Efficient Rule Engine for Smart Building Systems

Yan Sun, Tin-Yu Wu, Guotao Zhao, and Mohsen Guizani, *Fellow, IEEE*

Abstract—In smart building systems, the automatic control of devices relies on matching the sensed environment information to customized rules. With the development of wireless sensor and actuator networks (WSANs), low-cost and self-organized wireless sensors and actuators can enhance smart building systems, but produce abundant sensing data. Therefore, a rule engine with ability of efficient rule matching is the foundation of WSANs based smart building systems. However, traditional rule engines mainly focus on the complex processing mechanism and omit the amount of sensing data, which are not suitable for large scale WSANs based smart building systems. To address these issues, we build an efficient rule engine. Specifically, we design an atomic event extraction module for extracting atomic event from data messages, and then build a β-network to acquire the atomic conditions for parsing the atomic trigger events. Taking the atomic trigger events as the key set of MPHF, we construct the minimal perfect hash table which can filter the majority of the unused atomic event with $O(1)$ time overhead. Moreover, a rule engine adaption scheme is proposed to minimize the rule matching overhead. We implement the proposed rule engine in a practical smart building system. The experimental results show that the rule engine can perform efficiently and flexibly with high data throughput and large rule set.

Index Terms—Smart building system, rule engine, rule matching, minimal perfect hash function.

1 Introduction

With the development of the wireless sensor and actuator networks (WSANs), smart building systems have been extensively studied in recent years [1], [2]. The primary objective of such system is to control electric appliances intelligently according to the environmental information collected by sensors for energy conservation in buildings. The smart control process is usually performed according to certain rules. The rules triggered by events can be expressed as the form of condition-action. For example, a rule can be described as "when someone works in the office with dim light, the corresponding lamp is turned on automatically". In a smart building system, rule engine is an important component that can provide flexible control. The essence of a rule engine subsystem is to separate logics and data, so as to make logics as independent and maintainable parts.

In a smart building system, detected environment data may be sound, image, temperature, smoke/gas concentration, humidity, etc. Sensors around a certain monitoring region collect environmental data and report them to the server within a regular sampling period. The server analyzes and processes the data for identifying events, matching the rules, and executing the corresponding actions. The events are often sudden environmental changes such as sound, light, fire (temperature, smoke concentration) and surface vibration. Generally, the frequency of reporting data is far greater than that of generating events. In order to filter a great deal of redundant data and improve the efficiency and accuracy of event generation, we design an effective event preprocessing mechanism according to static properties of the data itself (e.g., a geographical position, type of node, etc.). Take the rule—if Temperature $> 60^{\circ}$C, then an alarm sound—for example, we can filter the data from two aspects: 1) we filter the data reported by all the sensors except temperature sensors according to the type of node; 2) we filter the data that is collected beyond related monitoring region by the geographical position. In this way, the real-time performance of event generation is improved. With the development of smart building systems, the rapid expansion of events and rules cause the rule engine encounter two main problems: how to filter plenty of meaningless events and how to improve rule matching efficiency. In this paper, we consider dynamic factors (e.g., time and combinational conditions) to further promote the operation efficiency of rules. Considering that many rules are triggered by conditions which are composed of several events instead of a single one, it is crucial to design an efficient rule matching mechanism to promote the real-time performance of rule engine [3].

Many current business rule engines (CLIPS [4], JESS [5], DROOLS [6], BizTalk [7], etc.) are employed to provide better flexibility and reduce the cost of designing, developing and delivering software. The traditional algorithms, including the RETE for rule engine [8], [9], [10], mainly focus on the complex processing mechanism of rule engine with a large rule set and limited event throughput. However, in WSANs based smart building systems, thousands of deployed sensors and actuators produce abundant data. As shown in Fig. 1, in a WASNs based smart building system, many kinds of sensors are deployed for collecting environmental information, each electric device is equipped with an actuator for receiving control commands, and each user

• *Y. Sun is with the Beijing Key Lab of Intelligent Telecommunication Software and Multimedia, Beijing University of Posts and Telecommunication, Beijing 100876, China. E-mail: sunyan@bupt.edu.cn.*
• *T.-Y. Wu is with the Department of Computer Science and Information Engineering, National Ilan University, Taiwan, R.O.C. E-mail: tyw@niu.edu.tw.*
• *G. Zhao is with the Networked Computing and Intelligent Systems, IBM China Research Laboratory, Beijing 100876, China. E-mail: guotaozh@cn.ibm.com.*
• *M. Guizani is with the Qatar University, Doha, Qatar. E-mail: mguizani@ieee.org.*

Manuscript received 12 Sept. 2013; revised 6 July 2014; accepted 16 July 2014. Date of publication 4 Aug. 2014; date of current version 13 May 2015. Recommended for acceptance by Y. D. Lin.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2014.2345385

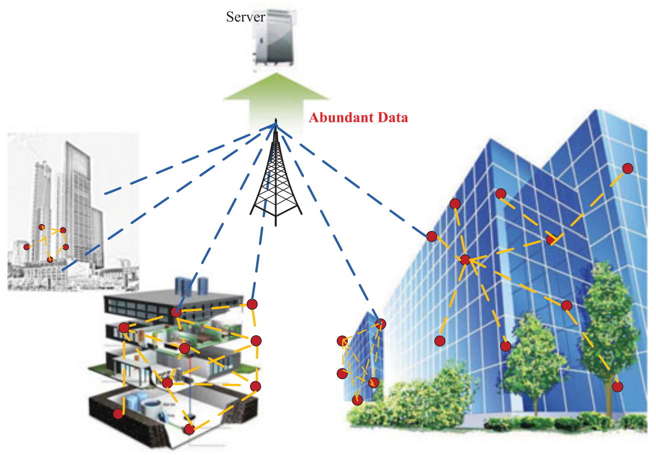


Fig. 1. The smart building system based on wireless sensors and actuator networks.

subscribes multiple rules to customize the required services. As a result, there are lots of events contributing to a large scale of rule set. In addition, many urgent events generated in smart building systems often have real-time response requirements. Existing rule engines mainly focus on traditional business scope and omit the problem of data load. Moreover, these engines are generally too heavy and complex to handle plenty of events, thus cannot be applied to a smart building system directly. On the other hand, traditional algorithms including the RETE cannot guarantee the quick matching between plenty of events and rules, and thus are not suitable for the system with lots of subscribed rules and produced events.

In this paper, aiming at large-scale smart building systems, we propose an efficient rule engine with high data load and large rule set, which can match events and execute rules in real time. First, by analyzing the features of data in a smart building system, we find that although the reported data is abundant, the execution frequency of triggered rules is relatively low. By filtering the unnecessarily processed data in time, we realize an efficient rule engine. In addition, with the increase of the scale of rule set, the rule conditions become more complex and rule executions are more frequently triggered. Hence, the performance of the rule engine can be further promoted by adjusting rule execution schemes dynamically according to the current system states.

Our main contributions can be summarized as follows.

- For the large scale smart building system containing abundant events and rules, we design a high-efficient rule engine for quick matching between events and rules and rule execution.
- We construct a minimal perfect hash table based on MPHF, in which the key set is composed of all the atomic trigger events. As an effective filtering table, the minimal perfect hash table discards the majority of unnecessarily processed data with only $O(1)$ time overhead. Our proposed engine adaption scheme, based on the rule matching feedback, can significantly reduce the rule matching overhead adaptively.
- We implement the proposed rule engine, and further verify it by a real smart building system. The experimental results show that our solution improve the performance of rule execution even with overwhelming data and large rule set.

The remainder of this paper is organized as follows. We discuss the related work in Section 2. Section 3 describes the preliminaries. The proposed efficient rule engine is detailed in Section 4. Section 5 analyzes the operational complexity of our solution. In Section 6, we give our experimental study and simulations. We conclude the paper in Section 7.

2 Related Work

In the field of rule-based systems, extensive research has been carried out on the rule processing scheme [11], [12], In the terms of rule engine, they mainly consist of two aspects: the RETE algorithms and the complex event processing mechanism.

RETE is a classic algorithm for the rule engine, which has been proposed by Forgy in [8] and [13]. The RETE algorithms were first employed in production systems [9], [14], [15]. Recently, many researchers have paid more attention to the algorithm implementation and improvement for some specific applications [16], [17], [18], [19], [20]. In [16], the authors improved the RETE algorithm with a matching scheme, which could rapidly reflect changes in the E-business and make the system dynamic and efficient. The authors in [17] proposed an extension of RETE networks, which was capable of handling a general inferential process. It includes several types of schemes for reasoning with imperfect information. In [18], to solve the security policy implementation efficiency problem of a network information system, the authors proposed an improved object-oriented RETE algorithm and a novel network structure model. In [19], to solve the problem of the RETE algorithm in the aspects of performance and flexibility, the authors applied three methods to improve the RETE algorithm in the rule engine: rule decomposition, Alpha-Node-Hashing, and Beta-Node-Indexing. In [20], by employing the mechanisms of nodes' sharing, types' preprocessing, and index-based searching optimization, the authors proposed an improved version of the RETE algorithm, IRETE, which was tested under multi-entity and multi-rule circumstances to be a much more efficient matching algorithm. However, the RETE algorithm was mainly designed for the task of pattern matching. RETE networks are limited to operations such as unification and the extraction of predicates from a knowledge base. For a smart building system with data stream processing, the traditional RETE algorithm cannot be used directly.

With the development of the service-oriented and event-driven architecture in WSANs, the rule engine is usually required to handle complex correlation rules with logical, temporal, content-based, and other operators. Complex event processing technology has been introduced. In [10], the authors proposed an extension of the RETE algorithm to support temporal operators using interval time semantics and presented the issues created by this extension as well as the pursued methodology. In [21], the authors also extended the RETE algorithm to detect relative temporal constraints.

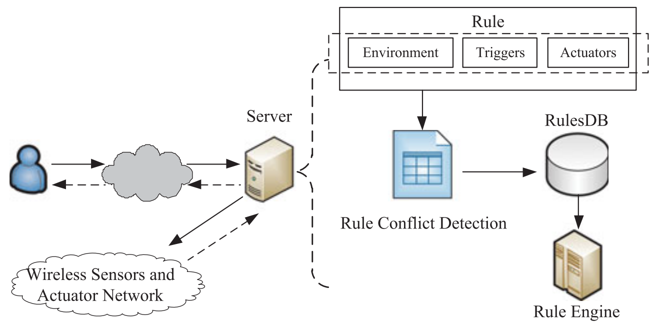


Fig. 2. Rule system.

They proposed an efficient method to perform the garbage collection in the RETE algorithm in order to discard events after they cannot fulfill their temporal constraints any more. In [22], to support the expression of time-sensitive patterns, the authors proposed an extension of the RETE through the concepts of time-stamped data and temporal constraints between reported data, which allows applications to write rules that process both facts and events. In [23], the authors presented the design, implementation, and evaluation of a system that can execute complex event queries over real-time streams of RFID readings encoded as events. The complex event queries filter and correlate events to match specific patterns, and transform the relevant events into new composite events for the use of external monitoring applications.

In all the research works mentioned above, the complex event processing schemes are usually introduced by integrating the event processing with the RETE. Nevertheless, they are too heavy and complex to satisfy the requirement of quick response. In our previous work, the call home analysis and response system (CHARS) [24] utilized neighborhood, composition, and association relationships between various network elements and software-based services to perform root cause analysis on collected failure messages. Thus, the system can correlate network and service logs and events to identify the root causes behind failures. However, the CHARS system focused on the process of the rule match without a quick rule matching scheme. In this paper, by analyzing the features of smart building systems, we propose several optimization approaches to build an efficient rule engine subsystem.

3 Preliminaries

In this section, we first introduce the rule system in our smart building platform, and then summarize the minimal perfect hash function (MPHF) which is used to develop our rule engine.

3.1 Rule System

Users access the smart building system through customizing their services. The rule system can convert these services into corresponding rules and detect conflicts among these rules, as illustrated in Fig. 2. If a rule conflicts with another existing rule in the database, this rule will not be executed and this conflict will be reported to the user. Otherwise, this rule will be stored into the database.

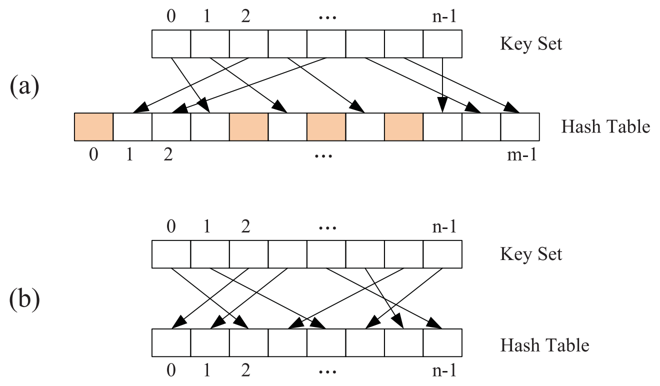


Fig. 3. Comparison between the perfect hash function and MPHF [28]: (a) Perfect hash function; (b) MPHF.

When events reported from sensors match a rule, the rule will be sent to the rule running set and executed by the rule engine.

To provide smart services, the rule execution engine is an indispensable component in a smart building system. Rule conflict detection is essential to ensure the correctness of rule execution. For rule conflict verification, most existing studies have focused on storage structure of service and their conflict detection algorithms [25], [26]. In our previous work [27], we have proposed a probability analysis method to assess the possibility of the conflicts and anomalies of rules to solve the problem.

On the other hand, in a smart building system, plenty of sensors and actuators are deployed to sample environmental information and control electric devices, which produce abundant events. Handling all these events needs expensive computation. Besides, urgent events usually require real-time response. Motivated by this, we design an efficient rule engine. In this rule engine, we construct a filtering table with the minimal perfect hash function to filter the majority of meaningless events and propose a rule engine adaption scheme to reduce the rule matching overhead greatly.

3.2 MPHF

A perfect hash function maps a static set of n keys into a set of m integer numbers without collisions, where m is no less than n. If $m = n$, the function is called the minimal one. A perfect hash function and a minimal perfect hash function are given in Figs. 3a and 3b, respectively. Minimal perfect hash functions have been widely used for memory storage and fast retrieval of items from static sets. In this paper, we get MPHF using the method proposed in [28].

The algorithm based on random graphs can construct minimal perfect hash function h. For a set of n keys, the algorithm outputs h in expected time $O(n)$. The evaluation of $h(x)$ requires two memory accesses for any key x and the description of h takes up $1.15n$ words. The core problem in the construction of a minimal perfect hash table is the selection of a key set in MPHF. In order to construct an efficient filtering table, we transform the rule set to a β-network for extracting the atomic conditions. Then, the atomic trigger events can be acquired according to a geographical position, the type of event, and the device ID. By taking the atomic trigger events as key set, we obtain the filtering table.

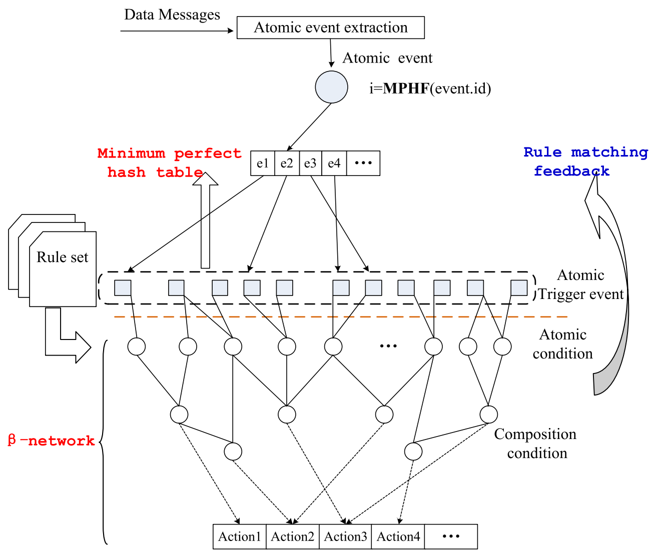


Fig. 4. The architecture of the proposed efficient rule engine.

4 Efficient Rule Engine

As shown in Fig. 4, the proposed efficient rule engine mainly consists of three components: atomic event extraction module for preprocessing abundant data, filtering table with a minimal perfect hash function, and dynamic adaption scheme with rule matching feedback.

In a WSANs based smart building system, a massive number of sensors and actuators is deployed for collecting the environmental data and controlling the electronic devices. To avoid data overload which causes network congestion and transmission delay, different types of data are reported in different ways. Hence, we design an atomic event extraction module. According to the rule set, we build a β-network to acquire the atomic conditions, from which the atomic trigger events can be parsed. Taking the atomic trigger events as the key set of MPHF, we construct the minimal perfect hash table which filters the majority of unused atomic events with $O(1)$ time overhead. On the other hand, the rule execution is usually triggered by several conditions jointly. We can dynamically adjust the time-window parameter of the minimum perfect hash table according to the rule matching results in the β-network. This adaption scheme can significantly reduce the rule matching overhead.

4.1 Atomic Event Extraction

In our smart building system, many sensors are deployed to capture discrete environmental data, such as temperature, humidity, illumination, and CO$_2$. Due to the small amount of data, the cycle report mode is commonly adopted in environment monitoring application and the cycle can be configured. For sensors which produce amount of discrete data, like low-power Bluetooth beacon, the differential report mode is used to reduce the data volume in the process of transmission. In some monitoring applications, sensors report their data using the threshold report mode, i.e., sensors will report the data once the sensed data exceeds a certain threshold. For audio and image sensors, the amount of data is even larger, so the event wake mode is utilized. For instance, when someone in a classroom is detected by an infrared sensor after lights out, audio and image sensors are turned on to collect data and compress these multimedia data for transmission. Hence, the data received by the rule engine may contain discrete data, differential data, and compressed media data. Comprehensive using of different system operation modes and data processing methods effectively avoids network congestion. To adapt to different data sources, we design an event extraction module for data separation, data recovery, and atomic event generation in the rule engine.

Definition 1 (Atomic Event). *An event that is triggered by only one type of data in one data message can be defined as an atomic event.*

Some sensor nodes can capture various kinds of data, so a data message may contain different types of data. The data preprocessing module first needs to separate data for atomic event abstraction. For example, in our smart building system, temperature, humidity, and light sensors are integrated on a wireless node. In this way, a data message includes three kinds of data and the respective data type identifiers. Different types of data need to be separated, and then these separated data will be recovered according to different report modes. For data by differential report, the current recovery data is calculated by the sum of difference received and the previous recovery data. Compressed media data (e.g., voice and image) is recovered through the corresponding compression/ decompression algorithms. The simple type of data from a simple sensor with the geographical property can be defined as atomic event. The event.*id* of an atomic event includes the information fields of geographical position, event type, and device ID. For example, in our smart building system, the device ID is the MAC address of a Zigbee node, the event type can be audio, temperature, humility, and light, etc., the geographical position can include a building, a floor, and a room number.

4.2 Filtering Table with MPHF

4.2.1 β-Network

During the construction of a filtering table, we first preprocess each rule and transform the rule set into a β-network. The rule can be usually expressed with the formula like $x_1 \wedge x_2 \wedge (x_3 \vee (y_1 \wedge y_2)) \rightarrow A_1$, where x_i, y_i denote the conditions and A_i represents the action. We know that each propositional formula can be converted into an equivalent formula in disjunctive normal form (DNF), for example, $(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge \ldots \wedge (x_n \vee y_n)$ is equal to $(x_1 \wedge \ldots \wedge x_{n-1} \wedge x_n) \vee (x_1 \wedge \ldots \wedge x_{n-1} \wedge y_n) \vee \ldots \vee (y_1 \wedge \ldots \wedge y_{n-1} \wedge y_n)$. Through the preprocessing, the rule $x_1 \wedge x_2 \wedge (x_3 \vee (y_1 \wedge y_2)) \rightarrow A_1$ is transferred as follows:

$$\begin{gathered} x_1 \wedge x_2 \wedge (x_3 \vee (y_1 \wedge y_2)) \rightarrow A_1 \\ \Downarrow \\ (x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge y_1 \wedge y_2) \rightarrow A_1 \\ \Downarrow \\ (x_1 \wedge x_2 \wedge x_3) \rightarrow A_1 \\ (x_1 \wedge x_2 \wedge y_1 \wedge y_2) \rightarrow A_1. \end{gathered}$$

The rule $x_1 \wedge x_2 \wedge (x_3 \vee (y_1 \wedge y_2)) \rightarrow A_1$ is decomposed into two atomic rules.

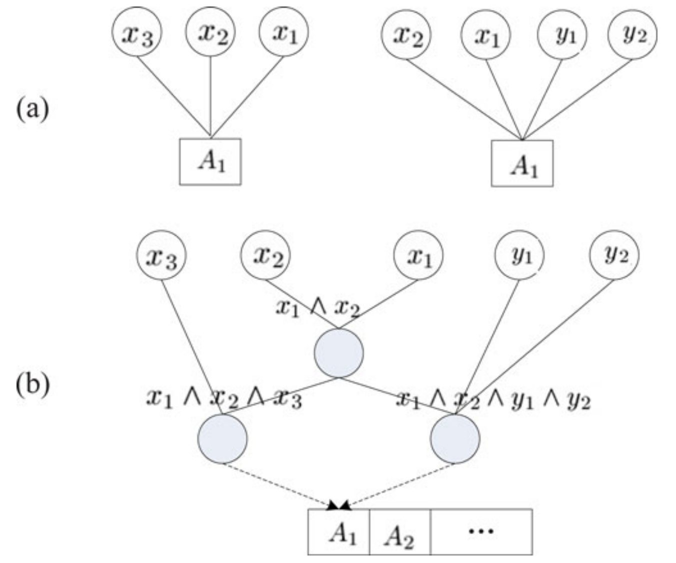


Fig. 5. Transformation from a rule set to a β-network: (a) The tree format of the atomic rule; (b) The β-network.

Definition 2 (Atomic Rule). *The action is triggered by several conditions jointly. The atomic rule can be executed only when the conditions are satisfied simultaneously. In other words, if only the* and *operation* $\wedge$ *exists among the conditions, the rule is an atomic rule.*

Based on the DNF transformation, we decompose a complex rule into multiple atomic ones, which provides a foundation for the rule engine optimization. According to the features of DNF, we get the following property.

Property 1. *In the process of rule execution, the atomic rules are mutually independent. The atomic rules can be parsed in parallel.*

Then, based on atomic rules in a rule set, we construct a β-network. Each rule is first converted into a tree. The action is the root and the atomic conditions are the leaf nodes. We optimize the rule tree, transform the atomic conditions to the composition conditions, and thus construct the β-network by the multiplexing principle.

Definition 3 (Atomic Condition). *The atomic condition is a basic component of an atomic rule which is not able to be divided any more.*

Definition 4 (Composition Condition). *The composition condition contains one or more atomic conditions and zero or multiple composition conditions. The composition condition can lead to the final action. Let* C_i, a_i *denote the composition condition and atomic condition, respectively. The composition condition can be represented with a regular expression* $(a_i)(\wedge a_i)*(\wedge C_i)*$.

We take an example to describe the transformation from rule set to β-network. As illustrated in Fig. 5, atomic rules of $(x_1 \wedge x_2 \wedge x_3) \rightarrow A_1$ and $(x_1 \wedge x_2 \wedge y_1 \wedge y_2) \rightarrow A_1$ are converted into two rule trees, respectively. Then, $x_1 \wedge x_2$ is extracted as the common composition condition. The β-network is composed of three parts: atomic condition, composition condition and action. In the β-network, the atomic condition acts as a device node, the composition condition acts as a router and the action represents the destination of the rule. We can deduce that the device nodes of the β-network contain all the atomic conditions in the rule set.

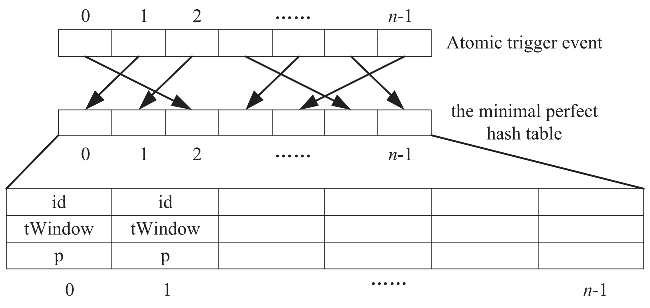


Fig. 6. The architecture of the minimal perfect hash table.

4.2.2 Minimal Perfect Hash Table

Next, by employing the atomic conditions of a β-network, we design an efficient filtering table which can filter useless data with the time complexity of $O(1)$. In a specific system, the atomic condition usually consists of one or more events. For example, the atomic condition *average temperature is greater than 25°C* means that we should integrate the events from multiple temperature sensors in the same period. By analyzing the atomic conditions and the event source property in the system, we can build an adaptor that defines the correspondence between atomic conditions and atomic trigger events. As we mentioned above, in a β-network, there only exists the logic operation *and* among the atomic conditions. An atomic condition can be triggered by multiple atomic trigger events, and an atomic trigger event can trigger multiple atomic conditions.

Definition 5 (Atomic Trigger Event). *An event parsed from atomic conditions according to the geographical position, event type, and device ID. This event can be defined as an atomic trigger event. One or more atomic trigger events can be integrated to build up one atomic condition.*

The atomic trigger event is extracted from the atomic conditions by different types of events, geographical position, and device ID. That is, an event can be generated from a simple type of data reported from a sensor or actuator node. With the atomic trigger events parsed from atomic conditions, we construct the minimal perfect hash table using MPHF proposed in [28]. If the rule set changes and the atomic trigger events change correspondingly, the minimal perfect hash table should be reconstructed due to the key set which is composed of all atomic trigger events.

The architecture of the minimal perfect hash table (H) is shown in Fig. 6. Each table item consists of three fields: *id*, *tWindow* and *p*. The field of *id* is used to identify an atomic trigger event, which can be the integration of geographical position, event type, and device ID. The field of *tWindow* denotes a time period, in which the specific event is valid. The field of *p* is a pointer, which points to the atomic trigger event in a designed adaptor.

As shown in Fig. 4, when an atomic event *e* arrives, we compute the index, *i*, of this atomic event by *MPHF(e.id)*, and then get the corresponding item, *H(i)*, from the hash table. If *H(i).id* is not equal to *e.id*, the event is then regarded as a useless one and discarded from the rule engine. Otherwise, we need to analyze whether the event occurrence time is valid by the field of *H(i).tWindow*. The *tWindow* field can be updated dynamically. The updating method will be described in the next section. We present the description of the event filtering algorithm in Algorithm 1.

Algorithm 1. Filtering Algorithm

```
1:  Construct MPHF with atomic trigger events.
2:  Construct the minimum perfect hash table.
3:  while 1 do
4:    if rule set is updated then
5:      Update the MPHF and filtering table.
6:    end if
7:    Receive an atomic event e.
8:    i=MPHF(e.id).
9:    if e.id==H(i).id && e.time is not in H(i).tWindow
      then
10:     Trigger the rule engine with e.p.
11:   end if
12:   Check to update the tWindow field in the filtering
      table.
13: end while
```

The proposed filtering algorithm with the minimal perfect hash table can detect meaningless data messages with $O(1)$ time cost and small amount of storage. From the experiments, we find that more than 88 percent of useless messages can be filtered out. As mentioned above, the minimal perfect hash table needs to be reconstructed as the rule set is updated. Fortunately, on one hand, the rule set needs not to be updated frequently. On the other hand, the experiment results show that when there are one million atomic trigger events in a rule set, the average time of the MPHF construction is about $6.1 \pm 0.3\,\mathrm{s}$, which can usually be ignored in the context of a smart building system.

4.3 Dynamic Adaption Scheme with Rule Matching Feedback

Most of the meaningless data messages are discarded by the proposed filtering table. The detected events are important components of the atomic condition. Because rule executions are usually triggered by multiple conditions jointly, once one condition fails the transformation of other atomic conditions in the β-network will be meaningless. This implies that the detected events seldomly arrive at the destination in a β-network. Therefore, detecting and stopping useless transformation instantly can improve the performance of the rule engine. Towards this end, we propose a dynamic adaption scheme with rule matching feedback.

In a smart building system, sensors report data periodically. We treat an event, happening in one reporting period, as an invalid one for the other reporting periods. The adaption scheme is performed mainly based on the time period. As shown in Fig. 4, when one condition fails, we can get the effective time period of the failed condition. Then, we search the atomic trigger events affected by the failed condition and send a rule matching feedback for updating the *tWindow* field of those events in the filtering table. This way, the useless transformation of corresponding atomic conditions is stopped at the filtering table and the rule matching overhead can be minimized adaptively. The detailed adaption scheme is given in Algorithm 2.

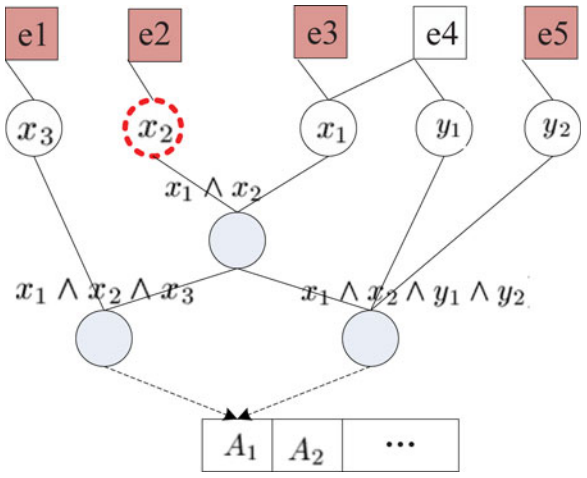

e1
e2
e3
e4
e5
x_3
x_2
x_1
y_1
y_2
$x_1 \wedge x_2$
$x_1 \wedge x_2 \wedge x_3$
$x_1 \wedge x_2 \wedge y_1 \wedge y_2$
A_1
A_2
...

Fig. 7. The algorithm of searching the atomic events affected by the failed atomic condition.

Algorithm 2. Dynamic Adaption Scheme

```
1:  Receive an atomic event.
2:  Transform to corresponding atomic conditions.
3:  for each corresponding atomic conditions do
4:    if the condition fails then
5:      Extract the effective time range.
6:      Search atomic events affected by failed condition.
7:      Send the rule matching feedback.
8:      Continue;
9:    end if
10:   Transform to the composition condition.
11: end for
```

In the proposed adaption scheme, the algorithm of searching atomic trigger events is one of the most important components. The atomic trigger events are affected by the failed atomic condition. Starting from the failed atomic condition, we traverse the whole β-network and find the corresponding affected atomic trigger events. If an event has the same ancestor nodes with the failed atomic condition, this event is regarded as the corresponding affected atomic trigger event. Next, we take an example to illustrate the searching algorithm.

As shown in Fig. 7, when the atomic condition x_2 cannot be satisfied by the event e_2, we traverse its ancestor nodes in the β-network. The composition node $x_1 \wedge x_2$ is the only parent node of x_1, and x_1 is the only parent node of e_3. Therefore, e_3 will be regarded as one of the target atomic trigger events. The event e_4 is excluded because it has two parents (x_1 and y_1) and y_1 is not affected by the failed conditions. Similarly, for the composition nodes $x_1 \wedge x_2 \wedge x_3$ and $x_1 \wedge x_2 \wedge y_1 \wedge y_2$, e_1 and e_5 are also considered as the target atomic trigger events. We give the detailed searching scheme in Algorithm 3.

Algorithm 3. Searching Algorithm

```
1:  Get the failed atomic condition $c$.
2:  $p = c$.parent
3:  while $p$ is not action node do
4:      for each children, $x_i$, of the node $p$ do
5:          if $p$ is the only parent of $x_i$ then
6:              Push $x_i$ into the set $S$.
7:          end if
8:      end for
9:      for each atomic trigger event $e_i$ do
10:         if all the parent nodes of of $e_i$ are in the set $S$
            then
11:             $e_i$ is affected.
12:         end if
13:     end for
14:     $p = p$.parent
15: end while
```

With the expansion of the smart building system, large amount of data will be reported and the rule set will be increased. From the experiment described in Section 6, we find that when the amount of events and scale of rule set are small, the processing performance of our proposed rule engine is better than that of the RETE algorithm and the traditional scheme. With the increase of the amount of events, the superiority of our solution is more significant. In addition, the time cost of our solution is much lower than that of two others with the increase in the size of rule set.

5 Operational Complexity

In this section, we analyze the computational complexity of the proposed rule engine and compare it with the traditional scheme of sequential rule executing through rule base and the RETE algorithm. Let N denote the number of rules in a rule set and n represent the number of atomic conditions. We first evaluate the time and space complexity of the proposed scheme. The time complexity evaluation includes two aspects: event filtering and rule matching. Let C_t denote the time complexity, we can get the equation:

$$C_t = O(1) + O(\mu\ log_2\ n), \tag{1}$$

where $0 < \mu < 1$ is a dynamic factor.

The event filtering is performed with the hash table. We know that the searching performance of the hash table is the best. So, the filtering can be done with $O(1)$ time cost. For the rule matching, we need to traverse a tree with all the atomic conditions. Leaf nodes in the tree consist of all the atomic conditions. The time cost of traversing a tree is O ($log_2\ n$) [29]. However, as we described in Section 3.2, once a dynamic adaption scheme is performed, many data messages can also be filtered by the time window. The time cost of the rule matching can be represented with $O(\mu\ log_2\ n)$. The dynamic factor μ represents the influence of the rule matching feedback.

Also, we evaluate the space cost of the proposed scheme, which includes the filtering table and the β-network. For the filtering table, it only needs n elements to store the atomic conditions [28]. For the β-network which is the tree-similar structure, the space cost is $2n - 1 + N$. Hence, the space complexity can be evaluated with

$$C_s \approx [n + (2n - 1 + N)] \times A, \tag{2}$$

where A represents a constant memory cost for each element (atomic event, atomic condition and action).

Next, we analyze the performance of the RETE algorithm. As presented in [16], the time and space complexities are:

$$C_t = O(m) + O(log_2\ n), \tag{3}$$

$$C_s \approx [m + (2n - 1 + N)] \times A, \tag{4}$$

where m denotes the number of static conditions in the rule set.

In the RETE algorithm, the number of static conditions decides the time cost of the α-network traversing. At the same time, there is no event filtering in the α-network. Compared to the proposed scheme, the RETE algorithm spends more time on the event filtering and rule matching. The difference of the space cost, by contrast, is tiny.

Finally, we analyze the traditional parsing scheme of sequential rule executing through rule base, and get the time and space complexities as follows:

$$C_t = O(m + n), \tag{5}$$

$$C_s \approx N \times B, \tag{6}$$

where B denotes an approximate memory cost of one rule.

For each reported data message, the traditional scheme will traverse all the rules, which means much more time cost. All the rules in the rule set are maintained in the memory.

From the above performance analysis, it is obvious that our scheme performs better than the traditional scheme and the RETE algorithm.

6 Experiments

To evaluate the performance of the proposed scheme, we implemented it on a practical smart building platform [2].

6.1 Platform Introduction

Aiming at providing convenient and comfortable living environment with less energy consumption, we deploy a WSAN based smart building system in one of our school buildings, as shown in Fig. 8. Sensor nodes have the ability of sensing multiple kinds of environmental information, e.g., temperature, humidity, light intensity, audio, and image information, each of which can be transferred to the router via multi-hop Zigbee network. A router automatically detects a Zigbee network, and joins/establishes it, so as to expand the scale of the network. The router plays the role of gateway which transfers messages from a Zigbee network to a WiFi network, and reports the sensing information to the server. After receiving data from routers, the server analyzes this data by user-defined rules. Once these rules are matched, the server will send control commands to the corresponding actuator(s). With the expansion of the quantity of events and rules, the rule system will encounter problems, including rule conflicts and low operation

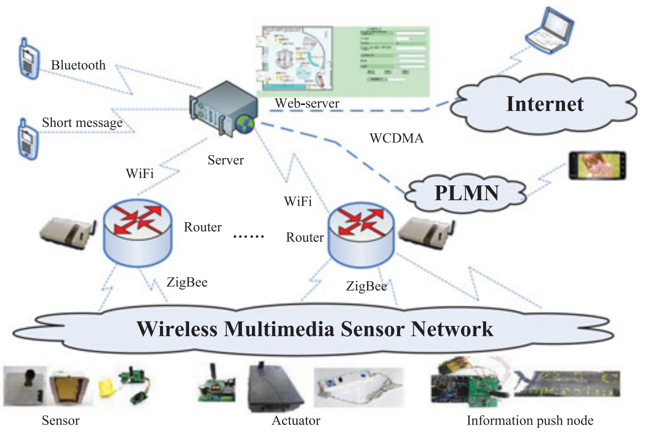


Fig. 8. The architecture of a smart building system [2].

efficiency. For this reason, the rule engine is deployed in the server to ensure the effective operation of the system.

Specifically, this platform consists of 200 temperature sensors, 160 light sensors, 100 humidity sensors, 40 audio sensors, 20 image sensors and 200 actuators. As shown in Table 1, the sensors and actuators are designed based on STM32F103 processing chip and CC2530 RF; the router is designed with the AT91SAM7X256 processing chip, CC2530 RF and WiFi module; the server consists of Intel Core2 Duo P8400, 2 GB memory, WiFi module and 100M Ethernet card.

In real application scenarios, there often exist some sensor nodes that sense the same targets/events simultaneously, i.e., many messages may include the same event signal. On the other hand, the information sensed by sensor nodes is sometimes inaccurate because of sensing components failure or external interference. Therefore, in this platform, we also exploit a fault-tolerant data aggregation framework proposed in our previous work [30]. As an aggregator, the router fuses the data reported from sensor nodes according to the spatial and temporal correlation, and calculates the trustworthiness of the aggregated result. Then, the transmission messages from routers to the server are the aggregated results. This framework is leveraged to not only reduce the throughput of data transmission, thus saving energy effectively, but also to reduce the impact of erroneous data and provide measurable trustworthiness for aggregated results. More details can be found in [30].

6.2 Rule Set

In the system, rules are created by users on web pages or terminal devices. The rule format and the webpage should be concise as much as possible, so as to help non-professional users easily customize the rules. Fig. 9 shows the webpage of creating rules. The left part demonstrates the deployment of all the sensors and actuators in a room. The right part illustrates the interface that users add, modify, and delete rules. We can browse the circumstances of all the rooms and configure the rules to control the building appliances. As an example, Fig. 10 shows the rule list in room No. 902 on the ninth floor of No.3 teaching building. Users can easily view all the rules and the details of each rule.

We use XML format as the rule-represent data structure for facilitating the rule matching. With the self-explaining tags, we formalize all information about one rule, including: the static attributions (control area and runtime), dynamic services (service content, trigger event, and action information), optional functions (whether to notify user, notification method), and other user related information. In order to execute rules easily, we extract the relevant fields from the XML format and transform into usable format in rule database. An example is shown in Fig. 11.

The smart building system has run nearly two years in our campus environment, and has stored more than 30,000 rules in the rule database. The following case study is based on the platform.

TABLE 1
Experimental Environment Specification

Device	Number	CPU	Transfer mode	Transmission Rate
node	720	STM32F103	ZigBee	250 Kbps
router	30	AT91SAM7X256	ZigBee/WiFi	250K/11 Mbps
server	1	Intel Core2 Duo P8400	WiFi / Ethernet	11M/100 Mbps

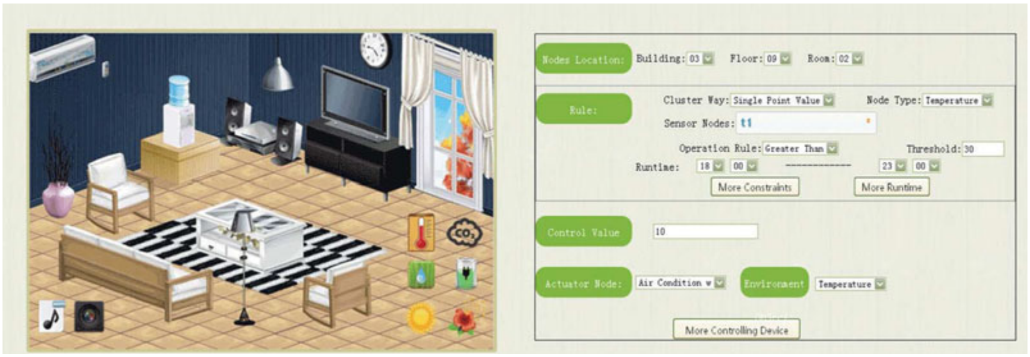
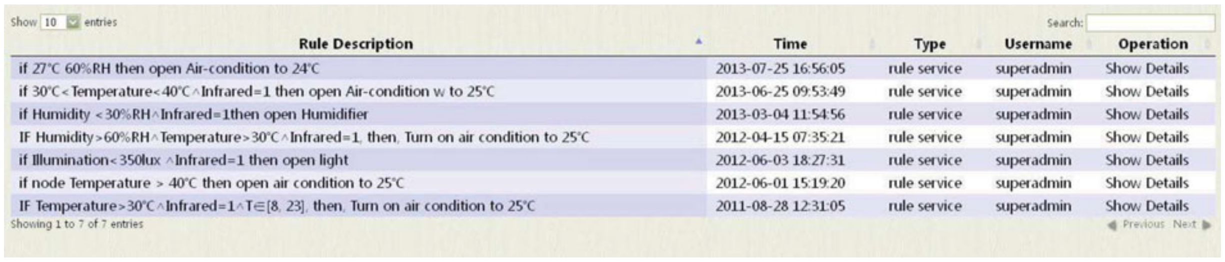

Nodes Location: Building: 03 Floor: 09 Room: 02
Rule: Cluster Way: Single Point Value Node Type: Temperature
Sensor Nodes: t1
Operation Rule: Greater Than Threshold: 30
Runtime: 18 00 ------------ 23 00
Control Value 10
Actuator Node: Air Condition Environment Temperature

Fig. 9. Rule creating diagram.


Show 10 entries Search:

Rule Description	Time	Type	Username	Operation
if 27°C 60%RH then open Air-condition to 24°C	2013-07-25 16:56:05	rule service	superadmin	Show Details
if 30°C<Temperature<40°C∧Infrared=1 then open Air-condition w to 25°C	2013-06-25 09:53:49	rule service	superadmin	Show Details
if Humidity <30%RH∧Infrared=1then open Humidifier	2013-03-04 11:54:56	rule service	superadmin	Show Details
IF Humidity>60%RH∧Temperature>30°C∧Infrared=1, then, Turn on air condition to 25°C	2012-04-15 07:35:21	rule service	superadmin	Show Details
if Illumination<350lux ∧Infrared=1 then open light	2012-06-03 18:27:31	rule service	superadmin	Show Details
if node Temperature > 40°C then open air condition to 25°C	2012-06-01 15:19:20	rule service	superadmin	Show Details
IF Temperature>30°C∧Infrared=1∧T∈[8, 23], then, Turn on air condition to 25°C	2011-08-28 12:31:05	rule service	superadmin	Show Details

Showing 1 to 7 of 7 entries

Fig. 10. Rule list.

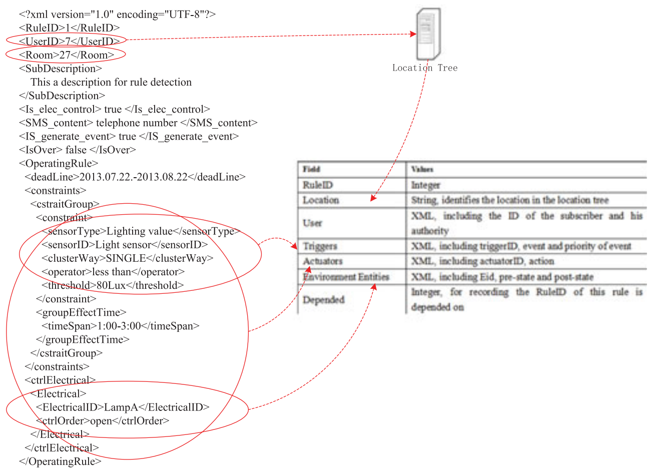
6.3 Case Study for Dynamic Adaption Scheme

We put forward two rules as an example to demonstrate the dynamic adaption scheme in execution. Suppose:

R_1 Humidity $>$ 60%RH $\wedge$ Temperature $>$ 30°C $\wedge$ Infrared=1 $\rightarrow$ Turn on air condition to 25°C

R_2 Temperature $>$ 30°C $\wedge$ Infrared=1 $\wedge$ Time $\in$ [8, 23] $\rightarrow$ Turn on air condition to 25°C.

As shown in Fig. 12, we transform the rules to a β-network. Both R_1 and R_2 are composed of three atomic conditions and an action, in which Temperature $>$ 30°C $\wedge$ Infrared=1 is extracted as a common composition condition. We use C_1, C_2, C_3, C_4, C_5 to indicate the involved atomic conditions in the β-network, and e_1, e_2, e_3, e_4, e_5 to express the events corresponding to the conditions. If e_2 receives a temperature event of 28°C, which cannot match the C_2, it means condition matching of C_2 fails. According to the proposed dynamic adaption algorithm, starting from C_2, we traverse the whole β-network to look for ancestor nodes of C_2. Firstly, $C_2 \wedge C_3$ is found as the ancestor node of C_2, and is also the only ancestor node of infrared condition C_3, which is the only ancestor node of infrared event e_3. Therefore, e_3 is judged as the filtering object of the algorithm. However, e_4 is excluded since it has two ancestor nodes C_3 and C_4, and C_4 will not be affected by C_2. In the same way, we can find the upper level ancestor node of C_2 – $C_1 \wedge C_2 \wedge C_3$ and $C_2 \wedge C_3 \wedge C_4$, and get the target filtering events e_1 and e_5. This case demonstrates that when the matching of C_2 fails, the system will filter the corresponding events of relevant conditions C_1, C_3, and C_5 to prevent matching them. In this way, the execution time can be cut down and the system effectiveness can be enhanced.


```
<?xml version="1.0" encoding="UTF-8"?>
<RuleID>1</RuleID>
<UserID>7</UserID>
<Room>27</Room>
<SubDescription>
  This a description for rule detection
</SubDescription>
<Is_elec_control> true </Is_elec_control>
<SMS_content> telephone number </SMS_content>
<IS_generate_event> true </IS_generate_event>
<IsOver> false </IsOver>
<OperatingRule>
 <deadLine>2013.07.22.-2013.08.22</deadLine>
 <constraints>
  <cstraitGroup>
   <constraint>
    <sensorType>Lighting value</sensorType>
    <sensorID>Light sensor</sensorID>
    <clusterWay>SINGLE</clusterWay>
    <operator>less than</operator>
    <threshold>80Lux</threshold>
   </constraint>
   <groupEffectTime>
    <timeSpan>1:00-3:00</timeSpan>
   </groupEffectTime>
  </cstraitGroup>
 </constraints>
 <ctrlElectrical>
  <Electrical>
   <ElectricalID>LampA</ElectricalID>
   <ctrlOrder>open</ctrlOrder>
  </Electrical>
 </ctrlElectrical>
</OperatingRule>
```

Field	Values
RuleID	Integer
Location	String, identifies the location in the location tree
User	XML, including the ID of the subscriber and his authority
Triggers	XML, including triggerID, event and priority of event
Actuators	XML, including actuatorID, action
Environment Entities	XML, including Eid, pre-state and post-state
Depended	Integer, for recording the RuleID of this rule is depended on

Fig. 11. Example of a rule data structure.

6.4 Performance Evaluation

In our smart building system, during the normal operation process, we set the report cycle be 1 second for cycle report mode, i.e., the send rate of each sensor node is 1 message per second. In the threshold report mode, because the reported event is emergent, we set the send rate of each sensor node be 10 warning messages per second.

Based on a set of 10,000 rules, we perform the experiment with different data arrival rates, the number of messages which arrive at the server per second, to evaluate the performance of the proposed scheme. To set the data arrival rate flexibly, the amount of historical reported data is utilized to simulate the arrival rate for the server. We further compare our proposed scheme with the RETE algorithm proposed in [18] and the traditional scheme of sequential rule executing through rule base.

As shown in Fig. 13a, when the arrival rate is 2,000 messages per second, it can be handled by all the three schemes. For our proposed scheme, we receive and process all the reported data until the arrival rate increases to 21,000. The RETE algorithm can cope with about 15,000 of the arrival rate because the RETE algorithm spends more time on the event filtering. The traditional scheme which traverses all the rules for each event performs even worse and can only undertake about 3,700 of the arrival rate. Compared to the RETE algorithm and the traditional scheme, the rule engine performance with our proposed scheme is improved by about 40 percent and 467 percent, respectively. In Fig. 13b, we test the memory occupancy, which includes the space cost for the storage of the rule set and the buffer used. To describe the experimental results

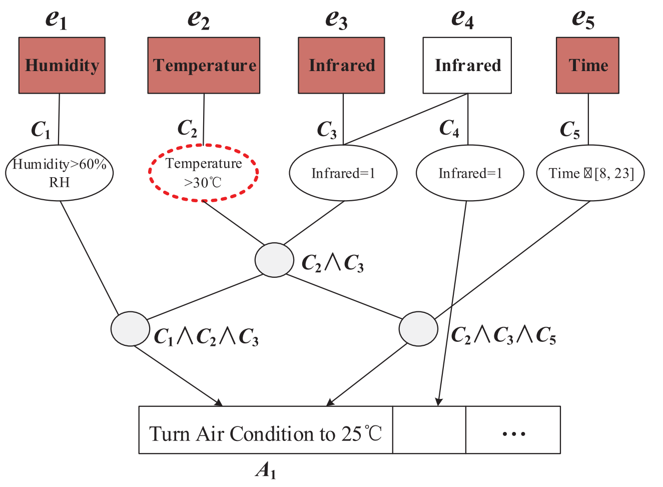
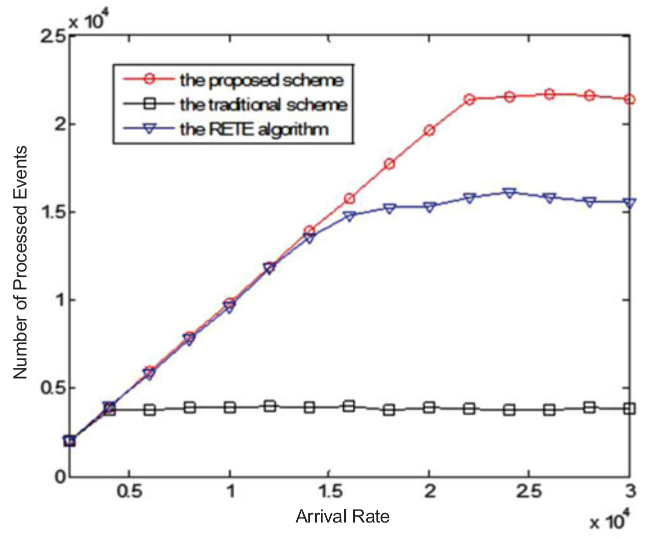
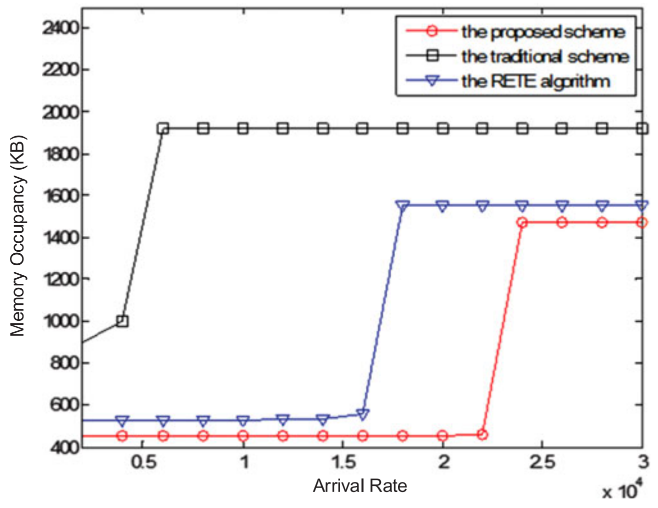


Fig. 12. The case of searching the atomic events affected by the failed atomic condition.

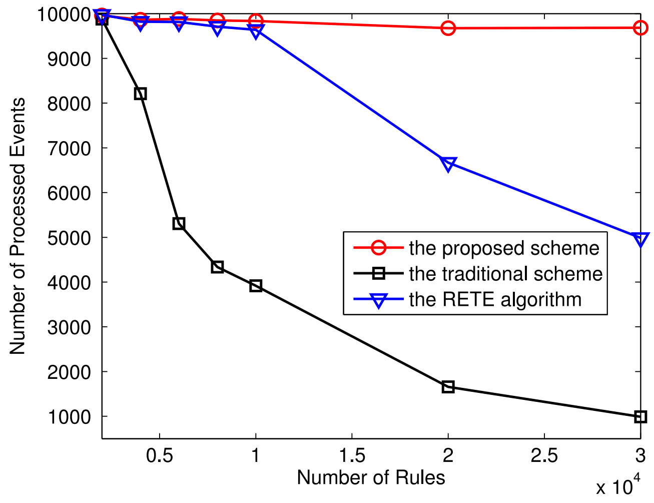


(a) Number of processed events

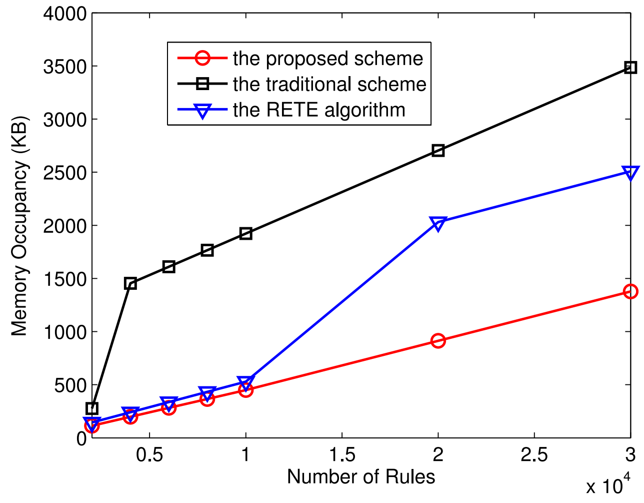


(b) Memory overhead

Fig. 13. The performance of the proposed scheme with different event arrival rate.



(a) Number of processed events



(b) Memory overhead

Fig. 14. The performance of the proposed scheme with different size of the rule set.

clearly, we set the maximum buffer size to 1 MB. For the proposed scheme, initially, the memory occupancy is mainly composed of a filtering table, a β-network and a little buffer cost. As the send rate increases to 21,000, the reported data cannot be processed in time. Therefore, the buffer is full and the memory occupancy increases drastically. The space cost of the RETE algorithm is similar to the proposed scheme. The traditional scheme spends more memory because it keeps the original rule set in the memory during the experiment.

Next, we perform the experiment with different sizes of the rule set. The data arrival rate is set to 10,000 messages per second. As shown in Fig. 14a, the size of the rule set has little influence on the performance of the proposed scheme. However, for the RETE algorithm and the traditional scheme, as the rule set size increases, the number of processed messages decreases greatly because we filter meaningless data with a minimum perfect hash table. Furthermore, in the rule matching component, atomic conditions are organized as the β-network and only a part of conditions are triggered by the corresponding atomic events. For the RETE algorithm, the increment of rule set will reduce the performance of the event filtering. For the traditional scheme, obviously, more rules shall be traversed for each message as the rule set increases. In Fig. 14b, we present the memory occupancy during the experiment. As the rule set increases, more space will be spent on the static storage.

6.5 Cost Evaluation

In this section, we explore the cost evaluation of the proposed scheme. We know that once the rule set changes, the β-network and the MPHF will be updated accordingly. We test the time cost of the β-network and MPHF reconstruction (T_β and T_{MPHF}) with different sizes of the rule set (n). As given in Table 2, when n is set to 1,000, the time spent on the β-network and MPHF reconstruction are about 25 and 30 ms, respectively. As the rule set size is set to 32 millions, T_β and T_{MPHF} approach 109.8 and 262.2 s, respectively.

TABLE 2
Cost Evaluation

n(millions)	0.01	0.1	1	2	4	8	16	32
T_β (s)	0.025	0.13	3.01	7.32	15.5	35.7	56.9	109.8
T_{MPHF} (s)	0.03±0.006	0.5±0.07	6.1±0.3	12.2±0.6	25.4±1.1	51.4±2.0	117.3±4.4	262.2±8.7

Although the time cost is a little high, it can be ignored in many practical applications because the size of the rule set is usually less than one million in a specific application. Moreover, the rule set does not update frequently.

We further evaluate the time cost of our proposed scheme with different sizes of rule set. In the experiment, the data arrival rate is set to 10,000 messages per second, as shown in Fig. 15. The time cost for the traditional scheme is about 0.75 s with 2,000 rules. When the size of rule set approaches 4,000, the time cost will be 1 s. This means that the traditional scheme cannot handle the current data reporting frequency (10,000 messages per second). For the RETE algorithm, the time cost increases with the increase of the size of rule set. This is because the α-network and β-network in the RETE algorithm will become more complex and time-consuming. However, the time cost of our proposed scheme is relatively low because the rules can be analyzed and transformed into atomic conditions and the minimum perfect hash table can be constructed to filer most of the useless events.

From the above experiments and simulations, we conclude that our proposed scheme can perform well in a practical smart building platform. Compared with the traditional scheme and the RETE algorithm, the proposed scheme significantly improves the performance of the rule engine.

7 Conclusion

In this paper, we proposed an efficient rule engine for smart building systems, which can guarantee real-time response to events and quick match between events and rules. First, we preprocessed the data reported from sensors and actuators to extract the atomic events. Then, we transformed the rule set to a β-network for acquiring the atomic trigger events which compose the key set of MPHF, and constructed the minimum perfect hash table to filter most of the meaningless atomic events. Based on the rule matching feedback, we further proposed a rule engine adaption scheme, which can decrease the rule matching overhead dynamically. Finally, we implemented the proposed rule engine and verified its effectiveness in our practical smart building system. A series of experimental results showed that the proposed scheme can improve the rule execution performance greatly even with abundant data and large rule set.

In our future work, we will study a distributed rule engine system, in which services are stored and executed by routers to avoid any system failure caused by server failures or network interruptions.

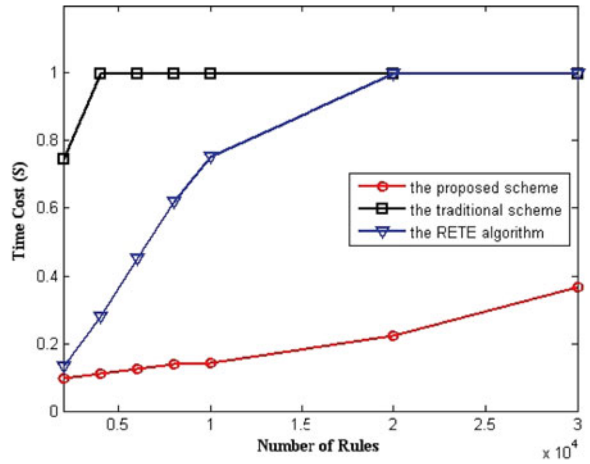


Fig. 15. The time cost comparison.

Acknowledgments

This work was partly supported by the National Key Technology Research and Development Program of the Ministry of Science and Technology of China under Grant No. 2014BAK14B03the National Natural Science Foundation of China under Grant 61272520, 61370196; the Research Fund for the Doctoral Program of Higher Education under Grant No. 20110005110007.

References

[1] Y. Agarwal, B. Balaji, and R. Gupta, "Occupancy-driven energy management for smart building automation," in *Proc. 2nd ACM Workshop Embedded Sensing Syst. Energy-Efficiency Building*, 2010, pp. 1–6.

[2] Y. Sun, G. Zhaoand H. Luo,"Smart building control based on wireless sensor-actuator networks," *Chinese J. Electron.*, vol. 20, no. 3, pp. 147–154, 2010.

[3] C.K. Liaskos and G.I. Papadimitriou, "Generalizing the square root rule for optimal periodic scheduling in push-based wireless environments," *IEEE Trans. Comput.*, vol. 62, no. 5, pp. 1044–1050, May 2013.

[4] R. Zhou, J. Pan, X. Tan and H. Xi, "Application of CLIPS expert system to malware detection system," in *Proc. Int. Conf. Comput. Intell. Security*, 2008, pp. 309–316.

[5] W. Chen, D. Ouyang and Y. Ye, "RIF2Jess: Inferencing RIF rules via translation to Jess rules," in *Proc. Int. Conf. Comput. Intell. Softw. Eng.*, 2009, pp. 1–6.

[6] M. Proctor, "Relational declarative programming with JBoss drools," in *Proc. Int. Symp. Symbolic Numeric Algorithms Sci. Comput.*, 2007, pp. 21–27.

[7] C. Herring and Z. Milosevic, "Implementing B2B contracts using biztalk," in *Proc. 34th Annu. Hawaii Int. Conf. Syst. Sci.*, 2001, pp. 211–218.

[8] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," *Artif. Intell.*, vol. 19, pp. 1021–1028, 1982.

[9] D. Batory, "The leaps algorithms," in Tech. Rep. FCS-94-28, University of Texs at Austin, Austin, TX, USA, pp. 102-108, 1994.

[10] K. Walzer, A. Schill and A. Loser, "Temporal constraints for rule-based event processing," in *Proc. ACM First Ph.D. Workshop CIKM*, 2007, pp. 176-188.

[11] M. Yoon, S. Chen and Z. Zhang, "Minimizing the maximum firewall rule set in a network with multiple firewalls," *IEEE Trans. Comput.*, vol. 59, no. 2, pp. 218–230, Feb. 2010.

[12] O. Rottenstreich, R. Cohen, D. Raz and L. Keslassy, "Exact worst case TCAM rule expansion," *IEEE Trans. Comput.*, vol. 62, no. 6, pp. 1127–1140, Jun. 2013.

[13] C. L. Forgy, "On the efficient implementation of production systems," Ph.D. dissertation, Dept. Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, pp. 102–108, 1979.

[14] D. P. Miranker, "TREAT: A better match algorithm for AI production system matching," in *Proc. 6th Nat. Conf. Artif. Intell.*, 1987, pp. 10–18.

[15] E. N. Hanson and M. S. Hasan, "Gator: An optimized discrimination network for active database rule condition testing," Tech. Rep. TR-93-036, CIS Dept., Univ. Florida, Gainesville, FL, USA, pp. 16–22, 1993.

[16] Z. Ren and D. Wang, "The improvement research on rule matching algorithm RETE in electronic commerce application systems," in *Proc. Int. Conf. Wireless Commun., Netw. Mobile Comput.*, 2008, pp. 1–4.

[17] D. Sottara, P. Mello, and M. Proctor, "A configurable RETE-OO engine for reasoning with different types of imperfect information," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 11, pp. 1535–1548, Nov. 2010.

[18] C. Tang and Y. Xie, "An improved object-oriented rete algorithm and network structure model," in *Proc. Int. Symp. Inf. Eng. Electron. Commerce*, 2010, pp. 1–4.

[19] D. Liu, T. Gu and J. Xue, "Rule engine based on improvement rete algorithm," in *Proc. Int. Conf. Apperceiving Comput. Intell. Anal.*, 2010, pp. 346–349.

[20] P. Yang, Y. Yang, and N. Wang, "IRETE: An improved RETE multi-entity match algorithm," in *Proc. Int. Conf. Electron., Commun. Control*, 2011, pp. 4363–4366.

[21] K. Walzer, T. Breddin, and M. Groch, "Relative temporal constraints in the RETE algorithm for complex event detection," in *Proc. ACM 2nd Int. Conf. Distrib. Event-Based Syst.*, 2008, pp. 170–178.

[22] B. Berstel, "Extending the RETE algorithm for event management," in *Proc. Int. Symp. Temporal Representation Reasoning*, 2002, pp. 10–16.

[23] E, Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams," in *Proc. ACM SIGMOD*, 2006, pp. 407–418.

[24] A. Al-Fuqaha, A. Rayes, M. Guizani, M. Khanvilkar, and M. Ahmed, "Intelligent service monitoring and support," in *Proc. Int. Conf. Commun.*, 2009, pp. 1–6.

[25] M. Nakamura, K. Ikegami, and S. Matsumoto, "Considering impacts and requirements for better understanding of environment interactions in home network services," *Comput. Netw.*, vol. 57, no. 12, pp. 2442–2453, 2013.

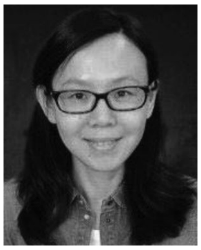
[26] C. Maternaghan and K. J. Turner, "Policy conflicts in home automation," *Comput. Netw.*, vol. 57, no. 12, pp. 2429–2441, 2013.

[27] H. Luo, R. Wang, and X. Li, "A rule verification and resolution framework in smart building system," in *Proc. Int. Conf. Parallel Distrib. Syst.*, 2013, pp. 438-439.

[28] F. Botelho, Y. Kohayakawa, and N. Ziviani,"A practical minimal perfect hashing method," *Lecture Notes Comput. Sci.*, vol. 35, no. 3, pp. 241–254, 2005.

[29] H. Lim, and H.N. Chu, "Hierarchical binary search tree for packet classification," *IEEE Commun. Lett.*, vol. 11, no. 8, pp. 689–691, Aug. 2007.

[30] Y. Sun, H. Luo, and S. K. Das, "A trust-based framework for fault-tolerant data aggregation in wireless multimedia sensor networks," *IEEE Trans. Dependable Secure Comput.*, vol. 9, no. 6, pp. 785–797, Nov./Dec. 2012.



Yan Sun received the BS degree from Beijing Jiaotong University in 1992, and the MS and PhD degrees from Beijing University of Posts and Telecommunications in 1999, and 2007, respectively. She is an associate professor at the School of Computer Science, Beijing University of Posts and Telecommunications, China. She is also a research member at the Beijing Key Lab of Intelligent Telecommunication Software and Multimedia. Her research interests include Internet of Things, sensor networks, smart environments, and embedded systems.

Tin-Yu Wu received the MS and PhD degrees from the Department of Electrical Engineering, National Dong Hwa University, Hualien, Taiwan in 2000 and 2007, respectively. He is currently an assistant professor in the Department of Computer Science & Information Engineering, National Ilan University, Taiwan. His research interests focus on the next-generation Internet protocol, mobile computing, and wireless network.

Guotao Zhao received the PhD degree in computer science from Beijing University of Post and Telecommunication in 2012. He is a staff member at Network Computing Middleware, IBM Research-China. He is working on the IoT Messaging advanced technologies for IBM MessageSight. He joined IBM Research in 2012. His research interests focus on Internet of Things.

Mohsen Guizani (S'85-M'89-SM'99-F'09) received the BS (with distinction) and MS degrees in electrical engineering, and the MS and PhD degrees in computer engineering in 1984, 1986, 1987, and 1990, respectively, all from Syracuse University, Syracuse, New York. He is currently a professor and an associate vice president of Graduate Studies at Qatar University, Qatar. Previously, he served as the chair at the Computer Science Department, Western Michigan University from 2002 to 2006 and the chair at the Computer Science Department, the University of West Florida from 1999 to 2002. He also served in academic positions at the University of Missouri-Kansas City, University of Colorado-Boulder, Syracuse University, and Kuwait University. His research interests include wireless communications and mobile computing, computer networks, mobile cloud computing and smart grid. He currently serves on the editorial boards of many International technical Journals and the founder and EIC of Wireless Communications and Mobile Computing Journal published by John Wiley (http://www.interscience.wiley.com/ jpages/1530-8669/). He is also the founder and general chair of the International Conference of Wireless Communications, Networking and Mobile Computing (IWCMC). He is the author of nine books and more than 350 publications in refereed journals and conferences. He guest edited a number of special issues in IEEE Journals and Magazines. He also served as a member, chair, and general chair of a number of conferences. He was selected as the Best Teaching Assistant for two consecutive years at Syracuse University, 1988 and 1989. He was the chair of the IEEE Communications Society Wireless Technical Committee and Chair of the TAOS Technical Committees. He served as the IEEE Computer Society Distinguished Speaker from 2003 to 2005. He is a fellow of the IEEE, a member of the IEEE Communication Society, the IEEE Computer Society, the ASEE, and a senior member of the ACM.