A Software-Based Error Detection Technique for Monitoring the Program Execution of RTUs in SCADA

Navid Rajabpour and Yasser Sedaghat(✉)

Dependable Distributed Embedded Systems (DDEmS) Laboratory,
Computer Engineering Department,
Ferdowsi University of Mashhad, Mashhad, Iran
n_rajabpour@stu.um.ac.ir, y_sedaghat@um.ac.ir

Abstract. A Supervisory Control and Data Acquisition (SCADA) system is an Industrial Control System (ICS) which controls large scale industrial processes including several sites over long distances and consists of some Remote Terminal Units (RTUs) and a Master Terminal Unit (MTU). RTUs collect data from sensors and control actuators situated at remote sites and send data to the MTU through a network. Since RTUs operate in a harsh industrial environment, fault tolerance is a key requirement particularly for safety-critical industrial processes. Studies show that a significant number of transient faults due to a harsh environment result in control flow errors in the RTU's processors. A software error detection technique has been proposed to detect control flow errors in several RTUs. For experimental evaluation 30,000 faults injected on network; the average performance and memory overheads are about 33.20 % and 36.79 %, respectively and this technique detected more than 96.32 % of injected faults.

Keywords: SCADA · RTU · Transient fault · Fault tolerance · Fault injection · Software-based error detection

1 Introduction

The most critical infrastructures, such as major electrical and mechanical system and industrial networks are controlled by industrial control systems (ICSs). These systems are typically employed to monitor and control the plants and industrial environments such as oil and natural gas pipeline, water distribution, electrical power grids and transportation [1]. In these applications, which are commonly safety-critical, a system failure may lead to significant risks to the health and the safety of human's life, serious damages to the environment, or serious financial and economical issues [2].

A Supervisory Control and Data Acquisition (SCADA) system is an ICS which controls large scale industrial processes included several sites over long distances [3]. A SCADA system is a distributed system typically comprised of a Master Terminal Unit (MTU) and several Remote Terminal Units (RTUs). MTU gathers data from RTUs, provides an operator interface to display information, and controls the remote sites. RTU interfaces with field sensing devices, local control switchboxes, and valve actuators. To transfer data between MTU and RTUs, a communication network based

F. Koornneef and C. van Gulijk (Eds.): SAFECOMP 2015, LNCS 9337, pp. 457–470, 2015.
DOI: 10.1007/978-3-319-24255-2_33

on the client/server over the communication protocols, such as TCP/IP or Ethernet/IP protocols, is commonly used [4, 23].

RTUs operate in an industrial environment, which is commonly a harsh environment [5]. In a harsh environment, transient faults may occur in electronic devices (i.e. microprocessors and microcontrollers, memory, internal bus) due to Electromagnetic Interferences (EMI), Power Supply Disturbances (PSD), radiations and high operating temperature [19] in RTUs. This fault resulting in an inversion of a bit state [24] (i.e. single bit flip). Presented studies in [6] show that transient faults in the electronic devices can cause control flow errors and data errors. Control flow errors refer to deviations from the normal instruction execution flow of the software program and data errors refer to alter the contents of memory variables or in a register. It has been shown that about 33 %–77 % of transient faults are converted to control flow errors [6] and the remaining are converted to data error. Monitoring the execution of program in RTU by MTU is very important and a CFE error would prevent the program from performing properly. Therefore, RTUs should be equipped with a technique in order to be able to detect such errors, and Control Flow Checking (CFC) is one of the best techniques to detect the occurrence of control flow errors.

Several CFC techniques have been presented since 1980s [6–17] that can be divided into three categories of hardware-based, software-based, and hybrid. Hardware-based techniques use an extra hardware such as a watchdog processor to monitor state performance and state of the master processor [7]. Software-based techniques employ software redundancy to detect deviations in software program execution flow by signature monitoring mechanisms [8, 9]. These techniques have software code and performance overheads. In comparison with hardware-based techniques, software-based techniques are more flexible, less costly and being easily updated and they have also a better maintenance facility [6]. In hybrid techniques, a software technique would be merged with a hardware-based technique, in order to make balance between overheads and costs and have advantages of both software-based and hardware-based techniques.

In this study, a software control flow checking technique, called PLC-CFC, is proposed in order to detect control flow errors in RTUs. This technique is employed to monitor the program execution flows of some RTUs in a SCADA system and composes of two parts. A software-based control flow checking technique has been embedded in each RTU and the MTU processor, along with doing its tasks, is employed to monitor the program execution flow of all RTUs. The ICS-CFC technique can be applied in all industrial control systems which employ microcontrollers, microprocessors, PLCs, or personal computers. As a case study, the proposed technique has been applied on a real ICS in Parin Beton Amood Company, manufacturing "Autoclaved Aerated Concrete" in Mashhad, Iran. The mentioned ICS includes eight personal computers as its RTUs and a main server as its MTU. The experimental results showed that among a total of 30,000 injected faults on distributed network and the presented technique detects more than 96.32 % of them.

The organization of this paper is as follows. Section 2, several related studies are reviewed. In Sect. 3 the proposed technique explained. The experimental results of different technique are given in Sect. 4. Finally Sect. 5 concluded the paper.

2 Related Works

Control flow checking techniques are typically employed signature monitoring mechanisms to check the execution flow of a program. Before the system's run-time, an abstract of the program for the correct program execution is extracted. Afterward, several signatures, representing the chosen abstract, are assigned to the program. During the run-time, signatures are again generated in real-time and are compared with the stored, expected signatures. If a disagreement occurs, the occurrence of a control flow error is detected and reported [10].

As mentioned before, control flow checking techniques can be divided into three main types. In software-based techniques such as CFCSS [14, 15], ECCA [13], RSCFC [16], I2BCFC [25], and SCFC [17], several redundant instructions are inserted into a program to check the execution flow of the program. These techniques do not have any hardware overhead. The main advantages of software-based techniques are low cost and flexibility (easily changeable). However, these techniques impose significant performance and memory overheads due to the redundant instructions. Moreover, since the monitoring is done by inserted instructions into the program, these techniques cannot detect the program crash failure.

In hardware-based techniques, to detect control flow errors a hardware device like Watchdog timer [11] and Lock stepping [12], is utilized to check the execution flow of a program or to trace memory accesses. In these techniques, the behavior of the main processor is monitored using only redundant hardware devices. Therefore, these techniques, commonly designed for a special purpose, cannot be easily changed or updated, and have considerable costs due to redundant hardware. In the hybrid control flow checking techniques, redundant instructions are inserted into a program. These instructions produce some signatures and send them to a redundant hardware device as an external monitor. In these techniques, the control flow checking is done partly in the program (employing software-based techniques) and partly in the redundant hardware. CFCBTE [18], SWTES [10], PECFC [24], are the samples of hybrid techniques. In the SWTES technique, a hybrid-base technique using encoded signatures monitors the behavior of a program and an on-chip microcontroller timer has been exploited as a watchdog timer to detect the program crashes. This technique is experimentally evaluated on an ATMEL MCS51 microcontroller. The CFCBTE is a hybrid control flow checking technique for the PowerPC processors. In this technique, beside redundant software codes which are employed to compare signatures, three hardware-based mechanisms, i.e., Machine Check Exception, watchdog timer, and Branch Trace Exception, have been utilized.

To the best of our knowledge, almost all control flow checking techniques have been proposed to protect a single processor from control flow errors; while, SCADA is a distributed system composed of several RTUs and a MTU connected to a communication network. RTUs and MTU include a microcontroller, a microprocessor, a PLC, or a personal computer as their main processor. To protect SCADA from control flow errors it is necessary to employ a control flow checking technique. In this paper a software control flow checking is proposed for industrial control systems.

3 The Proposed Technique: ICS-CFC

As mentioned before, SCADA system is a distributed industrial control system composed of several RTUs and a server as a MTU. RTUs interface with field sensors and local control devices provide an operator interface to control remote sites. These units execute their configuration and software control programs and send their control and sensing data to MTU through a client/server communication network. The MTU gathers data from RTUs to process and control the remote sites [5].

In the proposed control flow checking technique, called ICS-CFC technique, monitoring of the program execution is being done partly on the MTU and partly on each RTU. Employing this technique, the MTU ensures that the execution flow of the programs in all RTUs is correct, by online monitoring the received signatures from RTUs through the communication network. Local signature monitoring mechanism has been implemented in RTUs to reduce the number of transmitted signatures through the network.

3.1 ICS-CFC Technique Implementation in RTUs

To implement the local checking the control flow of an RTU's program, three steps should be taken as follows: Step 1: Partitioning the program into several basic blocks, Step 2: Assigning a unique signature to each basic block, and Step 3: Inserting control flow checking instructions to the program.

Partitioning the Program Code. The RTU's program code is first divided into several basic blocks. The basic block is a maximal set of ordered instructions such that its execution begins from the first instruction and terminates with the last instruction. There is no jumping or branching instruction in a basic block except for the last one [14]. A program can be represented by a directed graph, called Control Flow Graph (CFG) [20]; nodes are the basic blocks and the arcs represent a control flow transfer between the basic blocks. A simple program and its related CFG are shown in Fig. 1.

Assigning a Unique Signature. After indicating the basic blocks and extracting the CFG of the program, a unique signature should be assigned to each basic block, called Detection Signature (DS). The DS of each basic block demonstrates the successor blocks of the current basic block. Bits related to successor blocks of the present block equal 1. As shown in Fig. 2(a), the DS contains two fields. The first field (T) represents the type of the signature. The value of the T field for a Detection Signature is 1. If the number of basic blocks is N, then the second field of the DS (DS_{BN}), which represents the signature's value, have N bits. This field contains the assigned signature of the related basic block. If there are many basic blocks, then the number of signature bits can be equaled Log_2 N. For example, if N = 512, then the number of bits can be 9 lengths.

In addition to the Detection Signature (DS), two other types of signature are employed in the ICS-CFC technique: Alive Signatures and Error Signatures. An Alive Signature (AS) would be sent from RTUs to the MTU in certain periods of time. These periods are determined by the MTU in the configuration time of RTUs. Receiving an

```
// basic block 1
int n;
int i, temp;
while (n > 1){
// basic block2
i = 0;
while (i < n - 1){
  if (x[i] > x[i + 1]){
// basic block3
 temp = x[i]; x[i] = x[i + 1];
  x[i + 1] = temp;
  }
// basic block4
 i++;
 }
// basic block5
n--;
}
```

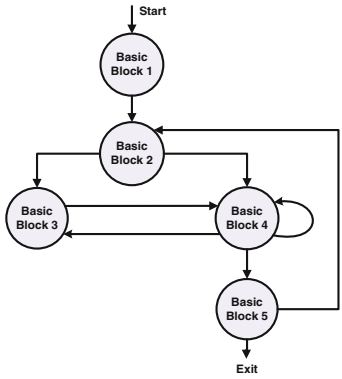


Fig. 1. A simple code and its related CFG of a bubble sort program

AS from an RTU informs the MTU that the sender RTU has not been crashed. As shown in Fig. 2(b) the signature has three fields. The first field (T) represents the type of the signature. The value of the T field for an Alive Signature is 2. The second field represents number of an RTU which has sent the signature. The third field shows the time of sending the AS in the RTU.

An Error Signature (ES) is sent when a control flow error being detected in an RTU by its local monitoring mechanism. Therefore, when an error occurs, *Send()* procedure would be run and this would be send ES to a function called CFE-Handler. Then programs control would be transferred to this function, which has two tasks. First, finding the basic block in which errors occurred by checking the value of the fields in ES. Second, sending an ES to MTU to handle error in RTU by checking the value of ES fields, in case that RTU crashes or cannot execute the program. Meanwhile, the technique has been designed in a way that if an error occurs in each basic block, it would be detected in next basic block and detection latency [20] would be reduced. Therefore, CFE-handler function needs ES signature fields.

As shown in Fig. 2(c), the ES has five fields. The first field (T) represents the type of the signature. The value of the T field for an error signature is 3. The second field (RN) represents number of an RTU which has been encountered a control flow error. The third field (BN) and the fourth field (DS_{BN}) represent number and assigned signature of a basic block, respectively, which a control flow error has occurred in it. Assessing these fields, the MTU discovers that a control flow error has occurred in which RTU and which basic block of that RTU's program (current basic block in that RTU). The last field (DS_{curr}) represents the signature of the preceding basic block of the current basic block in the faulty RTU.

Inserting Control Flow Checking Instructions. To check the control flow of an RTU's program, some control flow checking instructions are inserted in each basic block of the program. In Fig. 3(a), the structure of a basic block after applying the ICS-CFC technique has been presented.

The DS_{BN} variable contains the signature of a basic block and is composed of N bits, if the program has N basic blocks. In this variable, the i^{th} (DS_{BN}[i-1]) and j^{th}

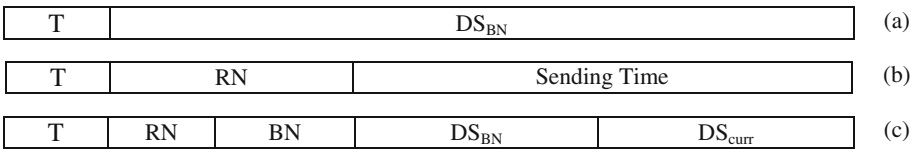


Fig. 2. The structures of three types of signatures in the ICS-CFC technique

(DS_{BN}[j-1]) bits are set to 1, if the BB_i and BB_j are successors of the current basic block. Another variable, called DS_{curr} contains the signature of a basic block should be executed before the current basic block and initialize to '0000…1'. This variable is checked and updated in the middle of the current basic block. The BN is number of a basic block and a variable called BN_{curr} is used to store number of a basic block should be executed after the current basic block. This variable is updated at the end of each basic block. In the ICS-CFC technique, if a basic block has more than one successor, a unique number is assigned to each successor, called SN. The SN variable is set at the end of a basic block with more than one successor basic block and is checked in the middle of the executed successor basic block.

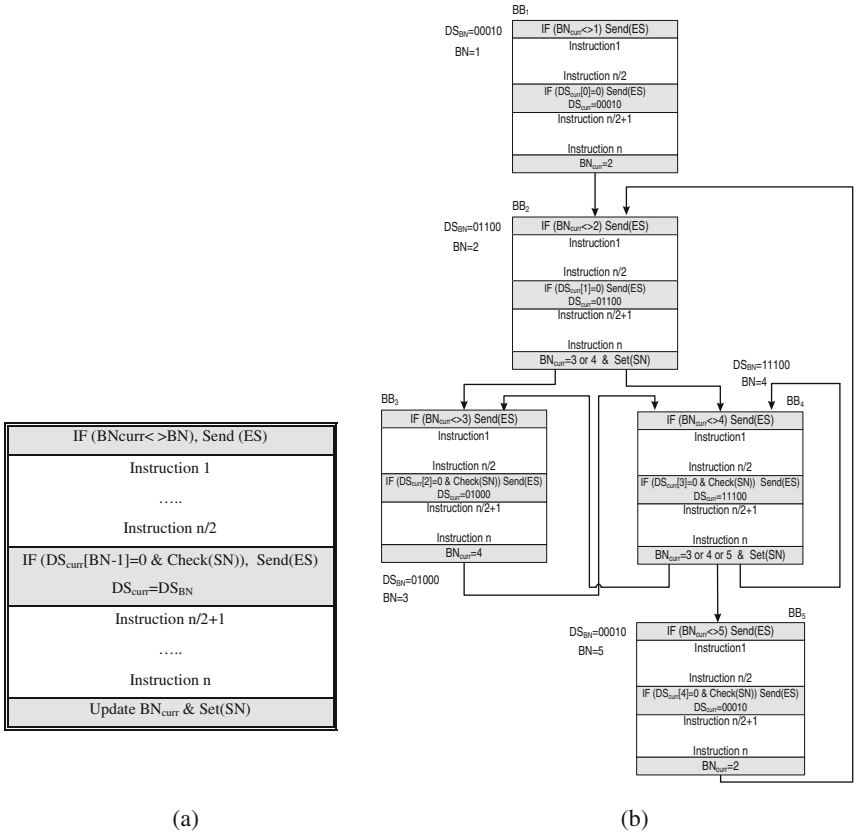
As shown in Fig. 3(a), in the beginning of each basic block, the flow upon its entrance into the basic block is checked by comparing the BN_{curr} with BN. If a mismatch is detected, due to an illegal jump to the beginning of the current basic block, an ES is sent to the CFE-Handler function.

In the middle of each basic block, DS_{curr} is checked. If the execution flow of the program is correct, n^{th} bit of DS_{curr} (DS_{curr}[BN-1]) in basic block BB_n should be '1'. In this situation, DS_{curr} is updated with the value of DS_{BN}. Otherwise, if that bit was '0', a control flow error is detected and an ES sent to the CFE-Handler function. In addition to this check, the value of SN should be assessed in the middle of each basic block by *check()* procedure. If the current basic block is one of the more successors of its predecessor basic block, SN value should be compared with successor number of the current basic block. If there a mismatch is detected, a control flow error is detected and an ES sent to the CFE-Handler function.

At the end of each basic block, BN_{curr} is updated with number of next basic block which should be executed after the current basic block. Moreover, if the current basic block has more than one successor blocks, SN is also updated with number of successor basic blocks by *set()* procedure, which should be run after the current basic block. In order to reduce detection latency, it is possible to insert the instructions, which have been added at the beginning, to the end as well. So, the error detection would be occurred in the basic block and *send()* procedure would be run.

The ICS-CFC technique divides a basic block into two parts. Analyses show that this technique can detect almost all illegal branches to a basic block from another basic block, even for branches from the first part of a basic block to its second part. Despite the software-based techniques, the proposed technique is able to detect the program crash, employing by the MTU.

Figure 3(b) illustrates the control flow graph of the bubble sort benchmark (presented in Fig. 1), after applying the ICS-CFC technique on it.



(a) (b)

Fig. 3. The structures of basic blocks in the proposed technique

3.2 ICS-CFC Technique Implementation in the MTU

As mentioned before, an Alive Signature (AS) is sent to the MTU by each RTU in a certain period of time. Receiving an AS from an RTU shows that the RTU has not been crashed. In the configuration time of an ICS, the MTU configures and programs each RTU. In this technique, at the configuration time, the time period of sending AS is set for each RTU and a profile for each of them is created in the MTU.

After receiving a signature from an RTU, the MTU recognizes the type of the signature. If the received signature is an AS, the MTU checks the "sending time" field of the signature and compares it with the "sending time" of the last received AS stored in the RTU's profile. The MTU detects a control flow error in that RTU, if the time between two received Alive Signatures is more than the predetermined time period, which is also stored in that RTU's profile. Inserting "sending time" into an Alive Signature ensures that network latencies cannot affect the correctness of the technique. In this situation, the network latency can only cause to detect a control flow error, later.

Moreover, if a received signature from an RTU is an ES, the content of the signature show that a control flow error has been occurred in which basic block of the

RTU's program. Employing this feature, the MTU can display the information to its human operator to select the best strategy to encounter the problem.

3.3 Overhead Analysis of the ICS-CFC Technique

When error detecting technique implemented on main program, there are different parameters which impose overheads on a system. Thus, to improve the proposed technique, the trade-off between parameters should be considered. In this section, the overhead of the ICS-CFC technique on SCADA system will be presented.

The ICS-CFC technique proposed in a client/server communication network. Therefore, network traffic is an overhead, because only Alive Signatures are sent over the network in the error free conditions. Moreover, the time period of sending these signatures is adjustable due to the network's properties. In addition to Alive Signatures, Error Signatures are only sent over the network when an error occurs in an RTU. In addition, since the sending time of an Alive Signature is stored in the signature, the network latency cannot affect the correctness of the technique. This latency can cause the MTU to detect an RTU's program crash with some delays. Furthermore, if new RTUs being added to network, then the number of Alive Signature will increase and lead to network latency. Thus, regarding SCADA structure and number on RTUs and Alive Signature sending time it would be possible to make a trade off.

The proposed technique has some memory and performance overheads due to inserted instructions into the main program. Some instructions which have been added at the beginning, in the middle or at the end of each basic block; But, the error detecting capability will increase and detect latency will decrease. Therefore, these instructions being customized and make a trade-off between parameters.

Compared with the other typical control flow checking techniques, the ICS-CFC technique does not impose any hardware redundancy. In this technique, the existed MTU is also employed as a hardware monitor. And for monitoring there is no need to add a watchdog for each RTU and therefore it would cause cost decrease in system.

4 Experimental Results

In this section, the setup environment is explained and the experimental results are given. In order to analyze proposed technique, first the fault models determined, then the technique applied to two following environment: (1) A distributed local network compose of some PC, (2) A real ICS network compose of some industrial PLCs. Thus, based on fault models, the faults in these two environments injected and results evaluated accordingly.

4.1 The Models of Fault

A fault can occur in a system memory, system bus and Internal CPU in RTUs. The errors which may occur due to these faults can be modeled as CPU Crash, Data errors and CFE. CPU Crash happens when the processor does not work and can be detected

by MTU, which is used in proposed technique. Data error is detectable by some techniques like assertion. CFE may occur either in memory content or processor internal registers.

Fault injection approach is used that has three kinds: (1) Random Branch Insertion: replacing a non-control instruction with a control instruction. (2) Random Branch Deletion: deleting some branches of the program randomly. (3) Random Branch Modification: the target address of a control instruction is being modified.

By use of the mentioned fault models, the behavior of a CFE in memory can be exactly modeled. These models are not capable to represent the CFE behavior due to an error occurred in processor internal registers. In these cases, the behavior of such errors can be modeled by manipulating Programs Counter (PC) and Status Register (SR). The control flow errors will be produced and also the efficiency of different techniques can be compared with each other by applying above mentioned fault models. The faults are randomly injected to the assembly code of benchmarks. By changing registers and program counter of the program, control flow errors would occur in the program. Thus, the efficiency of different techniques can be evaluated.

4.2 ICS-CFC Technique Execution in a Local Network

To evaluate the proposed technique, the ICS-CFC technique was applied to a local network composed of eight personal computers as RTUs connected to a main server as a MTU through a client/server communication network. Each RTU had an Intel Core i5 as its CPU, 4 GB RAM, and Windows 7 as its operating system. MTU had an Intel Core i7 as its CPU, 8 GB RAM, and Windows server 2008 as its operating system. The communication protocol was the TCP/IP and the Microsoft Visual Studio 2008 was employed to implement the benchmark programs. To communicate between RTUs and MTU over network and send signature, socket programming was employed [21]. Four benchmark programs, i.e. Bubble Sort (BS), Matrix Multiplication (MM), Quick Sort (QS), and Linked List Insertion (LLI), which are typical benchmarks employed in previous researches, were implemented on each RTU and ICS-CFC technique was applied to them.

For experimental evaluation the ICS-CFC technique was applied to a personal computer as an RTU and 30,000 faults were injected, based on fault models, into the mentioned benchmarks. Seven versions are considered for each benchmark. First is the original code (the code of the benchmark), to which other six techniques being applied. For each version, the fault injection randomly executed, based on fault models. According to the effects of the injected faults in the RTU's program, five different cases occur: (1) Correct Result (CR): injected fault does not change the final result of the program and no control flow error is detected. (2) Wrong Result (WR): fault results in a wrong output and is not detected by the ICS-CFC technique. (3) Time Out (TO): injected fault caused the program execution time to change and it does not finish in a specified time. This type of errors is detected by the MTU in the ICS-CFC technique. (4) Os Exception (OS): These faults cause the operating system exception. Generally, this percentage of faults is regarded as being detected by the operating system. (5) Single Detection (SD): injected fault results in a control flow error and is detected

by techniques in the RTU. The fault injection results into the four benchmarks for some software techniques running on the RTU have been presented in Table 1.

Table 1. Experimental evaluation average results of CFCSS [14, 15], ECCA [13], RSCFC [16], I2BCFC [25], SCFC [17], and ICS-CFC

Techniques	CR %	WR %	TO %	OS %	SD %	Memory overhead %	Performance overhead %	Fault coverage %	Evaluation result %
No CFC	38.10	42.20	07.01	12.69	00.00	00.00	00.00	31.83	31.83
CFCSS	40.48	12.01	08.30	08.59	30.62	31.30	25.00	79.82	43.49
ECCA	38.96	09.50	06.50	10.15	34.89	33.00	27.80	84.43	44.51
RSCFC	37.11	11.02	06.20	11.13	34.54	35.00	28.10	82.48	42.71
I2BCFC	32.37	06.05	08.15	07.70	45.73	35.80	31.00	91.05	45.51
SCFC	31.95	05.36	08.35	05.40	48.94	36.10	32.50	92.12	45.56
ICS-CFC	27.37	02.67	07.50	04.15	58.31	36.79	33.20	96.32	47.42

As presented in Table 1, among all 30,000 injected faults, 27.37 % faults lead to correct output results. Among all the remaining injected faults, only 02.67 % injected faults, which result in wrong output results, were not detected by the ICS-CFC technique. Therefore, the fault coverage of the proposed technique is about 96.32 %.

Table 1 presents average fault coverage for some techniques and also shows the memory and the performance overheads in average for the above mentioned techniques. The performance overhead of the ICS-CFC technique, due to execution of the redundant instructions, is about 33.20 %. The memory overhead of the proposed technique, due to insertion of redundant instructions into basic blocks of a program and signature variables, is about 36.79 %.

Moreover, the new parameter Evaluation Result (ER), which has been defined, would cover fault coverage, memory and performance overheads concurrently. This technique should be able to balance these parameters with each other. The ER is defined as follows:

$$\mathrm{ER} = \frac{\text{Fault Coverage}}{\text{Memory Overhead} * \text{Performance Overhead}} * 100 \tag{1}$$

Table 1 shows ER for some techniques, in which ICS-CFC has greater ER than other techniques and is more appropriated for employing in safety-critical systems.

The ICS-CFC technique does not have any hardware redundancy, compared to other typical hybrid-based control flow checking techniques. The existed MTU is also employed as a hardware monitor. It should be noted that in ICS-CFC technique, RTUs send the signatures to the MTU over the network without any effect on the performance of the RTUs.

4.3 Case Study: Execution ICS-CFC Technique in a Real ICS

The mentioned benchmarks are small and limited and their confidence level is too low and just used for different techniques comparison. Therefore, in order to clarify

technique better, it has been tested precisely and which high confidence level in case study [22]. So, it has been implemented on some big benchmarks and program's robustness in industrial environment.

The ICS-CFC technique applied to a real ICS distributed network comprise of three Steam Boiler (Steam boiler is basically a closed vessel into which water is heated until the water is converted into steam at required pressure) devices and three RO (Reverse osmosis is a process in which dissolved inorganic solids like salts are removed from a solution like water) devices and an HP server. There have been 3 PLC modules set up above mentioned machines, and are being monitored by HP server through the wireless network. SIMATIC S7 software installed on the server and some benchmarks, with STEP7 programming language, implemented on PLCs. Benchmarks specifications have been shown in Table 2.

Table 2. Benchmark programs used in the experiments

Benchmarks	Devices	PLC models	#Basic blocks	Memory overhead	Performance overhead
SB200E	Steam Boiler1	SIMATIC ET200	39	13.00 %	28.34 %
SB300S	Steam Boiler2	SIMATIC S7-300	41	15.67 %	27.10 %
SB400S	Steam Boiler3	SIMATIC S7-400	42	14.10 %	25.85 %
R200E	RO1	SIMATIC ET200	26	15.25 %	26.09 %
R300S	RO2	SIMATIC S7-300	29	16.30 %	24.10 %
R400S	RO3	SIMATIC S7-400	33	16.45 %	23.50 %

For experimental evaluation the ICS-CFC technique was applied to PLCs and 30,000 faults were injected into the mentioned benchmarks. Two versions are considered for each benchmark. The first is the original code, to which ICS-CFC technique is being applied. Table 3 shows the experimental results of the original program and ICS-CFC programs, respectively. For each version, according to the effects of the injected faults in the PLC's programs, five different cases occur: Correct Result, Wrong Result, Time Out, Os Exception and Single Detection. The occurrence percentages of these cases are shown in Table 3.

Table 3. Experimental evaluation average results of NO-CFC and ICS-CFC

Benchmarks	No CFC					ICS-CFC				
	CR %	WR %	TO %	OS %	SD %	CR %	WR %	TO %	OS %	SD %
R400S	14.20	64.45	11.60	09.75	0.00	11.00	04.30	08.90	07.60	68.20
R300S	13.80	64.15	12.09	09.96	0.00	12.30	05.90	09.10	07.90	64.80
R200E	15.00	63.80	10.67	10.53	0.00	12.80	05.20	09.90	08.50	63.60
SB400S	12.90	65.10	12.90	09.10	0.00	09.10	06.30	10.80	07.10	66.70
SB300S	13.80	62.90	11.45	11.85	0.00	10.10	05.28	08.60	06.98	69.04
SB200E	11.65	59.20	09.60	19.55	0.00	08.30	06.10	08.90	07.20	69.50

Table 2 presents the memory and the performance overhead of 6 benchmarks. The average memory overhead of the ICS-CFC technique, due to insertion of redundant

instructions into basic blocks of a program and signature variables, is about 15.13 %. The average performance overhead of the technique, due to execution of the redundant instructions, is about 28.83 %.

5 Conclusions

Industrial Control Systems (ICS) are essential factors to ensure execution of an industrial process safe and successfully. SCADA system, a type of an ICS, is a widely distributed system primarily used to remotely control and monitor of industrial processes from a central location. SCADA covers the transfer of data between a server as MTU and a number of remote sites as RTUs. Since an RTU works in a harsh environment, fault tolerance is one of the most significant among many challenges in industrial networks. In this paper, a hybrid control flow checking technique, called ICS-CFC, was proposed in order to detect control flow errors in RTUs. This technique is employed to monitor the program execution flows of several RTUs in a SCADA system as a distributed system. The proposed technique can be applied to all ICSs which employ microcontrollers, microprocessors, PLCs, or personal computers as their RTUs. To evaluate the fault coverage of ICS-CFC technique, 30,000 faults were injected on distributed system. Among all injected faults, the ICS-CFC technique detects more than 96.32 % of faults resulted.

Acknowledgement. The authors would like to appreciate Parin Beton Amood Company for providing the opportunity of field work and evaluation of the ICS-CFC technique in a real Industrial Control System.

References

1. Mollah, M.B., Islam, S.S.: Towards IEEE 802.22 based SCADA system for future distributed system. In: Proceedings of IEEE International Conference on Informatics, Electronics & Vision, pp. 1075–1080. Dhaka, Bangladesh, 18–19 May 2012
2. Atlagic, B., Milinkov, D., Sagi, M., Bogovac, B.: High-performance networked SCADA architecture for safety-critical systems. In: Proceedings of the Second Eastern European Regional Conference on the Engineering of Computer Based Systems, pp. 147–148, Bratislava, Slovakia, 5–6 September 2011
3. Avhad, M., Divekar, V., Golatkar, H., Joshi, S.: Microcontroller based automation system using industry standard SCADA. In: Proceedings of Annual IEEE India Conference, pp. 1–6. Mumbai, India, 13–15 December 2013
4. Qiang, Z., Danyan, C.: Design and implementation of distribution network SCADA system based on J2EE Framework. In: Proceedings of International Forum on Information Technology and Applications, pp. 633–636. Chengdu, China, 15–17 May 2009
5. Misbahuddin, S.: Fault tolerant remote terminal units (RTUs) in SCADA systems. In: Proceedings of International Symposium on Collaborative Technologies and Systems (CTS), pp. 440–446. Chicago, USA, 17–21 May 2010

6. Tan, L., Tan, Y., Xu, J.: CFEDR: control-flow error detection and recovery using encoded signatures monitoring. In: Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 25–32. New York, USA, 2–4 October 2013
7. Mahmood, A., McCluskey, E.J.: Concurrent error detection using watchdog processors-a survey. J. IEEE Trans. Comput. **37**(2), 160–174 (2002)
8. Makoto, S.: A dynamic continuous signature monitoring technique for reliable microprocessors. J. IEICE Trans. Electron. **94**(4), 477–486 (2011)
9. Chen, Y.Y., Leu, K.L.: Signature-monitoring technique based on instruction-bit grouping. IET Proc. Comput. Digital Tech. **152**(4), 527–536 (2005)
10. Sedaghat, Y., Miremadi, S.G., Fazeli, M.: A software-based error detection technique using encoded signatures. In: Proceedings of the 21st IEEE International Symposium on Fault-Tolerance in VLSI Systems (DFT 2006), pp. 389–400. Arlington, USA, 4–6 October 2006
11. Benso, A., Carlo, S.D., Natale, G.D., Prinetto, P.: A watchdog processor to detect data and control flow errors. In: Proceedings of the 9th IEEE On-line Testing Symposium, pp. 144–148, 9–7 July 2003
12. Horst, R.W., Harris, R.L., Jardine, R.L.: Multiple instruction issue in the nonstop cyclone processor. In: Proceedings of the 17th International Symposium on Computer Architecture, pp. 216–226. Seattle, Washington, USA, 28–31 May 1990
13. Nicolescu, B., Velazco, R.: Detecting soft errors by a purely software approach: method, tools and experimental results. In: Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE 2003), pp. 57–62. Munich, 3–7 March 2003
14. Oh, N., Shirvani, P.P., McCluskey, E.J.: Control-flow checking by software signatures. J. IEEE Trans. Reliab. **51**(1), 111–122 (2002)
15. Yu, J., Garzaran, M.J., Sni, M.: Techniques for efficient software checking. In: Proceedings of the 20th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2007), pp. 16–31. Urbana, Illinois, USA, 11–13 October 2007
16. Li, A., Hong, B.: On-line control flow error detection using relationship signatures among basic blocks. J. Comput. Electr. Eng. **36**(1), 132–141 (2010). Elsevier
17. Asghari, S.A., Taheri, H., Pedram, H., Kaynak, O.: Software-based control flow checking against transient faults in industrial environments. J. IEEE Trans. Indus. Inform. **10**(1), 481–490 (2014). IEEE
18. Fazeli, M., Farivar, R., Miremadi, S.G.: Error detection enhancement in powerpc architecture-based embedded processors. J. Electron. Test. Theory Appl. (JETTA) **24**(1–3), 21–33 (2008). Springer
19. Koren, I., Krishna, C.M.: Fault-Tolerant Systems. Elsevier, San Francisco (2007)
20. Chaudhari, A., Park, J., Abraham, J.: A framework for low overhead hardware based runtime control flow error detection and recovery. In: 2013 IEEE 31st VLSI Test Symposium (VTS), pp. 1–6. Berekley, CA, April 2013
21. Xue, M., Zhu, C.: The socket programming and software design for communication based on client/server. In: Proceedings of Pacific-Asia Conference on Circuits Communications and System (PACCS), pp. 775–777. Chengdu, China, 16–17 May 2009
22. Leveugle, R., Calvez, A., Maistri, P.: Statistical fault injection: quantified error and confidence. In: Design, Automation and Test in Europe Conference and Exhibition, DATE 2009, pp. 502–506. Nice, 20–24 April 2009
23. Rysavy, O., Rab, J., Halfar, P.: A formal authorization framework for networked SCADA systems. In: Proceedings of the 19th International Conference and Workshops on Engineering of Computer Based Systems (ECBS), pp. 298–302. Serbia, 11–13 April 2012

24. Regel, R.G., Parameswaran, S.: Hardware assisted pre-emptive control flow checking for embedded processors to improve reliability. In: Proceedings of the 4th International Conference hardware/software codesign and system synthesis, pp. 100–105, Seoul, Korea, 22–25 October 2006
25. Asghari, S.A., Taheri, H., Pedram, H., Abdi, A.: An effective intra-inter block control flow checking method against single event upsets. Res. J. Appl. Sci. Eng. Tech. **4**, 4367–4379 (2012)