Telecommun Syst (2017) 65:155–168
DOI 10.1007/s11235-016-0223-x

Content-based deep communication control for networked control system

Ming Wan[1,2] · **Wenli Shang**[1,2] · **Linghe Kong**[1,3] · **Peng Zeng**[1,2]

Published online: 30 August 2016

Abstract In smart cities, the networked control system plays a significant role in transportation systems, power stations or other critical infrastructures, and it is facing many security issues. From this point, this paper proposes a content-based deep communication control approach to guarantee its security. Based on the layer architecture, this approach analyzes the interactive content in depth according to different industrial communication protocols, and implements the access control between two distinct enclaves. For OPC Classic, we acquire the dynamic port provided by OPC server, and open a new connection belonging to this port; for Modbus/TCP, we not only analyze the ordinary function codes and addresses, but also check the register or coil values by using the multi-bit Trie-tree matching algorithm. Besides, the white-listing strategy is introduced to satisfy the special requirements of industrial communication. Our experiment results show that, on the one hand the proposed approach provides OPC and Modbus/TCP defenses in depth; on the other hand it has less than 1 ms forwarding latency and 0 packet loss rate when the rule number reaches 200, and all these meet the availability requirements in the networked control system. In particular, this approach has been successfully applied in several real-world petrochemical control systems.

Keywords Content-based deep communication control · OPC Classic · Modbus/TCP · White-listing

1 Introduction

As an iconic symbol, the networked control system [1] has become an important basis in critical infrastructures, such as power stations, petrochemical plants and transportation systems, and obtains the public attention gradually. Especially, the Industry 4.0 revolution, defined by Germany, further emphasizes the essential role of networking technology in the networked control system [2]. However, the application of networking technology has broken the original closure of industrial control systems, such as SCADA (supervisory control and data acquisition), DCS (distributed control system) and PLC (programmable logic controller), and the accompanying security issues are increasing exposed [3–5]. The research reports of the USA ICS-CERT (industrial control systems cyber emergency response team) point out in recent years the number of security incidents in industrial control systems show a sensible rise, and only in 2014 the attack number against critical infrastructures reaches 245 [6]. Especially, the Stuxnet virus attacking Irans nuclear facilities sounds an alarm all over the world [7]. As described by Hadziosmanovic [8], one critical reason is due to the fact that these systems are not built with security in mind. Although there are various kinds of security methods in the regular

✉ Ming Wan
wanming@sia.cn

Wenli Shang
shangwl@sia.cn

Linghe Kong
linghe.kong@sjtu.edu.cn

Peng Zeng
zp@sia.cn

1 Shenyang Institute of Automation Chinese Academy of Sciences, No.114, Nanta Street, Shenhe District, Shenyang 110016, People's Republic of China

2 Key Laboratory of Networked Control System Chinese Academy of Sciences, No.114, Nanta Street, Shenhe District, Shenyang 110016, People's Republic of China

3 Shanghai Jiao Tong University, No. 800, Dongchuan Road, Minhang District, Shanghai 200240, People's Republic of China

IT system, the information security in the networked control system is essentially different from that in the IT system, and the traditional security methods cannot be applied directly.

The vulnerability of the networked control system has been widely studied and discussed by both academia and industry, mainly including: information management vulnerability [9], operating system and database vulnerability [10,11], embedded control device vulnerability [11–13], and industrial communication protocol vulnerability [14–16]. As the major communication carrier for different industrial control devices, the industrial communication protocols are always utilized by malicious hackers as a breakout to perform various attacks. The primary causes can be summarized as follows: on the one hand, many industrial communication protocols, such as Modbus/TCP and OPC Classic, belong to the application layer protocol specification, and rely on the underlying TCP/IP protocol to support logical connection. So, this design may inherit some security issues caused by the TCP/IP protocol. On the other hand, in the original design of industrial communication protocols, the engineers do not consider the information security problems, and the protocols lack the appropriate security mechanisms [17]. To sum up, in order to effectively defend network attacks, a feasible strategy is to parse the industrial communication protocol in depth when we design the defense approach. At the current stage, a famous approach to improve the security in the networked control system is the communication control approach, and its typical application is industrial firewalls or industrial NetGaps. By using DPI (deep packet inspection) technology, industrial firewalls run an extended analysis of industrial communication packets according to the specific protocol format, and filter the malicious attack flows [18,19]. Although the existing approaches can protect the networked control system from cyber attacks in some degree, some shortages are summarized as follows: First, the rule granularity is not fine, and the defense capability is limited. For example, for Modbus/TCP, most existing industrial firewalls only check the function codes, and miss detecting the register or coil values. Second, the efficiency of these approaches need more considerations, because the availability of the networked control system comes first. Therefore, it is worth considering the above problems when we design an more effective and feasible communication control approach.

It is generally appreciated that NIST (National Institute of Standards and Technology) emphasizes the defense in depth strategy, which divides the industrial control system into different security enclaves [11]. On this basis, this paper proposes a content-based deep communication control approach to secure the networked control system. Differently, this approach can carry out the deep analysis of industrial communication protocols which are included in the highest layer of the OSI model. By this way, this approach can block the improper control commands which only exist in the application layer. Furthermore, the content-based communication control can be explained as follows: by capturing the packets from one enclave to another enclave, this approach analyzes the interactive information in depth and restores the critical contents according to the protocol format. After matching the critical contents with the white-listing rules, this approach can filter the malicious or undesired communication flows. Based on the basic architecture of the networked control system, this approach supports two industrial communication protocols: OPC Classic and Modbus/TCP. For OPC Classic, this approach acquires the dynamic port of the OPC server, and opens a new connection belonging to this port. It can prevent the access request which uses the illegal port, and filter the packets which cannot conform to the DCE/RPC message type. For Modbus/TCP, this approach can provide three key parts, including packet integrity detection, deep parsing based on function code and multi-bit Trie-tree matching algorithm. It not only analyzes the ordinary function codes and addresses, but also checks the register or coil values by using the multi-bit Trie-tree matching algorithm. By the experimental evaluation and analysis, it is particularly worth mentioning that our approach can meet the availability requirements in the networked control system when it provides the security protection for OPC and Modbus/TCP communications. Besides, our approach has been successfully applied in several real-world petrochemical control systems.

The rest of this paper is organized as follows: Sect. 2 describes the basic layer architecture of the networked control system. Sect. 3 provides the proposed content-based communication control approach in detail. Our simulation and experimental results are presented in Sect. 4. Besides, we also depict our laboratorial demo and practical application in Sect. 5 and Sect. 6, respectively. Finally, Sect. 7 provides some concluding remarks regarding this research.

2 Basic layer architecture

According to the common characteristics, we sum up the basic architecture of the networked control system into three layers (see Fig. 1) [8,20]. More generally, Layer 1 is the basic control layer, and it consists of physical field devices (drivers, sensors, actuators, et al.) and programmable embedded electronic devices (PLCs, RTUs, DCS controllers, et al.). The programmable embedded electronic devices accomplish the industrial control process, and they not only provide the operational controls for the field devices through field bus, but also send the real-time signals to upper layer. Layer 2 is supervisory control layer, and it is composed of some critical servers and workstations, such as HMI (human machine interface), OPC server, engineer workstation, and so on. Furthermore, this layer performs several important tasks, for example, OPC

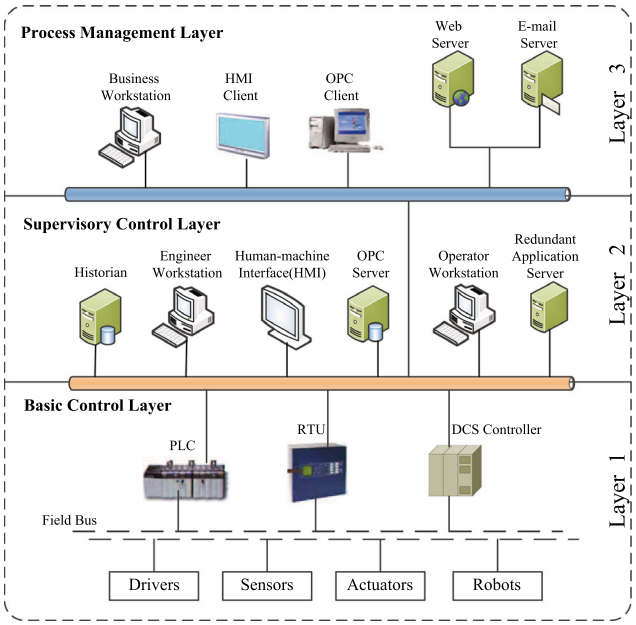

Fig. 1 Basic layer architecture of networked control system

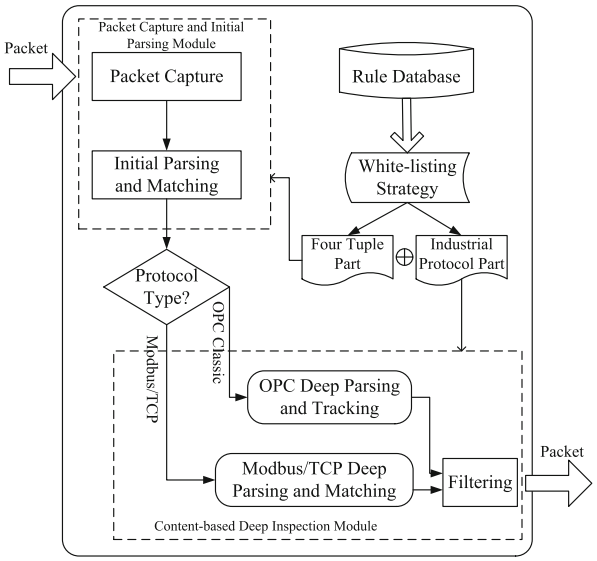

Fig. 2 Basic model of content-based deep communication control approach

server gathers the status information of PLCs and provides a method for OPC client to access its data. Engineer workstation can carry out the system configuration, install or update program elements, recover from failures, and provide system administrations. Layer 3, called process management layer, includes various clients and corporate network. In particular, OPC client is in this layer, and it associates with OPC server to display the information about field managements and process managements. In this basic layer architecture, the interaction between layers uses different industrial communication protocols. For example, the communication between Layer 1 and Layer 2 is typically performed over Modbus/TCP, and the communication between Layer 2 and Layer 3 is typically operated by OPC Classic.

3 Content-based deep communication control approach

3.1 Basic model

Figure 2 describes the basic model of our approach. By setting the white-listing rules, this model can control and filter the communication flows from one enclave to another. According to the packet processing, this model mainly includes two parts: packet capture and initial parsing module and content-based deep inspection module. In the packet capture and initial parsing module, the packets sent from some industrial devices (such as OPC clients or PLCs), can be captured by this module. After that, this module preliminarily analyzes the TCP/IP headers of the packets and extracts the four tuples. When the four tuples match with the rules in the white list, this module judges the protocol type of the packets according to the source or destination ports. If the protocol type is OPC Classic or Modbus/TCP, the application data of the packets can be delivered to the content-based deep inspection module. If the protocol type is OPC Classic, the application data can be parsed in depth to track the dynamic port. Otherwise, if the protocol type is Modbus/TCP, the application data can be analyzed hierarchically to check the key information. After matching with the white-listing rules, the illegal access can be filtered by this module. Besides, all rules are stored in the rule database, and we can create the white-listing strategy from this database.

3.2 OPC deep parsing and tracking

As is well known, OPC Classic is based on Microsoft's DCOM protocol, which brings a great challenge to the security and reliability of the networked control system [21,22]. Normally, most application protocols based on TCP or UDP use the unique port number to establish a new connection. Unfortunately, establishing an OPC connection requires the following two steps: firstly, OPC client queries OPC server by the port 135 to obtain another port number for next data connection; secondly, OPC client establishes a new connection to access data by the obtained port number. In fact, the used port number in the second step is a random port number which is assigned dynamically by OPC server, so it is not possible to know this port number in advance. To sum up, when protecting OPC server, the traditional IT firewall cannot meet the characteristic to open the dynamic port.

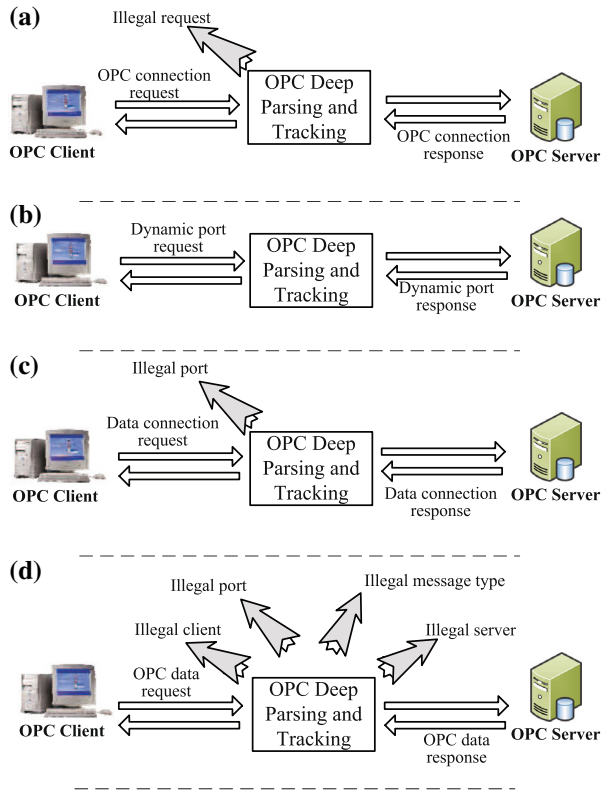

Fig. 3 Defense function of OPC deep parsing and tracking sub-module. **a** The first OPC connection request and response using the port 135, **b** the dynamic port request and response, **c** the new data connection request and response using the dynamic port, **d** the OPC data request and response

According to the OPC security specification [23], the communication control should be deployed between OPC server and OPC client, and the minimized authority constraint can be realized by controlling the access procedure. Figure 3 depicts the corresponding defense function of this sub-module. From this figure we can see that, in the whole OPC access procedures [from a to d], this sub-module can filter the illegal access requests. Especially, this sub-module can not only prevent the access request which uses the illegal port, but also filter the packets which cannot conform to the DCE/RPC message type. Based on the dynamic tracking, the main parsing process of this sub-module can be summarized as follows:

Step 1 When receiving the application data and the four tuples from the packet capture and initial parsing module, this sub-module firstly judges whether this packet belongs to the OPC connection message according to the port 135. If the packet comes from OPC server, it will be regarded as the OPC connection response. After that, based on OPC protocol format, this sub-module analyzes the application data in depth and gets the random port number which is allocated dynamically by OPC server.

Step 2 After obtaining the above four tuples and the port number, this sub-module matches the information with the doubly-linked lists which store the used four tuples and port number. If unmatched, this sub-module will establish a new doubly-linked list, and store the above four tuples and port number; if matched, this sub-module will close the corresponding port, and inform the client to resend the OPC connection request.

Step 3 In accordance with the doubly-linked lists, this sub-module generates the implicit white-listing rules which can open new connections. Besides, this sub-module also tracks the work status of each new connection. When the status is not active in a specified time, this sub-module will close this port and delete the corresponding doubly-linked list and white-listing rule. Specifically, the default setting of this specified time is 2 min, and it is mainly because that most of the time intervals between two continuous OPC data requests range from 30 to 60 s according to the practical experience. Therefore, in order to avoid missing the regular requests, this specified time is set to twice the maximum. Moreover, we can manually adjust this time based on the actual OPC application requirements.

3.3 Modbus/TCP deep parsing and matching

According to our earlier work [24], we further describe the Modbus/TCP deep parsing and matching sub-module, and this sub-module mainly includes three parts: packet integrity detection, deep parsing based on function code and multi-bit Trie-tree matching algorithm.

3.3.1 Packet integrity detection

When a malicious attacker wants to damage some Modbus slave, a very simple way is to construct a malformed packet which does not conform to Modbus/TCP protocol format. Therefore, when receiving the application data and the four tuples, this sub-module need to detect the packet's integrity according to Modbus/TCP protocol, including:

Firstly, according to Modbus/TCP packet format shown in Fig. 4, we check whether the application data consists of three parts: MBAP (Modbus application protocol) header, function code and data.

Secondly, we analyze whether the protocol identifier in MBAP header is set to 0x0000, because Modbus/TCP specifies that Modbus master and Modbus slave can identify Modbus/TCP by the protocol identifier.

Thirdly, we verify the application data length in Modbus/TCP packets. By comparing TCP load length with the length field value in MBAP header, we confirm that this packet does not carry excessively long application data. The judging formula is as follows:

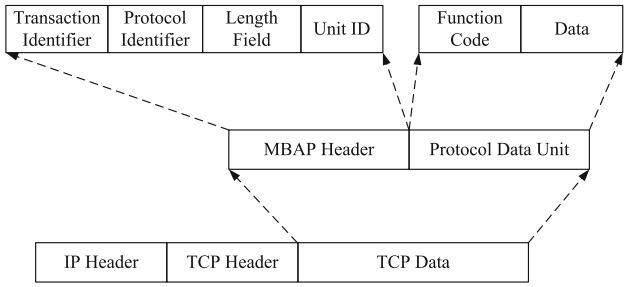

Fig. 4 Modbus/TCP packet format

$$L_{len-MBAP} = L_{TCP} - \sum_{i=1}^{3} LM_i \quad (1)$$

here, $L_{len-MBAP}$ is the length field value in MBAP header, L_{TCP} is the TCP load length, and LM_i (i=1,2,3) are the byte numbers of transaction identifier, protocol identifier and length field, respectively.

3.3.2 Deep parsing based on function code

By using DPI technology, this part analyzes the application data in depth according to different function codes, and extracts the key contents from each Modbus/TCP packet. Remarkably, this part not only analyzes the common function codes and addresses, but also acquires the register or coil values.

Based on different roles of all function codes, the forms of the data portion in the packet's format are distinct. For example, for the function code 01 whose function is to read coils, the data portion mainly includes start address (2 bytes) and coil number (2 bytes). Dissimilarly, for the function code 05 whose function is to write a single coil, the data portion primarily contains output address (2 bytes) and coil value (2 bytes). Thus, we should classify different function codes, and extract the key contents from the data portion according to the classification result. In our approach, we divide the function codes into three classes: read operation class, write operation class and other operation class. By the function code classification, we can obtain the more fine-grained composition information in Modbus/TCP packets, and the more detailed contents make the communication control more accurate. Also, the deep parsing based on function code facilitates the rule setting, and the rules can become more targeted.

Figure 5 depicts the pseudo-code of rule implementation algorithm based on different function code classes. For the rules from *Rule 1* to *Rule n*, we perform the following matching procedures: if the function code in the packet belongs to write operation class, we will continue to check the address and the register or coil value, and if matching, the *Pass* operation will be performed for this packet; if the function code in the packet belongs to read operation class, we will continue to check only the address, and if matching, the *Pass* operation will be performed for this packet; if the function code in the packet belongs to other operation class, we will continue to complete the following steps: if the function code is 22, we need to check the *And_Mask* and *Or_Mask* fields, and if matching, the *Pass* operation will be performed for this packet; else if the function code is 23, we need to check the read address, the write address and the register or coil value, and if matching, the *Pass* operation will be performed for this packet. Nevertheless, if all rules are not matched, the *Drop* operation will be performed to discard this packet.

```
for Rule 1 to Rule n
        ......
do check FunctionCode
  if FunctionCode ∈ [Writing Operation Set]
    then do check DataAddress
      if matched
        then do check DataValue
          if matched
            then do pass the packet
          else do jump to next rule
      else do jump to next rule
  else if FunctionCode ∈ [Reading Operation Set]
    then do check DataAddress
      if matched
        then do pass the packet
      else do jump to next rule
  else if FunctionCode ∈ [Other Operation Set]
    if FuncitonCode = 22
      then do check And_Mask and Or_Mask
        if matched
          then do pass the packet
        else do jump to next rule
      else if FuncitonCode = 23
        then do check ReadingDataAddress
          if matched
            then do check WritingDataAddress
              if matched
                then do check DataValue
                  if matched
                    then do pass the packet
                  else do jump to next rule
              else do jump to next rule
          else do jump to next rule
      else do jump to next rule
if n rules are unmatched
  then do drop the packet
```

Fig. 5 Pseudo-code of rule implementation algorithm

3.3.3 Multi-bit Trie-tree matching algorithm

In order to improve the rule matching efficiency, we build the multi-bit Trie-tree [25,26] to check the address and the register value. As is known to all, there may be many addresses and register values in one Modbus/TCP packet, for exam-

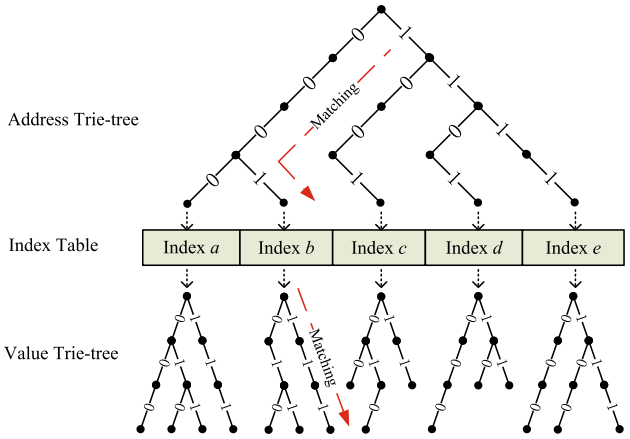

Fig. 6 Structure of multi-bit Trie-tree matching algorithm

ple, for the function code 04 whose function is to read the input register, one response packet from Modbus slave can contain up to 125 register values. So if we make the one-by-one comparison, it can yield poor performance. However, the multi-bit Trie-tree matching algorithm, which is widely used in the routing lookup approach, is a very powerful measure to solve this problem. In our multi-bit Trie-tree matching algorithm, we build two classes of multi-bit Trie-trees: the address Trie-tree and the value Trie-tree. More exactly, the address Trie-tree is composed of all addresses existing in some rule, and each branch on this tree represents an address. Also, the value Trie-tree is composed of all register values existing in the same rule, and each value Trie-tree represents the presetting register value range. Besides, the terminal node of each branch on the address Trie-tree stores an index which points to a value Trie-tree, and the index declares that the value of the corresponding register whose address is one branch of the address Trie-tree is limited by the pointed value Trie-tree. Figure 6 shows the structure of our multi-bit Trie-tree matching algorithm.

The matching procedures of this algorithm can be summarized as follows: when we extract one address from the Modbus/TCP packet, we firstly match with the address Trie-tree by using the linear search. If we find the corresponding branch on the address Trie-tree, we can get the value Trie-tree according to its index. After that, we can match the register value extracted from the packet with the value Trie-tree by using the linear search once again. If we can find the corresponding branch on the value Trie-tree, it means that the matching result is successful.

4 Evaluation and analysis

In this section, we evaluate our approach in detail by some simulation experiments and performance tests, and our main purpose is not only to discuss its obvious advantages but also to illustrate that it is suitable for security defense in the networked control system. In addition, we use Linux OS to implement our approach, which is based on the transparent bridge mode. The basic configuration is as follows: Linux kernel version is 2.6.24, system memory is 2 GB, CPU is Atom D525 (1.8 GHz), and the network interface card is 10/100/1000 Mbps adaptive Ethernet card. Especially, in the following parts of this paper we denominate the device applying our approach as industrial defense device (IDD).

4.1 Function demonstration

4.1.1 OPC defense

In this experiment, we deploy IDD between one OPC client and one OPC server to verify the OPC defense function. Furthermore, we use the software OPC Quick Client and the software KEPServerEx V4.0 as OPC client and OPC server respectively, and the corresponding IP addresses are 192.168.1.101 and 192.168.1.102. The rules of this experiment can be depicted as follows:

White-listing rule

[Operation: *Pass*] [Source IP: *192.168.1.101*] [Destination IP: *192.168.1.102*] [Protocol: *tcp*] [Source Port: *any*] [Destination Port: *135*] [State: *opc_tracking*]

Default rule

[Operation: *Drop*] [Source IP: *any*] [Destination IP: *any*] [Protocol: *any*] [Source Port: *any*] [Destination Port: *any*]

We carry out two experiments: in the first one, OPC client uses the legal IP address to access OPC sever; and in the second one, OPC client uses an illegal IP address to access OPC server. The difference of two experimental results reflects that our approach can open a dynamic port for the eligible OPC client and deny the anomalous access for the illegal OPC client.

Figure 7 shows the successful access result of the legal OPC client. From Fig. 7a we can see that, OPC Quick Client can adequately display the successful access process and receive the OPC data from KEPServerEx V4.0. Besides, Fig. 7b gives the interactive communication packets between OPC Quick Client and KEPServerEx V4.0. In this figure, packets 1–3 describe the first TCP connection establishment by the port 135. After that, a new dynamic port is negotiated by DCE/RPC protocol from packet 4 to packet 9. Finally, the last three packets explain that a new data connection is established by using the new dynamic port.

In the second experiment, we change OPC client's IP address to 192.168.1.100, which does not conform to the white-listing rules. Figure 8 depicts the failed access result of the illegal OPC client. As shown in Fig. 8a, OPC Quick Client is failed to communicate with KEPServerEx V4.0, and receives no OPC data. Similarly, in Fig. 8b, OPC Quick Client

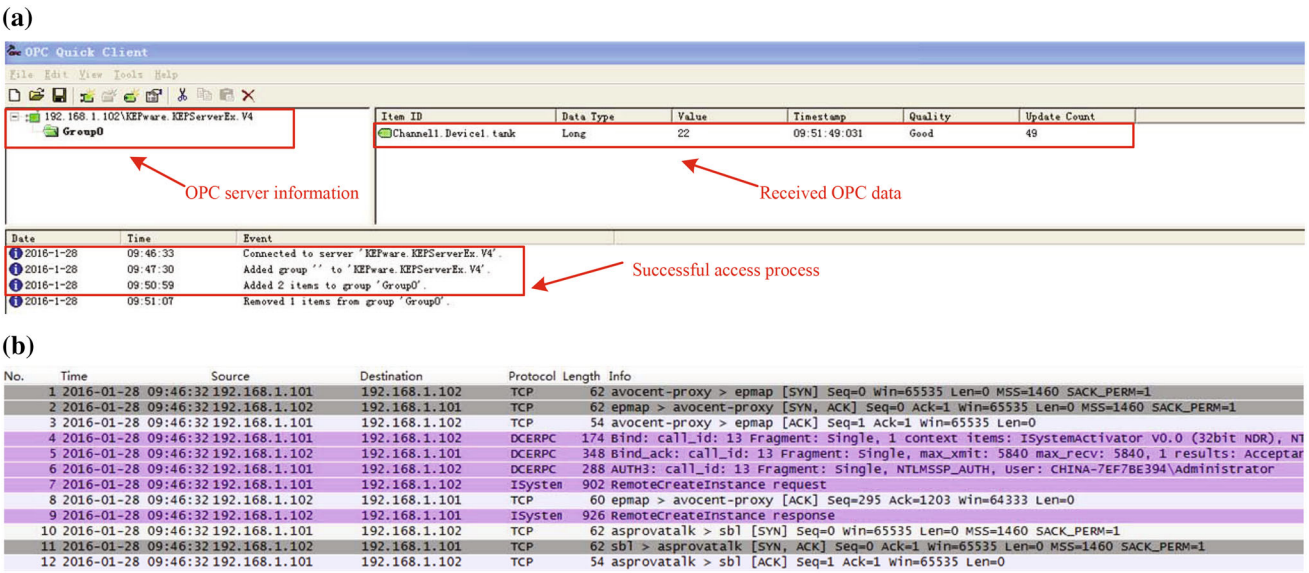

Fig. 7 Successful access result of the legal OPC client. **a** Result of OPC quick client, **b** communication packets between OPC quick client and KEPServerEx V4.0

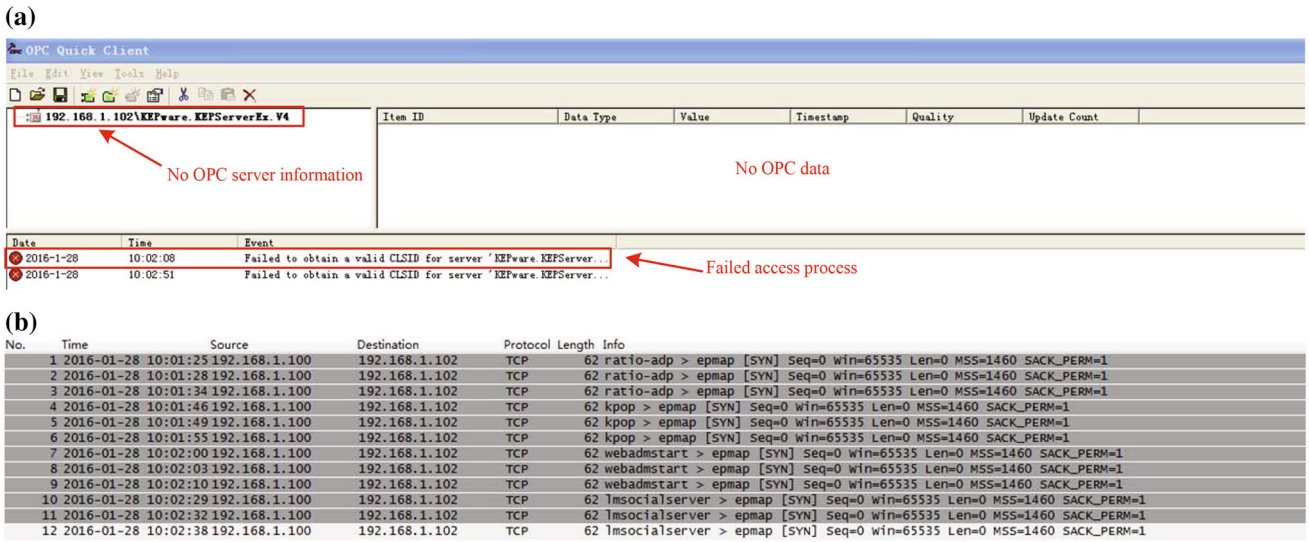

Fig. 8 Failed access result of the illegal OPC client. **a** Result of OPC quick client, **b** communication packets between OPC quick client and KEPServerEx V4.0

keeps sending the TCP connection request packet whose destination port is 135, and receives no corresponding response. That is to say, our approach can resoundingly intercept the unauthorized access request to OPC server.

4.1.2 Modbus/TCP defense

According to the function code classification, our approach compares the key contents in Modbus/TCP packets with the white-listing rules, and determines whether these packets are legitimate. It is worth mentioning that the rule setting for Modbus/TCP packets can support to match with the parsed contents, including the function codes, the addresses and the register or coil values. Compared with other existing communication control approaches for the networked control system [18,27,28], our rule granularity is finer. For example, Fig. 9 shows the granularity comparison between our rule setting and the general rule setting. Furthermore, our partial rule setting for Modbus/TCP is described below: the function code is 16 whose function is to write multiple registers, the address is limited between 50 and 100, and the register value is restricted between 100 and 200. If some Modbus/TCP packet conforms to the above rule, this packet will be forwarded, namely *Pass* operation.

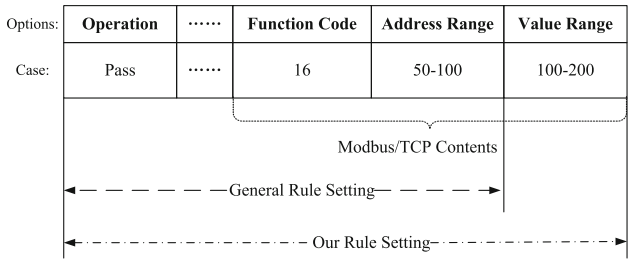

Fig. 9 Granularity comparison between our rule setting and the general rule setting

In order to prove the Modbus/TCP defense effect, we perform the corresponding experiments like the ones in [24], and also deploy IDD between one Modbus master and one Modbus slave. Furthermore, Modbus master uses the software modpoll to control Modbus salve. By setting the white-listing rules, IDD restricts the communication behaviors of Modbus master. In this experiment, we use the function code 16 to write multiple registers of Modbus slave, and the IP address of Modbus master and Modbus slave is 192.168.1.101 and 192.168.1.4, respectively. Besides, the service port of Modbus slave is 502. The rules of this experiment can be described as follows:

White-listing rule

[Operation: *Pass*] [Source IP: *192.168.1.101*] [Destination IP: *192.168.1.4*] [Protocol: *tcp*] [Source Port: *any*] [Destination Port: *502*] [Function Code: *16*] [Address Range: *9-13*] [Value Range: *80-120*]

Default rule

[Operation: *Drop*] [Source IP: *any*] [Destination IP: *any*] [Protocol: *any*] [Source Port: *any*] [Destination Port: *any*]

Under normal circumstances, we first use modpoll to execute a write operation which aims to change one register of Modbus slave. Here, the function code of this operation is 16, and the address and the value of the register is 10 and 100, respectively. As we known, this operation is fully consistent with the above-mentioned white-listing rule. Figure 10a shows the modpoll's execution result. From this figure we can see that, modpoll presents "Writing 1 reference.", and the written value of the 10th register is 100. That is, the communication packets of this operation are favorably forwarded by IDD to Modbus slave, and the operation is executed smoothly.

In order to verify the ability to drop the illegal value, we execute an abnormal write operation which changes the value of the 10th register to 200. Figure 10b shows the modpoll's execution result. In this figure, modpoll presents "Reply time-out!", that is, there is an error in the communication between Modbus master and Modbus slave and the response from Modbus slave may time out. In this case, it clearly indicates that the communication packets of this abnormal operation have been successfully dropped by IDD and modpoll cannot

(a)

```
C:\Users\Administrator>D:\modpoll\win32\modpoll.exe -m tcp -t 4 -r 10 192.168.1.
4 100
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration...: address = 1, start reference = 10, count = 1
Communication.........: 192.168.1.4, port 502, t/o 1.00 s, poll rate 1000 ms
Data type.............: 16-bit register, output (holding) register table

Written 1 reference.
```

(b)

```
C:\Users\Administrator>D:\modpoll\win32\modpoll.exe -m tcp -t 4 -r 10 192.168.1.
4 200
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration...: address = 1, start reference = 10, count = 1
Communication.........: 192.168.1.4, port 502, t/o 1.00 s, poll rate 1000 ms
Data type.............: 16-bit register, output (holding) register table

Reply time-out!
```

(c)

```
C:\Users\Administrator>D:\modpoll\win32\modpoll.exe -m tcp -t 4 -r 14 192.168.1.
4 100
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration...: address = 1, start reference = 14, count = 1
Communication.........: 192.168.1.4, port 502, t/o 1.00 s, poll rate 1000 ms
Data type.............: 16-bit register, output (holding) register table

Reply time-out!
```

(d)

```
C:\Users\Administrator>D:\modpoll\win32\modpoll.exe -m tcp -t 0 -r 10 192.168.1.
4 1
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration...: address = 1, start reference = 10, count = 1
Communication.........: 192.168.1.4, port 502, t/o 1.00 s, poll rate 1000 ms
Data type.............: discrete output (coil)

Reply time-out!
```

Fig. 10 Modpoll's execution results under different conditions. **a** Modpoll's execution result under normal circumstances, **b** modpoll's execution result when the value is 200, **c** modpoll's execution result when the register address is 14, **d** modpoll's execution result when when the function code is 05

receive the response packets from Modbus slave. So, this abnormal operation is executed unsuccessfully.

Similarly, in order to verify the ability to drop the illegal address, we also execute an abnormal write operation which writes the value 100 to the 14th register. Figure 10c shows the modpoll's execution result. As it shown, modpoll also presents "Reply time-out!", and the result of this abnormal operation is the same as the above. Besides, in order to verify the ability to drop the illegal function code, we execute a write single coil operation whose function code is 05, and the 10th coil is written to 1. Figure 10d shows the modpoll's execution result. The "Reply time-out!" appears in this figure, and this operation is also executed unsuccessfully.

As can be seen from the above ordinary experiments, the communication control approach in this paper can restrict the access behaviors of Modbus master by matching the communication packets in depth and insulating the illegal communication data flows. Therefore, our approach can pre-

vent the malicious operations to Modbus slave, and ensure Modbus slave operating regularly.

In order to illustrate the rule granularity comparison between our approach and the one in [27], we load its Modbus/TCP filtering module into the Linux Iptables/Netfilter system according to the mentioned procedures. Without loss of generality, we implement this approach in one device whose hardware resources are the same with IDD, and we also deploy it between one Modbus master and one Modbus slave. The main rules of this experiment can be described as follows:

White-listing rule

iptables -A *FORWARD* -p *tcp* -s *192.168.1.101* -d *192.168.1.4* -m *modbus* --funcode *16* --refnum *10* --allowtcp *1* -j *ACCEPT*

Default rule

iptables -P *FORWARD DROP*

Here, the parameter "funcode" represents the allowed function code, and the parameter "refnum" represents the allowed register address. Besides, we also suppose that the normal values which can be written to the 10th register range from 80 to 120.

We use modpoll to execute a normal write operation to change the register of Modbus slave, and this operation conforms to the above white-listing rule. As shown in Fig. 11a, the value 95 is successfully written to the 10th register, and modpoll presents "Writing 1 reference.". That is, the communication packets of this operation are forwarded to Modbus slave without any difficulty, and the operation is executed smoothly. However, when we use modpoll to execute an abnormal write operation which changes the register value to 1000 by using the same function code and register address. Figure 11b shows that this operation is also executed successfully, and modpoll presents the same result "Writing 1 reference.". In other words, it obviously proves that the malicious communication packets are still forwarded to Modbus slave and this abnormal operation cannot be filtered effectively. In addition, we also repeat the experiments of Fig. 10c and d, and the responding results are the same with these two figures. As seen from the above experiments, although the approach in [27] can protect Modbus slave from the illegal access to some extent, there are still some disadvantages: On the one hand, this approach cannot restrict the register value, and lacks the ability to drop the illegal access with the anomalous value. So, the control system may be damaged because Modbus slave receives too large or too small values. On the other hand, the white-listing rule in this approach cannot offer the address range setting, and in each rule there is only one address. So, if we set too many rules, the execution efficiency may be affected seriously. To sum up, our approach has the finer rule granularity, and is superior to the one in [27].

(a)

```
C:\Users\Administrator>D:\modpoll\win32\modpoll.exe -m tcp -t 4 -r 10 192.168.1.
4 95
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration...: address = 1, start reference = 10, count = 1
Communication.........: 192.168.1.4, port 502, t/o 1.00 s, poll rate 1000 ms
Data type.............: 16-bit register, output (holding) register table

Written 1 reference.
```

(b)

```
C:\Users\Administrator>D:\modpoll\win32\modpoll.exe -m tcp -t 4 -r 10 192.168.1.
4 1000
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration...: address = 1, start reference = 10, count = 1
Communication.........: 192.168.1.4, port 502, t/o 1.00 s, poll rate 1000 ms
Data type.............: 16-bit register, output (holding) register table

Written 1 reference.
```

Fig. 11 Modpoll's execution result under different values. **a** Modpoll's execution result when the value is 95, **b** modpoll's execution result when the value is 1000

4.2 Simulation experiment and analysis

We evaluate our approach in detail by establishing a real attack and defense simulation experiment, and our main purpose is to discuss the availability and effectiveness of its defense in depth. In the experiment, we build a small control system based on Modbus/TCP. As shown in Fig. 12, the experiment environment is composed of three layers: the supervisory control layer includes two Modbus masters (one operator workstation and one attacker), and the operator workstation can not only monitor the actual process states but also change technological operation; the control unit layer

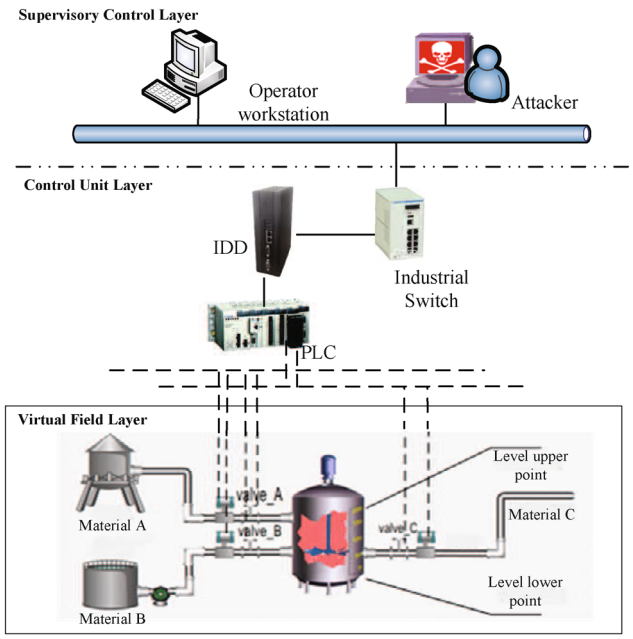

Fig. 12 Experimental environment topology

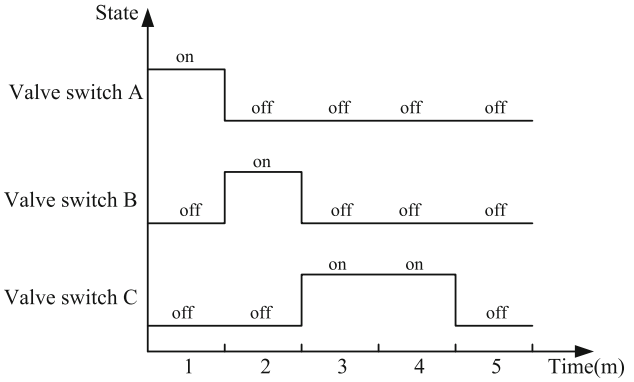

Fig. 13 On-off states of three valve switches in one technological process

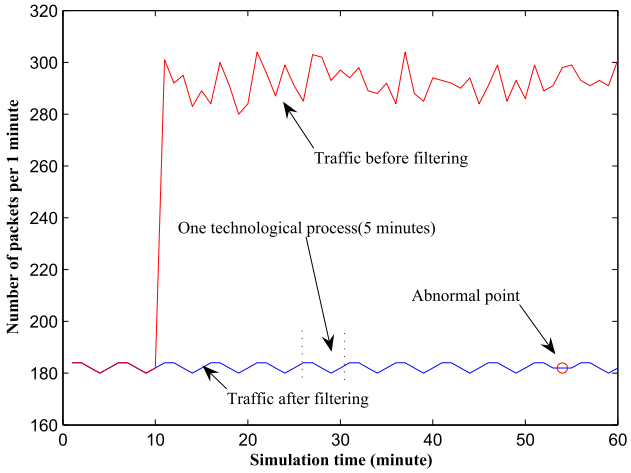

Fig. 14 Traffic comparison before and after IDD's filtering

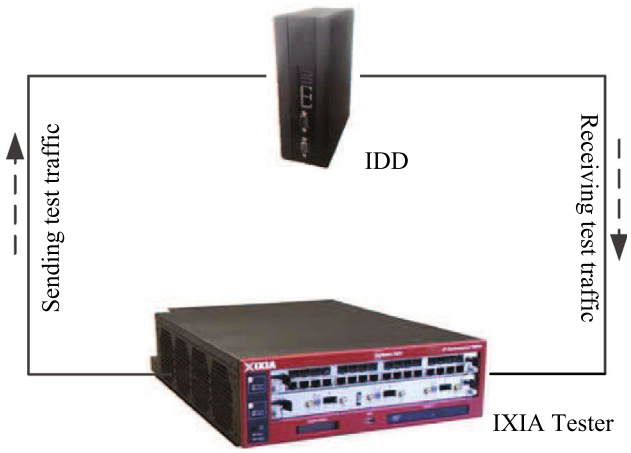

Fig. 15 Performance test topology

includes one PLC, one IDD and an industrial switch. Besides, IDD is deployed between the PLC and the switch, and the PLC accepts the control commands from the operator workstation to control the field actuators; the virtual field layer includes three switches and some liquid level sensors, and these three switches represent two inlet valve switches and one outlet valve switch.

The whole technological process can be simply depicted as follows: when the valve switches A and B are respectively turned on, materials A and B successively flow into the container through the valve switches A and B to produce material C. When material C in the container reaches the level upper point, the valve switches A and B are turned off, and then the valve switch C is turned on. When material C in the container exhausts and reaches the level lower point, the valve switch C is turned off. Besides, the above-described technological process is repeatedly performed every 5 min, and the on-off states of three valve switches in one technological process are depicted in Fig. 13.

In this experiment, we control the valve switches by writing the corresponding coil, and each valve switch is mapped to a coil address. In addition, we suppose that the attacker cannot know the specific control parameters of this technological process, and just blindly sends the attack packets to execute a write single coil operation whose function code is 05. Briefly, the ultimate aim of the attacker is to destroy the workflow of the virtual field devices. Furthermore, we also assume that the attack rate ranges from 100 to 120 packets per minute. Here, each attack packet only contains one random coil address, and the corresponding value is written to 1. We start the attack at 11th minute, and the white-listing rules are set to pass the packets which just have 01 and 05 function codes. Figure 14 shows the traffic comparison before and after IDD's filtering when the attack happening. As seen in this figure, almost all attack packets are dropped by IDD, and the filtered traffic present periodic variations, namely the technological process is repeatedly performed every 5 min.

However, it is worth noting that an abnormal point appears in the filtered traffic. That is because an attack packet which conforms to the white-listing rules is generated accidentally by using random coil addresses, causing the devices to go screwy. Nevertheless, this is an extreme case, and it will not happen in the actual control system, because IDD can timely generate an alert when the attack is just beginning. Besides, the probability of this case is extremely small, and is only $(1/2^{16}) \times (4/5 + 4/5 + 3/5)$. With the increasing complexity of the technological process, the probability will become smaller.

4.3 Performance evaluation

In order to analyze IDD's performance indicators, this paper uses IXIA Optixia XM2 IP Performance Tester to perform the basic performance test under different numbers of rules. Figure 15 shows the performance test topology.

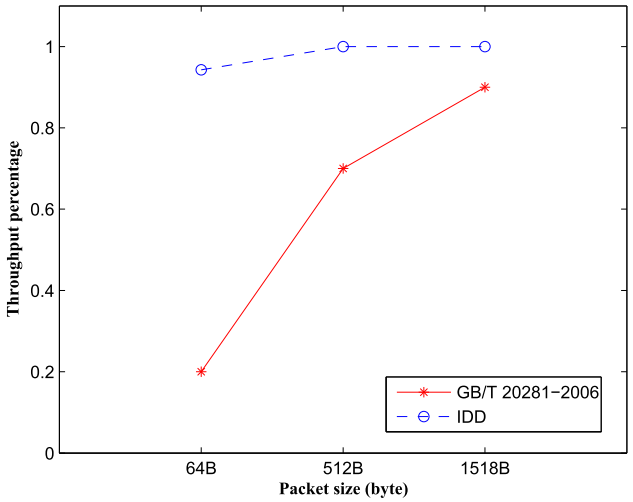

Fig. 16 Throughput percentage comparison between IDD and GB/T 20281-2006

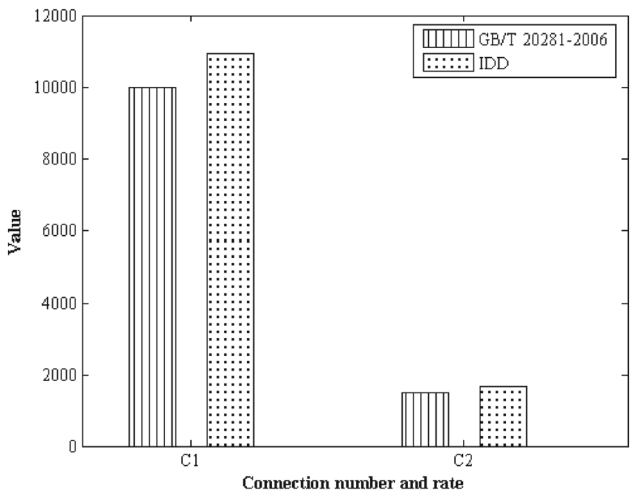

Fig. 17 Maximum number of concurrent connections and maximum connection rate comparison between IDD and GB/T 20281-2006

The performance test includes the following performance indicators: throughput, latency, packet loss rate, maximum number of concurrent connections and maximum connection rate. For a start, we perform the throughput tests under different packet sizes. The test time is 60 s, and the actual test traffic speed is 200 Mbps (bidirectional, 100 Mbps in each direction). Furthermore, we perform three tests for every packet size, and the test results can be concluded as follows: for 512- and 1518-byte packets, both of the throughputs reach 100 %; for 64-byte packet, the corresponding throughput reaches 94.3 %. Figure 16 shows the throughput percentage comparison between IDD's test results and the first-level technique requirements of the throughput in [29]. From this figure we can see that, IDD's throughputs always beyond the corresponding technique requirements under different packet sizes. On this basis, we also perform the tests about the latency and the packet loss rate, and the actual test traffic speed is exactly the same as the above one. However, the difference is that these tests are completed under one rule and 200 rules, respectively. As illustrated in Table 1, we find that the packet loss rate is always 0 whether under one rule or 200 rules, and the forwarding latency is as low as hundreds of microseconds. In practice, in order to meet the availability requirements in the networked control system, the throughput only needs to reach several tens of megabits per second, and the latency has to be less than a few milliseconds or tens of milliseconds. In particular, the maximum number of concurrent connections reaches 10946, and this means that IDD can simultaneously support more than ten thousand connections from different Modbus masters or OPC clients. Besides, the maximum connection rate exceeds 1650 per second, that is, IDD can forward more than 1650 connection requests per second from different Modbus masters or OPC clients. Figure 17 describes the maximum number of concurrent connections and maximum connection rate comparison between IDD's test results and the relevant technique requirements of 100 M firewall in [29]. Here, C1 and C2 represent the maximum number of concurrent connections and the maximum connection rate, respectively. As shown in this figure, IDD's test results are slightly better. From the above results, IDD not only can provide security defense for the networked control system, but also can satisfy its network communication requirements.

5 Demo platform and prototype system

Aiming at illustrating the positive effect of our approach to defend the key industrial devices (such as PLCs) with various industrial viruses, we build a demo platform by using the

Table 1 Basic performance test under different rule numbers

Test condition		1 rule		200 rules	
Packet size (B)	Test traffic speed (Mbps)	Packet loss rate	Latency (μs)	Packet loss rate	Latency (μs)
64	200	0	342.60	0	192.76
512	200	0	699.22	0	901.28
1518	200	0	891.82	0	884.03

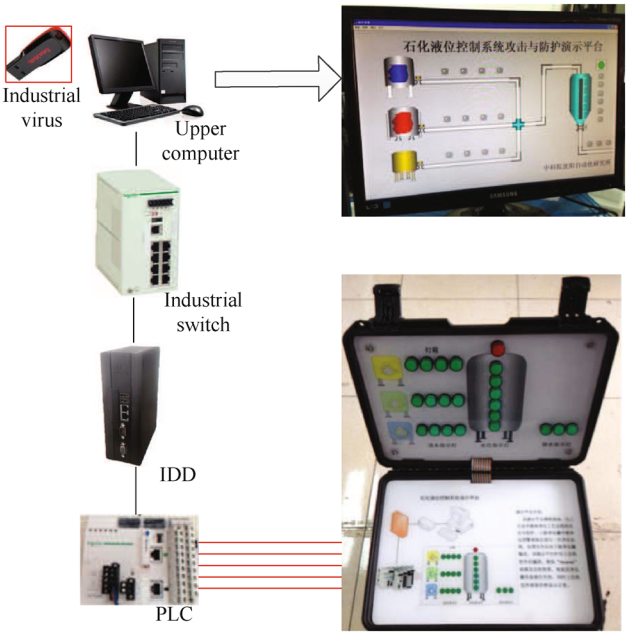

Fig. 18 Attack and defense demo platform for petrochemical level control

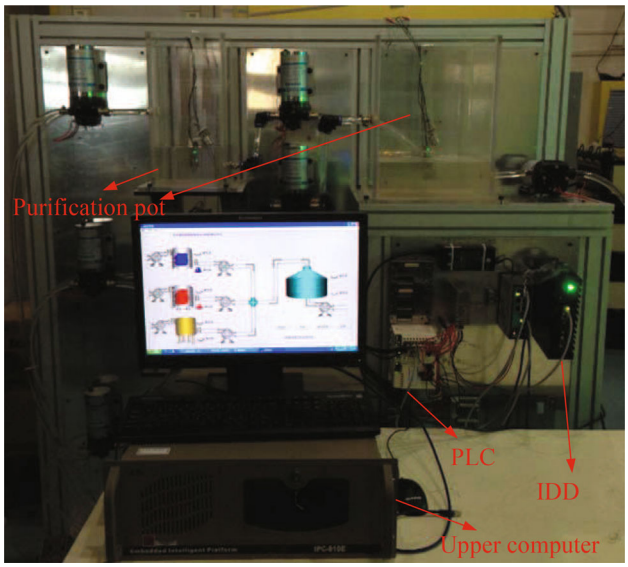

Fig. 19 Attack and defense prototype system for petrochemical level control

light box to simulate attack and defense in the petrochemical level control system. As shown in Fig. 18, this platform demonstrates the level control process of liquid purification. By virtue of the buffer overflow vulnerability in the configurable software (the main reason for this vulnerability is that the configurable software uses non-validated user input to write the buffer, and the input exceeds the length of the buffer to cause the buffer overflow), the industrial virus first damages the configurable software's stack, and then makes the execution pointer point to and run a malicious piece of code. Furthermore, the behaviors of the malicious code mainly include two parts: the first one is to keep the monitor screen displaying the normal process, so that the attack cannot be perceived by the engineer or operator; the second one is to send a malicious Modbus/TCP packet to execute an abnormal write operation, which makes the liquid in three sub pots flow into the main pot simultaneously and causes an explosion. The whole attack operation can be summarized below: to begin with, when we insert the USB disk with an industrial virus into upper computer, the industrial virus can actively save the malicious code in the specified memory segment because of the AutoRun function in Windows XP. Next, with the help of the buffer overflow vulnerability in the configurable software, industrial virus can execute the malicious code and carry out sabotage. Finally, industrial virus not only sends the malicious Modbus/TCP control command to the PLC, but also shows the normal states in the upper computer. In conclusion, because of the excessive liquid in the main pot, the undetectable and significant accident is going to occur. However, when we deploy IDD between the upper computer and the PLC, if we insert the same USB disk into the upper computer again, we can find that the level control system can run normally and stably, and after a period of time there is no unusual behavior in the PLC.

Moreover, based on the demo platform, we also build a prototype petrochemical level control system to show IDD's defense capability, as shown in Fig. 19. In this system, we successfully fight against not only the fore-mentioned industrial virus but also various DoS (denial of service) attacks. In summary, both the demo platform and the prototype system can prove the effectiveness and feasibility of our approach.

6 Practical application

At present, IDD has been successfully applied in several real-world petrochemical control systems. As depicted in Fig. 20, the software Aspen serves as OPC client, and communicates with OPC server to acquire the information of the field instruments, such as temperature and pressure. Our IDD is deployed between Aspen and OPC server, and we only set one white-listing rule which allows Aspen to access OPC server. By the field test, the throughput reaches 40Mbps, and the delay is much less than 1ms. In particular, IDD can correctly identify Aspen's OPC access request, and provide security protection for OPC server. To be more precise, on the one hand, IDD can effectively stop the malicious access of the unauthorized client and protect data from destructing or stealing. On the other hand, IDD can avoid exposing the unused and open ports, and prevent the spread of industrial viruses to some extent. Through applications in the real-world system,

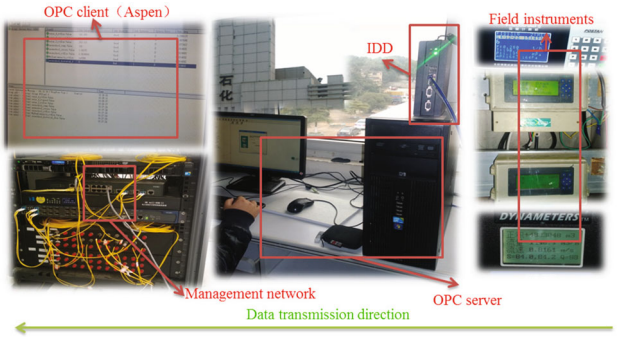

Fig. 20 IDD's application in real-world control systems

it shows that IDD can commendably reduce the number of the unauthorized or erroneous alerts.

7 Conclusions

This paper aims to propose a content-based deep communication control approach for the networked control system, and the basic idea behind the proposed approach is very simple. That is, implementing the access control by parsing industrial communication protocols in depth. Briefly, our approach mainly support two industrial communication protocols: for OPC Classic, this approach can open a dynamic port for the authorized OPC client; for Modbus/TCP, this approach can check the key control information between Modbus master and Modbus slave. In this paper, we first put forward basic layer architecture of the networked control system. And then, we present the detailed design of the proposed approach, including basic model design, OPC deep parsing and tracking and Modbus/TCP deep parsing and matching. At last, we evaluate our approach in detail by some simulation experiments and performance tests. We show that, our approach is comparatively advantageous and suitable for security defense in the networked control system. In addition, based on IDD, we also build a demo platform and a prototype system to simulate attack and defense, and successfully apply IDD in several real-world petrochemical control systems.

Acknowledgements This work is supported by the National Natural Science Foundation of China (Grant Nos. 61501447, 61502474) and Independent project of Key Laboratory of Networked Control System Chinese Academy of Sciences: Research on abnormal behavior modeling, online intrusion detection and self-learning method in industrial control network. The authors are grateful to the anonymous referees for their insightful comments and suggestions.

References

1. Gupta, R. A., & Chow, M. Y. (2010). Networked control system: Overview and research trends. *IEEE Transactions on Industrial Electronics*, *57*(7), 2527–2535.
2. Kagermann, H., Wahlster, W., & Helbig, J. (2014). Recommendations for implementing the strategic initiative INDUSTRIE 4.0. Final Report. http://wwwplattform-i40.de/finalreport2013.
3. Genge, B., Siaterlis, C., Fovino, I. N., & Masera, M. (2012). A cyber-physical experimentation environment for the security analysis of networked industrial control systems. *Computer and Electrical Engineering*, *38*(5), 1146–1161.
4. Zhang, H., Cheng, P., Shi, L., & Chen, J. (2016). Optimal DoS attack scheduling in wireless networked control system. *IEEE Transactions on Control Systems Technology*, *24*(3), 843–852.
5. Lin, S., & Wu, H. (2015). Bloom filter-based secure data forwarding in large ccale cyber-physical systems. *Mathematical Problems in Engineering*, *2015*(1), 1–10.
6. ICS-CERT. (2015). ICS-CERT year in review 2014. https://ics-cert.us-cert.gov/sites/default/files/Annual_Reports/Year_in_Review_FY2014_Final.
7. Nourian, A., & Madnick, S. (2015). A systems theoretic approach to the security threats in cyber physical systems applied to stuxnet. *IEEE Transactions on Dependable and Secure Computing*, *99*, 1–20.
8. Hadziosmanovic, D., Bolzoni, D., Etalle, S., & Hartel, P. (2012). Challenges and opportunities in securing industrial control systems. *Proceedings of 2012 Complexity in Engineering (COMPENG12)* (pp. 1–6).
9. Davis, K. R., Davis, C. M., Zonouz, S. A., Bobba, R. B., Berthier, R., Garcia, L., et al. (2015). A cyber-physical modeling and assessment framework for power grid infrastructures. *IEEE Transactions on Smart Grid*, *6*(5), 2464–2475.
10. Yeole, A. S., & Meshram, B. B. (2011). Analysis of different technique for detection of SQL injection. *Proceedings of 2011 International Conference & Workshop on Emerging Trends in Technology (ICWET11)* (pp. 963–966).
11. Stouffer, K., Falco, J., & Scarfone, K. (2011). Guide to industrial control systems (ics) security. National Institute of Standards and Technology (NIST), US Department of Commerce, Technical Report NIST Special Publication (pp. 800–82). http://csrc.nist.gov/publications/nistpubs/800-82/SP800-82-final.
12. Papp, D., Ma, Z., & Buttyan, L. (2015). Embedded systems security: threats, vulnerabilities, and attack taxonomy. *Proceedings of 2015 13th Annual Conference on Privacy, Security and Trust (PST15)* (pp. 145–152).
13. Beresford, D. (2011). Exploiting siemens simatic S7 PLCs. https://media.blackhat.com/bh-us-11/Beresford/BHUS11BeresfordS7PLCsWP.
14. Darias, Z., Serhrouchni, A., & Vogel, O. (2015). Taxonomy of attacks on industrial controls protocols. *Proceedings of 2015 International Conference on Protocol Engineering (ICPE15) and New Technologies of Distributed Systems (NTDS15)* (pp. 1–6).
15. Zhao, W., Xie, F., Peng, Y., & Gao, Y. (2013). Security testing methods and techniques of industrial control devices. *Proceedings of 2013 9th International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP13)* (pp. 433–436).
16. Voyiatzis, A. G., Katsigiannis, K., & Koubias, S. (2015). A Modbus/TCP fuzzer for testing internetworked industrial systems. *Proceedings of 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA15)* (pp. 1–6).
17. Igure, V. M., Laughter, S. A., & Williams, R. D. (2006). Security issues in SCADA networks. *Computers and Security*, *25*(7), 498–506.
18. Cereia, M., Bertolotti, I. C., Durante, L., & Valenzano, A. (2014). Latency evaluation of a firewall for industrial networks based on the Tofino industrial security solution. *Proceedings of 2014 IEEE Emerging Technology and Factory Automation (ETFA14)* (pp. 1–8).

19. Zhang, S. S., Shang, W. L., Wan, M., Zhang, H., & Zeng, P. (2014). Security defense module of Modbus TCP communication based on region/enclave rules. *Computer Engineering and Design*, *35*(11), 3701–3707.
20. Krotofil, M., & Gollmann, D. (2013). Industrial control systems security: what is happening? *Proceedings of 2013 11th IEEE International Conference on Industrial Informatics (INDIN13)* (pp. 670–675).
21. Tan, V. V., Yoo, D. S., & Yi, M. J. (2007). Security in automation and control systems based on OPC techniques. *Proceedings of 2007 International Forum on Strategic Technology (IFOST07)* (pp. 136–140).
22. Schwarz, M. H., & Borcsok J. (2013). A survey on OPC and OPC-UA: about the standard, developments and investigations. *Proceedings of 2013 XXIV International Symposium on Information, Communication and Automation Technologies (ICAT13)* (pp. 1–6).
23. OPC Foundation. (2000). The OPC security custom interface specification. http://opcfoundation.org/.
24. Wan, M., Shang, W. L., Zeng, P., & Zhao, J. M. (2016). Modbus/TCP communication control method based on deep function code inspection. *Information and Control*, *45*(2), 248–256.
25. Shang, F. J., Pan, Y. J., Pan, X. Z., & Bin, B. (2008). Research on a stochastic distribution multibit Trie tree IP classification algorithm. *Journal on Communications*, *29*(7), 109–117.
26. Jiang, W., & Prasanna, V. K. (2013). Data structure optimization for power-efficient IP lookup architectures. *IEEE Transactions on Computers*, *62*(11), 2169–2182.
27. Pothamsetty, V., & Franz, M. (2004). Transparent Modbus/TCP filtering with Linux. http://modbusfw.sourceforge.net/.
28. Igor, N. F., Alessio, C., Andrea, C., & Masera, M. (2012). Critical state-based filtering system for securing SCADA network protocols. *IEEE Transactions on Industrial Eletronics*, *59*(10), 3943–3950.
29. GB/T 20281-2006. (2006). Information security technology-technique requirements and testing and evaluation approaches for firewall products. National Standard of the People's Republic of China. http://www.spc.org.cn/gb168/.

Ming Wan is currently an associate professor in Key Laboratory of Networked Control System of Shenyang Institute of Automation Chinese Academy of Sciences. Before that, he received the B.S. degree from Beijing Jiaotong University in Jul. 2007, and received the Ph.D. degree in Communication and Information System from National Engineering Laboratory for Next Generation Internet Interconnection Devices of Beijing Jiaotong University in Jan. 2013. His research interests include the areas of architecture of future Internet, network and information security and industrial control network security.

Wenli Shang is currently an associate professor and master tutor in Key Laboratory of Networked Control System of Shenyang Institute of Automation Chinese Academy of Sciences. Before that, he received the Ph.D. degree from the Shenyang Institute of Automation Chinese Academy of Sciences in 2005. His research interests include the areas of industrial control system network security, machine learning.

Linghe Kong is currently an associate professor in Department of Computer Science and Engineering at Shanghai Jiao Tong University. Before that, he was a postdoctoral researcher at McGill University from 2014 to 2015 and a postdoctoral researcher at Singapore University of Technology and Design in 2013. He received his Ph.D. degree in computer science from Shanghai Jiao Tong University, China, 2012, his Master degree in Telecommunication from TELECOM SudParis (ex. INT), France, 2007, and his B.E. degree in Automation from Xidian University, China, 2005. He was also a joint Ph.D. student at University of California, San Diego, 2011, and a visiting researcher in Microsoft Research Asia, 2010. His research interests include wireless communication, sensor networks, mobile computing, Internet of things, and smart energy systems.

Peng Zeng is now a professor and the director of the Laboratory of Industrial Control Network and System, Shenyang Institute of Automation, Chinese Academy of Sciences. He also holds the positions of Member, Expert Group of IEC TC65 WG16. Before that, he received his Ph.D. degree in mechatronic from Shenyang Institute of Automation Chinese Academy of Sciences in 2005. His research interests include industrial automation and the wireless networks for such automation.