$E^1_{G,i,j} = E[A_G | s_G = i, t_G = j] \tag{3}$$

By applying the law of total expectation, we get Eq. (4).

$$\begin{aligned} E_{G,i,j} = &\sum_{x=0}^{2} P(t_{0G} = x | s_G = i, t_G = j) \cdot \\ &E[A_G | s_G = i, t_G = j, t_{0G} = x] = \\ &\sum_{x=0}^{2} P(t_{0G} = x | s_G = i, t_G = j) \cdot \\ &\left(\sum_{y=0}^{2} P(t_{1G} = y | s_G = i, t_G = j, t_{0G} = x) \cdot \right. \\ &\left. E[A_G | s_G = i, t_G = j, t_{0G} = x, t_{1G} = y] \right) = \\ &\sum_{x=0}^{2} \sum_{y=0}^{2} P(t_{0G} = x | s_G = i, t_G = j) \cdot \\ &P(t_{1G} = y | s_G = i, t_G = j, t_{0G} = x) \cdot \\ &E[A_G | s_G = i, t_G = j, t_{0G} = x, t_{1G} = y] \end{aligned} \tag{4}$$

By the definition of conditional probability, $E_{G,i,j}$ can be further rewritten as Eq. (5).

$$\begin{aligned} E_{G,i,j} = &\sum_{x=0}^{2} \sum_{y=0}^{2} P(t_{0G} = x, t_{1G} = y | s_G = i, t_G = j) \cdot \\ &E[A_G | s_G = i, t_G = j, t_{0G} = x, t_{1G} = y] \end{aligned} \tag{5}$$

To compute $E_{G,i,j}$, it is sufficient to compute both $P(t_{0G} = x, t_{1G} = y | s_G = i, t_G = j)$ and $E[A_G | s_G = i, t_G = j, t_{0G} = x, t_{1G} = y]$. We assume that the S_G is irrelevant of $P(t_{0G} = x, t_{1G} = y | s_G = i, t_G = j)$. This assumption reduces computational complexity without significantly affecting the performance of our algorithm.

$$\begin{aligned} &P(t_{0G} = x, t_{1G} = y | s_G = i, t_G = j) \approx \\ &\quad P(t_{0G} = x, t_{1G} = y | t_G = j) \end{aligned} \tag{6}$$

According to the definition of t_G, $t_G = t_{0G} \oplus t_{1G}$, or $j = x \oplus y$. If a combination of x and y does not meet this requirement, the combination is impossible; thus,

$$P(t_{0G} = x, t_{1G} = y | t_G = j) = 0 \tag{7}$$

By contrast, if x and y satisfy $x \oplus y = j$, consider the following two facts: $t_G = j$ is implied from $t_{0G} = x$ and $t_{1G} = y$, while the random variables t_{0G} and t_{1G} are independent because the pending events are independent.

$$\begin{aligned} &P(t_{0G} = x, t_{1G} = y | t_G = j) = \\ &\frac{P(t_{0G} = x, t_{1G} = y, t_G = j)}{P(t_G = j)} = \\ &\frac{P(t_{0G} = x, t_{1G} = y)}{P(t_G = j)} = \frac{P_{0G,x} P_{1G,y}}{P_{G,j}} \end{aligned} \tag{8}$$

$E[A_G | s_G = i, t_G = j, t_{0G} = x, t_{1G} = y]$ can be computed as follows:

$$\begin{aligned} &E[A_G | s_G = i, t_G = j, t_{0G} = x, t_{1G} = y] = \\ &E[C_{0G} + C_{1G} | s_G = i, t_G = j, t_{0G} = x, t_{1G} = y] = \\ &E[C_{0G} | s_G = i, t_G = j, t_{0G} = x, t_{1G} = y] + \\ &E[C_{1G} | s_G = i, t_G = j, t_{0G} = x, t_{1G} = y] \end{aligned} \tag{9}$$

First, consider the first term $E[C_{0G} | s_G = i, t_G = j, t_{0G} = x, t_{1G} = y]$. The condition $t_{0G} = x$ is introduced because $s_{0G} = 0$ holds for any group $0G$ according to our algorithm. The condition $t_G = j$ can be removed because it is implied by $t_{0G} = x$ and $t_{1G} = y$. The condition $t_{1G} = y$ can be removed because the events in which each device is pending are independent. We assume that C_{0G} is not related to s_G, for the same reason above.

$$\begin{aligned} &E[C_{0G} | s_G = i, t_G = j, t_{0G} = x, t_{1G} = y] = \\ &E[C_{0G} | s_G = i, t_{0G} = x, s_{0G} = 0] \approx \\ &E[C_{0G} | t_{0G} = x, s_{0G} = 0] = E_{0G,0,x} \end{aligned} \tag{10}$$

Then, we calculate the other term $E[C_{1G} | s_G = i, t_G = j, t_{0G} = x, t_{1G} = y]$. Similarly, we can obtain $E[C_{1G} | s_G = i, t_G = j, t_{0G} = x, t_{1G} = y]$ as follows. Note that the bus master knows that there are at least $i \ominus x$ pending devices in $1G$.

$$\begin{aligned} &E[C_{1G} | s_G = i, t_G = j, t_{0G} = x, t_{1G} = y] = \\ &E[C_{0G} | s_G = i, t_{1G} = y, s_{1G} = i \ominus x] \approx \\ &E[C_{0G} | t_{1G} = y, s_{1G} = i \ominus x] = E_{1G,0,x} \end{aligned} \tag{11}$$

Based on the above discussion, we can obtain

$$\begin{aligned} P_{G,j} E^1_{G,i,j} = \sum_{x \oplus y = j} & P_{1G,y} \left(P_{0G,x} E_{0G,0,x}\right) + \\ & P_{0G,x} \left(P_{1G,y} E_{1G,i \ominus x,y}\right) \end{aligned} \tag{12}$$

If the response to an arbitration check is silence or correct, one arbitration check is required. Else, the bus master knows that there are at least two pending devices and the algorithm is executed on both the subgroups. Therefore $E^2_{G,i,2} = E^1_{G,2,2} + 1$, where 1 denotes the check for this group. Thus, $P_{G,k} E^2_{G,i,k}$ can be obtained as Eq. (13).

$$\begin{cases} P_{G,0} E^2_{G,i,0} = P_{G,0}; \\ P_{G,1} E^2_{G,i,1} = P_{G,1}; \\ P_{G,2} E^2_{G,i,2} = P_{G,2} + \\ \quad \sum\limits_{x \oplus y = 2} \left(P_{1G,y} \left(P_{0G,x} E_{0G,0,x}\right) + \right. \\ \quad \left. P_{0G,x} \left(P_{1G,y} E_{1G,2 \ominus x,y}\right)\right) \end{cases} \tag{13}$$

The probability of an event such that $s_G = i$ and $t_G = j$ if $s_G = i$ is

$$\frac{P_{G,j}}{\sum\limits_{k=i}^{2} P_{G,k}} \tag{14}$$

The expected number of arbitration checks on group G, where the minimal number of pending devices is known, is calculated as follows:

$$\left(\sum_{j=i}^{2} P_{G,j}\right) E^c_{G,i} = \sum_{j=i}^{2} \left(P_{G,j} E^c_{G,i,j}\right), \quad c \in \{1,2\}, i \in \{0,1,2\} \tag{15}$$

After all the above values are computed, the decision on skipping the check of the group is formulated, as Eq. (16).

$$\text{skipDecision}(G, i) = \begin{cases} \text{true (to skip)}, & \text{if } \left(\sum\limits_{j=i}^{2} P_{G,i}\right) E^1_{G,i} < \left(\sum\limits_{j=i}^{2} P_{G,i}\right) E^2_{G,i}; \\ \text{false (not to skip)}, & \text{otherwise} \end{cases} \tag{16}$$

The values of $E_{G,i,j}$ and $E_{G,i}$ are obtained from $E^{1,2}_{G,i,j}$ and $E^{1,2}_{G,i}$, respectively. This computation process has a constant complexity for a single node. The processes of groups having common suffixes of the same length are independent. The node tree is a perfect binary tree, so the number of nodes in the tree is exactly twice the size of the address space minus 1, and the number of levels of the tree is 1 plus the binary logarithm of the address space size. Thus, the *Preparation* function has linear complexity in the address space and logarithmic complexity in a parallel implementation.

4 Evaluation

4.1 Experiment design

To evaluate the performance of our algorithms, we conducted a simulation experiment and computed the number of arbitration checks in a single round. Three algorithms were selected for comparison with our algorithm: round-robin polling algorithm, basic DFS algorithm, and reference DFS algorithm. For a fair comparison, we assumed that all devices are class X devices that can transmit message data.

In the experiment, each device was associated with a given probability of pending. To remove uncertainty, 10 000 rounds were simulated for each input probability vector of the devices. The total number of checks within a single arbitration round was recorded.

The probability of events such that devices are pending was assumed to follow a logit normal distribution. This distribution maps a variable following normal distribution into $(0, 1)$ by the inverse of a sigmoid function. Different values are generated by changing the expected value and standard deviation of the distribution.

4.2 Result and analysis

Figure 2 shows that the proposed algorithm outperforms all three existing algorithms in terms of the average number of checks. Figure 3 shows that in 95% of the rounds in this workload, our algorithm required fewer arbitration checks than the average number of checks required by the standard algorithm.

In the polling algorithm, if there are many pending devices, the first arbitration check will cause a collision. Thus, all devices need to be checked. Because it is assumed that all devices can send messages, an almost straight line can be observed at 257 checks on the chart. This is the number of device plus 2 (indicating the start and the end of a round). When the number of pending devices increases, the performance of the polling algorithm remains the same, while the average number of checks required by our algorithm increases, although it still outperforms the polling algorithm. Overall, our algorithm has the most improvement with the workload of few (more than 2) devices pending, as shown in Fig. 2a.

Figures 2b and 2c show the existence of similar relationships among the two existing DFS algorithms and our algorithm. This indicates that the improvement of our algorithm over the reference DFS algorithm is not huge. This is because the condition for skipping the check of a group is hard to satisfy in the reference DFS algorithm. If a check on a group is to be skipped, it should contain at least two pending devices. More specifically, such a group needs to represent a right subgroup and contain at least two pending devices, and the sibling of this group should contain no pending device. In our algorithm, if this condition is met, the check for that group is skipped as well. In other cases, the probability that this group has no or one pending device is $P_{G,0} + P_{G,1}$. It is smaller than $(P_{0G,0} + P_{0G,1})(P_{0G,0} + P_{0G,1})$. For low workload, because the probabilities that each device is pending p_i are low, most groups are very likely to have no or one device pending, and only the checks for a few large groups are skipped. Thus, the improvement is not obvious. When workload is large, smaller groups are more likely to contain at least two pending devices, so a greater number of checks are skipped. As the result, our algorithm shows significant improvement in high workload cases.

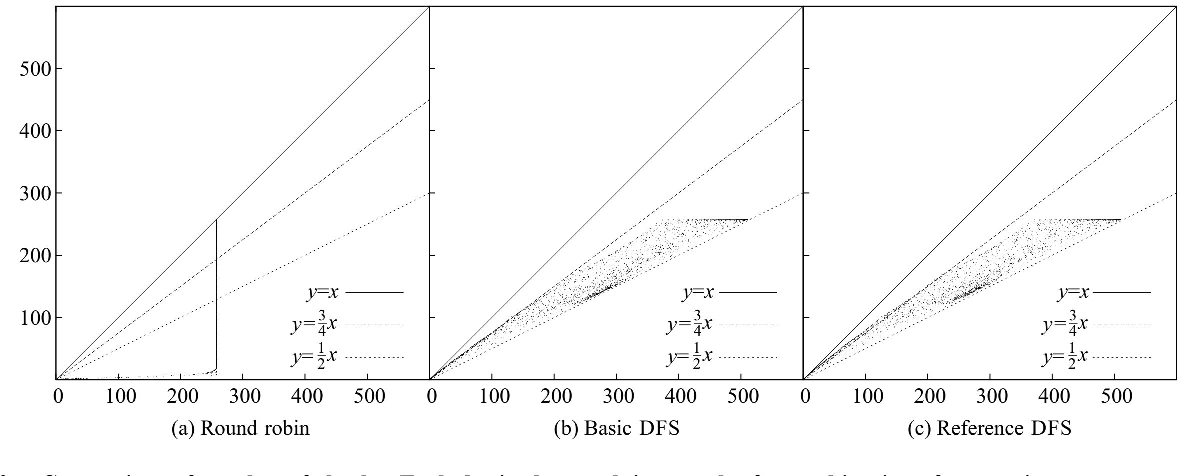


Fig. 2 Comparison of number of checks. Each dot in the graph is a result of a combination of p_i. y-axis: average number of checks performed in a round using our algorithm. x-axis: average number of checks performed in a round using (a) round-robin, (b) basic DFS, and (c) reference DFS algorithm.

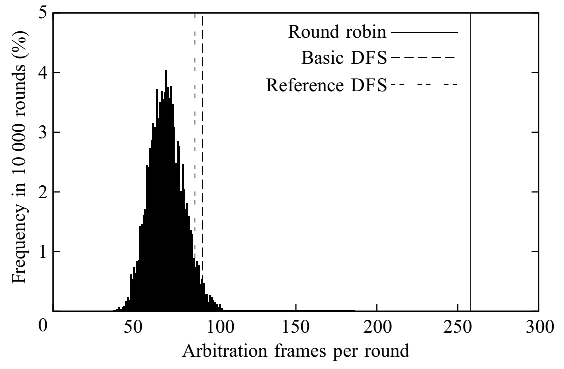


Fig. 3 Distribution of number of arbitration checks. The histogram shows the frequency of number of checks for 10 000 scenarios for moderate workload. The scenarios are generated randomly. For the *i*-th device, it is ensured that the proportion of scenarios, where the *i*-th device is active, is p_i.

5 Related Work

There are some related works on the performance analysis and optimization of MVB.

Liu et al.[2] analyzed the performance of a MVB network, including expected polling period, expected latency, and other properties. Zhu et al.[3] discovered the relationship between network efficiency and the number of devices and frame length by using simulation.

Zeng et al.[4] compared two strategies of bandwidth allocation for different aperiodic data. One follows the standard, allowing for best-effort transfer ability of high-priority events, and the other strategy offers static bandwidth. The former has low latency for high-priority data and higher latency for low-priority events, while the latter improves the performance of low-priority data under heavy loads.

Nie et al.[5] described a method involving the reallocation of device addresses and using the reserved priority levels from 2 to 16. Their algorithm also checks the devices that have sent an event frame in the prior round. They claimed that their algorithm reduces latency for high-priority data. Chen et al.[6] introduced an Particle Swarm Optimization–based scheme by following the former two approaches to reduce the number of collisions.

Zhu[7] and Zhu and Guo[8] proposed a WTB-like system, in which a slave sends process data with an additional frame to inform the bus master whether a device is pending. The bus master performs arbitration accordingly. Thus, collisions can be avoided among devices with process data abilities.

The two latter methods aim to reduce latency for high-priority data, which is assumed to be short in length and is used in emergencies. Latency is closely related to the time used to lookup the next pending device. If a few checks are skipped, latency is expected to decrease. Thus, this effect can be achieved in our algorithm. However, these methods require modification of existing devices, which limits their application. Our algorithm preserves compatibility with the standard IEC 61375, so existing devices need not be modified. Only the bus masters need to adopt the poll algorithm.

Wang et al.[9] proposed an arbitration algorithm that employs dynamic priority allocation to reduce network latency and avoid starvation of low-priority communication requests.

Jimenez et al.[10] presented an approach to simulate the behavior of MVBs. Their work includes the modeling of message data transmission. There are some earlier discussions to optimized aperiodic data transmitted in other industrial buses.

Cavalieri et al.[11] described a polling protocol based on Fieldbus. The protocol enables a device to request data that resides on another device. One of their methods uses a bit from period data frames to indicate that the sending device is requesting. The other method uses a standalone frame to check whether a device is requesting. Cavalieri et al.[12] also introduced a priority-based mechanism to increase larger bandwidth and lower the delay of critical data.

Stefano et al.[13] introduced the communication protocol BRAIN for process control systems. The protocol defines a distributed network. The devices send periodic data based on a common schedule. When a device sends a frame, if the device is pending to send aperiodic data, it reserves a time slot in the unused period by writing to specified fields in the frame. Thus, this reservation is broadcast to all devices. The device later sends aperiodic data in the reserved slot. This mechanism is similar to WTB, but it permits an aperiodic data frame to reserve another time slot. Thus, they attempted to maintain fairness in the method, whereas WTB master uses the round-robin method[1] to poll all pending devices.

6 Conclusion

In this paper, we explored the optimization of the arbitration process of MVBs. A probabilistic model of arbitration was introduced. Based on the novel probabilistic characterization, we proposed a new approach to handle arbitration effectively. The experimental results demonstrate the effectiveness of our approach for both high and low workloads.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Nos. U1201251 and 61402248), the National Key Technologies Research and Development Program of China (No. 2015BAG14B01-02), and MIIT IT funds (Research and application of TCN key technologies) of China.

References

[1] International Electrotechnical Committee, Train communicate network, IEC 61375-1, Geneva, Switzerland, 2007.

[2] J. W. Liu, N. L. Tan, F. C. Jiao, R. Xing, and X. M. Gao, Performance analysis of multifunction vehicle bus system, *Journal of the China Railway Society*, vol. 6, p. 017, 2006.

[3] J. Zhu, F. Li, L. Wang, and Y. Li, Study on network dynamic performance of multifunction vehicle bus based on simulation model, in *Transportation Electrification Asia-Pacific (ITEC Asia-Pacific), 2014 IEEE Conference and Expo*, 2014, pp. 1–5.

[4] Q. Zeng, T. Chen, and Y. Liu, Study on bandwidth allocation strategy of MVB aperiodic communication, (in Chinese), *Application of Electronic Technique*, vol. 37, no. 4, pp. 106–108, 2011.

[5] X. Nie, L. Wang, and P. Shen, Real-time performance and optimization of MVB network of rail vehicle, (in Chinese), *Journal of the China Railway Society*, vol. 33, pp. 40–44, 2011.

[6] J. Chen, Y. He, and W. Wei, The optimization of aperiodic message transmission on MVB based on particle swarm optimization, in *Sixth International Conference on Advanced Computational Intelligence*, 2013.

[7] Q. Zhu, Study on theory and methods for real-time of the train communication network, (in Chinese), PhD dissertation, Department of Traffic and Transportation Engineering, Tongji University, Shanghai, China, 2008.

[8] C. Zhu and Q. Guo, Research on multi-optimizing strategy for MVB sporadic data of rail transit on-board network, *Applied Mechanics and Materials*, vols. 543–547, pp. 1987–1994, 2014.

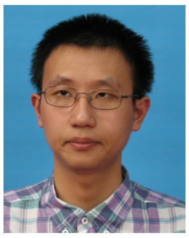
[9] H. Wang, J. Xu, and H. Hu, Multifunction vehicle bus arbitration real-time scheduling algorithm, (in Chinese), *Journal of Jilin University*, vol. 5, p. 027, 2015.

[10] J. Jimenez, I. Hoyos, C. Cuadrado, J. Andreu, and A. Zuloaga, Simulation of message data in a testbench for the multifunction vehicle bus, in *IEEE Industrial Electronics, IECON 2006-32nd Annual Conference on*, 2006, pp. 4666–4671.

[11] S. Cavalieri, A. D. Stefano, and O. Mirabella, Acyclic traffic management in fieldbus scenarios, in *Emerging Technologies and Factory Automation, 1992. IEEE International Workshop on*, 1992, pp. 342–347.

[12] S. Cavalieri, A. D. Stefano, and O. Mirabella, Optimization of acyclic bandwidth allocation exploiting the priority mechanism in the fieldbus data link layer, *Industrial Electronics*, vol. 40, pp. 297–306, 1993.

[13] A. D. Stefano, L. L. Bello, and O. Mirabella, BRAIN: A new distributed communication protocol for synchronous process control applications, in *Emerging Technologies and Factory Automation, 1996. EFTA'96. Proceedings., 1996 IEEE Conference on*, 1996, vol. 2, pp. 616–622.



Lifan Su received the BS degree from Tsinghua University, China in 2012. He is currently working toward the PhD degree in software engineering at Tsinghua University. His research interests include scheduling in embedded and realtime systems.

Ming Gu is currently a professor of the School of Software, Tsinghua University, and vice director of Ministry of Education Key Laboratory of Information Security. She received the MS degree from Chinese Academy of Science in 1986. Her research areas include software formal methods, software trustworthiness, and middleware technology.

Min Zhou is a research assist at the School of Software, Tsinghua University. He received the BS degree in 2007 and PhD degree in 2014, both from Tsinghua University, China. His research interests include model checking, and program analysis and testing.

Hai Wan received the PhD degree from Tsinghua University, China, in 2011. He is currently a research assistant in the School of Software at Tsinghua University, China. His research interests include embedded systems, real time systems, and formal modeling and verification.