International Journal of Applied Science and Engineering
2019. 16, 3: 201-214

Fault Tolerance Events Ordering with Coverage and Connectivity Aware Clustering in Wireless Sensor and Actuator Networks

Yi-Chao Wu

Interdisciplinary Program of Green and Information Technology, National Taitung University, Taitung, Taiwan

Abstract: A network delay or a fail link is prone to be occurred in wireless sensor and actuator networks (WSAN). Since the events captured by sensors are related to others, events ordering is an vital issue in WSAN. However, it is hard to guarantee that no prior event is transmitted while the latter events arrive at actuator in network. While a fail link is occurred, it could be assumed as a special case that the delay is set to infinity. Event ordering by double confirmation (OBDC) was a typical events ordering while a network delay was occurred. Fault tolerance events ordering by aging learning (FTEOAL) was a typical events ordering mechanism while a fail link existed. However, both of them assumed that sensors were all covered by one actuator and none of them was the orphan sensor. Once a sensor was non-covered by an actuator or an orphan sensor existed, the rate of correct events ordering will decrease. Therefore, in this paper, we proposed a fault tolerance events ordering with coverage and connectivity aware clustering, FTEOCCAC. The simulation results demonstrated that events were treated in correct order even if the sensors are deployed randomly or an orphan sensor existed while a network delay or a fail link existed in network. The rate of correct events ordering could be closed to the expect value.

Keywords: Wireless sensor and actuator networks; network delay; fail link; orphan sensor; coverage and connectivity aware clustering; rate of correct events ordering.

1. Introduction

WSAN called wireless sensor and actuator networks was consisted of a set of sensors and some rich-resource actuators. Sensors send the sensed data for the physical environment. Actuators perform actions based on the sensed data. Unlike WSN, where the sink takes actions for events in a centralized manner, actuators in WSAN perform actions with a local and distributed manner. Since the transmission radius, computation, and energy of WSAN are different from those of WSN, the coordination and communication of sensor-actuator becomes an important feature in WSAN. Therefore, WSAN has brought up new challenges and dimensions to WSN in the past and the current solutions of WSN cannot be used for WSAN, directly [1-3]. In WSAN, it also could monitor critical conditions in the physical environments [4-5]. In the clusters of WSAN, each actuator requires the correct translating of contexts in the context-aware environments, where the contexts were consisted of inter-related events [6-13].

Most of researches in WSAN focused on simple context events to prolong the network lifespan, do fault tolerance, and save power energy.

Corresponding author; e-mail: alanwu@nttu.edu.tw
doi:10.6703/IJASE.201911_16(3).201

Received 8 March 2019
Revised 18 September 2019
Accepted 15 October 2019

By the progressing of sensor technique and the increasing of human requirement, the accurate, critically monitored, reacted applications took attentions gradually. For these applications, events were related to each other and the ordering of events to be discovered became an important issue. Few of researches however addressed the events ordering issue and no method ensures the correct events ordering [6-13].

Since the relation of events could be considered as temporal, the ordering of events captured by the source sensors is vital to treat events timely and correctly. Hence, to track the events ordering correctly was required in WSAN [7-13]. To show the practical usefulness of our proposed model, an object moving path tracking application was illustrated. In this application, events ordering is required since actuator cannot guarantee that no prior event is transit over network while the actuator receives the current event. In Figure 1, an intruder is detected by sensor 1 (s_1), s_2, s_3, s_4, s_5, and s_6. Once no delay existed, the ordering, such as s_1-s_2-s_3-s_4-s_5-s_6, is the same as the moving path of the invader. However, the event captured by s_5 may be subjected to a delay. The ordering, such as s_1-s_2-s_3-s_4-**s_6-s_5**, is different from the correct moving path. The error decision is made for arresting the invader. No action is performed at once while receiving other events, since the later events captured at t_j (timestamp j) may arrive before the events captured at t_i ($i < j$). It is caused that some previous events still being in transit in network due to the propagation delay. Hence, an events ordering algorithm is needed. Undoubtedly, events ordering is a indispensible mechanism for some applications with the rate of correct events ordering up to 100%. Therefore, an ordering by double confirmations, OBDC, to ensure the correct events ordering was proposed [13].

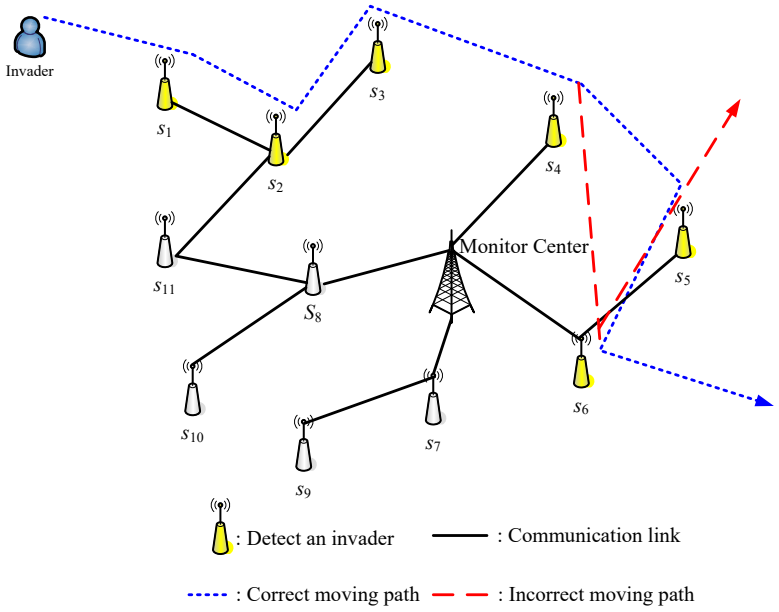

Figure 1. Temporal events ordering [14].

In OBDC, it assumed that the routing paths were kept and never broken in the system model [13]. Once a fail link in a cluster existed, OBDC will be fail because some confirmation messages may not arrive at the actuator. However, in the real situation, the fail link in wireless network may be prone to be happened. Hence, how to execute events ordering with fault tolerance needs to be addressed. Hence, a fault tolerance events ordering by aging learning, FTEOAL, was proposed to consider the network delay and fail link problems for events ordering [15].

Although FTEOAL solved the network delay and fail link for events ordering, jointly, it also assumed that the all sensors must be covered by one actuator and there is no orphan sensor as the same as the existed events ordering algorithms [7-13]. Once a sensor is out of coverage of actuator or no routing among sensors and actuator, FTEOAL cannot be processed. In fact, some sensors may not be covered by at least one actuator and some sensors may be the orphan sensors. Therefore, we combined the coverage and connectivity aware clustering within *k* hops (CCAC-k) [16] with FTEOAL [15] to propose a fault tolerance events ordering with coverage and connectivity aware clustering, FTEOCCAC, in this paper.

The remainder of this paper was arranged in the following chapters. Section 2 stated the related work. Section 3 presented our algorithm, FTEOCCAC. The performance evaluation was described in Section 4. Finally, we concluded the paper and described the future work.

2. Related Work

In WSAN, it was divided into automated and semi-automated network [3]. Since sensors could send the data to actuators in automated network, the actuators could take decisions and perform actions. The sensors thus could save energy because they could not forward data to the sink via multi-hop. Moreover, the network lifespan could be increased and the latency of performing actions could be decreased. Hence, we applied the automated architecture for WSAN [3]. Certainly, the events ordering was needed in WSAN [7-13].

Moreover, the existing events ordering algorithms in WSAN addressed nothing for fault tolerance. Actually, the fail link and network delay were all prone to be happened in wireless communication medium. Hence, an efficient events ordering should consider the fail link and network delay, jointly. Here, the actor was defined as the same as the actuator.

In most of researches, events were often defined as the independent ones. Few of researches discussed the issues for ordering events before. Recently, the ordering of events became critical since the events interpreted in different order may carry out the different results. For this issue, some algorithms of events ordering were proposed in WSN [7-13].

Once events are co-related, the time differences among events can suggest to detecting the incident in an emergency environment. If the co-related events are captured, a mechanism is needed to interpret the co-related events. To ensure the correct ordering, an actuator must guarantee that no prior event is still in transit while the latter events are treated in network. However, most of algorithms addressed nothing for the event ordering issue, since no delay was assumed in network. Therefore, the ordering of events arriving at the actuator must be in the same ordering as the ordering of events captured by sensors.

Ordering by confirmation, OBC, was another typical events ordering mechanism in WSAN [7-11]. In OBC, no sensor sent a duplicate message to actuator and no logical rings was needed as temporal message ordering in sensor networks, TMOS [17]. Only the leaf sensors received confirmation messages. The leaf sensors then routed the confirmation message to the actuator through the routing path. OBC, however, cannot ensure that the ordering of all events was in correct order. As the number of events increased, the probability of treating events in correct order may be decreased. Therefore, an ordering by double confirmations, OBDC was proposed [13]. In OBDC, only one buffer was required. Moreover, OBDC ordered the events correctly with a little higher energy consumption and time than OBC.

OBDC was to set the first in first out (FIFO) channel between any two sensors. The object of FIFO channel was to determine the order if more than one event arrived at the same sensor. For example, while s_i received e_i, s_i detected e_j in this moment. Because the property of FIFO channel, e_j was routed back to actuator first. To make events treated correctly in OBDC, an

actuator needed to send the confirmation messages to all leaf sensors when it received an event forwarded by sensor. While an actuator received e_i, it stored e_i in buffer. After a predefined time, the actuator broadcasted confirmation message m_{max} to all leaf sensors, where *max* indicated the maximal order of events in buffer. For example, max was set to 2, such as $(e_0 \rightarrow e_2 \rightarrow e_1)_a$. After all m_{max} sent back to actuator, the actuator treated each e_i in buffer and handed these event(s) orderly out to application, if $t^s_{e_i} \leq t^g_{m_{max}}$, where $t^s_{e_i}$ and $t^g_{m_{max}}$ were defined as the timestamp of e_i sent by source sensor and the timestamp of m_{max} sent by actuator.

In OBDC, only leaf sensors needed to route these messages back to the actuator. Hence, OBDC did not suffer a non-determinism delay. Moreover, OBDC only needed one buffer to save the received events to be treated correctly. However, OBDC cannot execute the events ordering while a fail link was occurred in network. To address the fail link and network delay issue, jointly, FTEOAL was proposed [15].

However, OBC, OBDC, and FTEOAL all assumed that the link among sensors is not in fail and no sensor is out of the coverage of the actuator. In the real condition, the assumptions are impossible. While the deployment of sensors is random, the OBC, OBDC, and FTEOAL all may not be processed. Hence, a fault tolerance events ordering with coverage and connectivity aware clustering, FTEOCCAC, was proposed based on FTEOAL and CCAC-*k* [15-16] in this paper.

3. Fault Tolerance Events Ordering with Coverage and Connectivity Aware Clustering

In the existing protocols and algorithms in WSAN, events are often defined as the independent ones because they are not co-related to each other. By the progress of technique in both micro-electronic and communication, WSAN may not only perform actions but also perform them in correct order. Here, the correct order was defined as that the ordering of events detected by sensors were the same as the ordering of events treated by actuators even through the ordering of events detected by sensors was different from the ordering of events reach the actuator. However, events ordering may be not correct if the former event arrived at actuator before the latter event.

Before ordering events, the pre-configuration included clustering algorithm, clock synchronization, and FIFO channel was required. For clustering, the existing events ordering algorithms all assumed that no orphan sensor existed and each sensor must be covered by at least actuator for clustering [7-13]. Actually, the above assumption is impossible since the sensors are scattered. If a sensor was not covered by an actuator, the confirmation message cannot be broadcasted to the sensor and events may be treated in incorrect order.

Moreover, most of events ordering mechanisms only considered the network delay without addressing the fail link problem. In fact, both of them are prone to be happened in wireless communication medium. Once a fail link existed, the existing events ordering mechanisms cannot be executed. Hence, a fault tolerance events ordering with coverage and connectivity aware clustering, FTEOCCAC, was proposed in WSAN. FTEOCCAC was involved CCAC-k [16] and FTEOAL [15] in WSAN. Table 1 described the notation in FTEOCCAC based on FTEOAL [15].

Table 1. Notation in FTEOCCAC [15].

Notation	Definition	Unit
$t^s_{e_i}$	Timestamp of e_i sent by source sensor	ms
$t^a_{e_i}$	Timestamp of e_i arrived at actuator	ms
$t^t_{e_i}$	Timestamp of e_i treated by actuator	ms
$t^g_{m_i}$	Timestamp of m_i generated by actuator	ms
$t^a_{m_i}$	Timestamp of m_i arrived at actuator	ms
DT_{e_i}	Delay time of e_i	ms
DT_{m_i}	Delay time of m_i	ms
DTG_{e_i}	Delay time of generating e_i	ms
DT_{one}	Propagation time in one hop	ms
PT_{e_i}	Period time of e_i treated by actuator	ms
H_{e_i}	Hop count of e_i from source sensor to actuator	

In FTEOCCAC, sensors have the limited computing, sensing, and wireless communication capabilities without short battery life. Actuators were the resource rich nodes with higher computing, longer transmission radius, and longer battery life. Actuator could also have the sensing capability. The number of sensors was much more than *that* of actuators.

Sensors and actuators were stationary. Sensors were scattered randomly. The hop count from the sensor to actuator was set to k. The sensing range and probability were regular. The probability of network delay between any two sensors was from 0% to 30%. The probability of a fail link between any two sensors was from 0% to 30%. The fail communication link was temporal not fixed. The probability of a sensor to be a source sensor was random.

The ordering of events captured by sensors, the ordering of events arrived at actuator, and the ordering of events treated by actuators were defined in the following. Without any delay, $(e_i \rightarrow e_j \rightarrow e_l)_t$ must be true even if $(e_i \rightarrow e_j \rightarrow e_l)_s$ was false, where e_i was denoted as event i and $i < j < l$. It was caused that the time of transmitting data must be much less than *that* of moved events.

$(e_i \rightarrow e_j)_s$: e_i was sent by sensors before e_j.

$(e_i \rightarrow e_j)_a$: e_i arrived at actuator before e_j.

$(e_i \rightarrow e_j)_t$: e_i was treated by actuators before e_j.

However, delay was prone to happened in wireless communication. Delays may originate as listed as follows. Once an event meant a delay in its routing path, the later event may arrive at actuator before the former one, such as $(e_{i+1} \rightarrow e_i)_a$. Unfortunately, it was difficult to know whether any prior event was still transited in network while actuator received a message of event. For some applications required all events treated in correct order, how to ensure the events to be treated in correct order became an important research.

The $(e_i \rightarrow e_j \rightarrow e_l)_t$ should be true even if $(e_i \rightarrow e_j \rightarrow e_l)_a$ is false. The formulation of events ordering is illustrated in the following example. Figure 2 showed a time sequence for error events ordering. Without any delay, $(e_0 \rightarrow e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4)_t$ should be true. Once some of events meet a delay, such as e_0 and e_3, the later e_1 arrives at actuator prior to e_0 and the later e_4 arrives at actuator prior to e_3. When the actuator receives e_1, it could not know e_0 still in transit over the network and removes e_1 from buffer in incorrect order $(e_1 \rightarrow e_0)_t$. In the same way, actuator removes e_4 from buffer in incorrect order $(e_4 \rightarrow e_3)_t$. Without any events ordering solution, the ordering of events treated must be indentical as ordering of events arrived at actuators. Thus $(e_0 \rightarrow e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4)_t$ was changed to incorrect ordering, such as $(e_1 \rightarrow e_0 \rightarrow e_2 \rightarrow e_4 \rightarrow e_3)_t$. For example, in Figure 3, s_8 and s_{25} sent e_1 and e_2 to actuator. In the same time, the communication link between s_{15} and s_{16} was fail. Some of m_0 thus cannot be forwarded to the actuator [15].

Moreover, the events ordering algorithms will not be performed if the sensors are not covered by a least one actuator or an orphan sensor existed. For example, s_{20}, s_{21}, s_{22}, and s_{24} are the orphan sensors, as shown in Figure 4. The s_{20} and s_{24} are out of the coverage of the actuator, as shown in Figure 4 [15]. Therefore, FTEOCCAC was proposed in this paper.

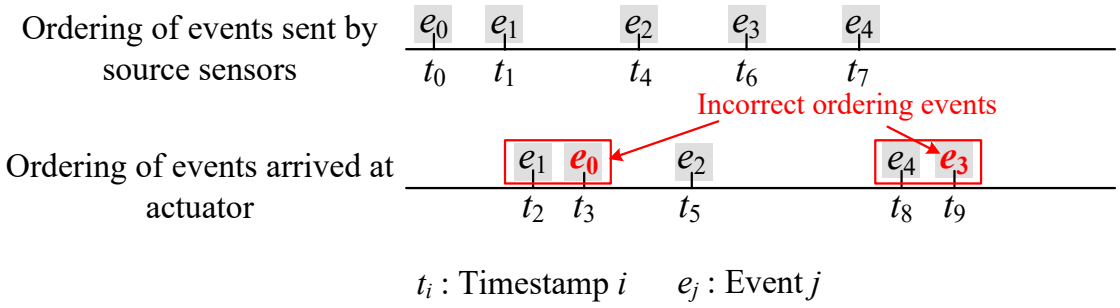

Figure 2. Time sequence of error events ordering [13].

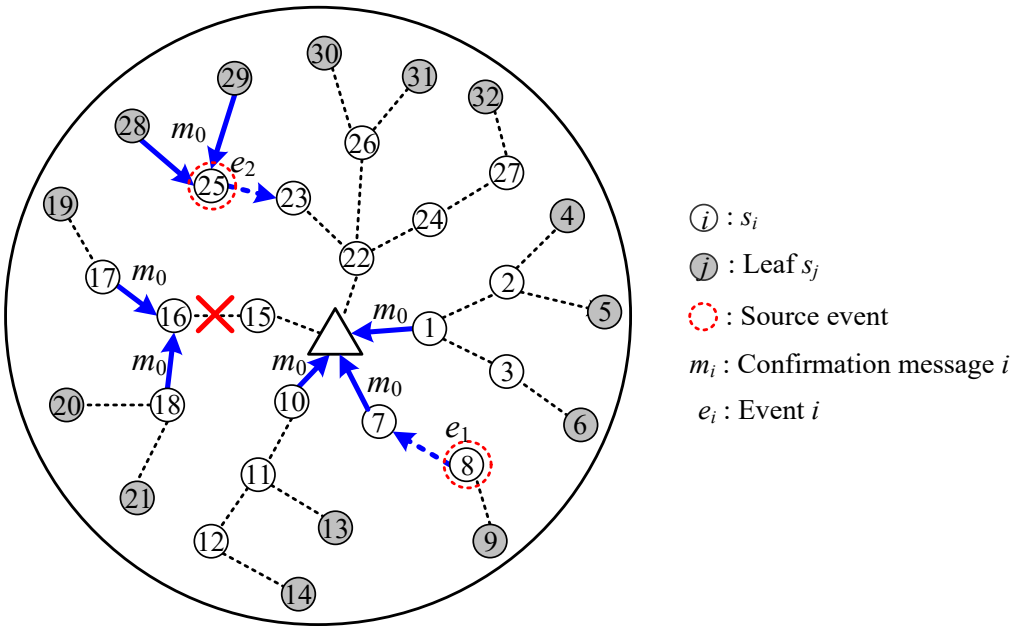

Figure 3. Fail link existed in events ordering [14].

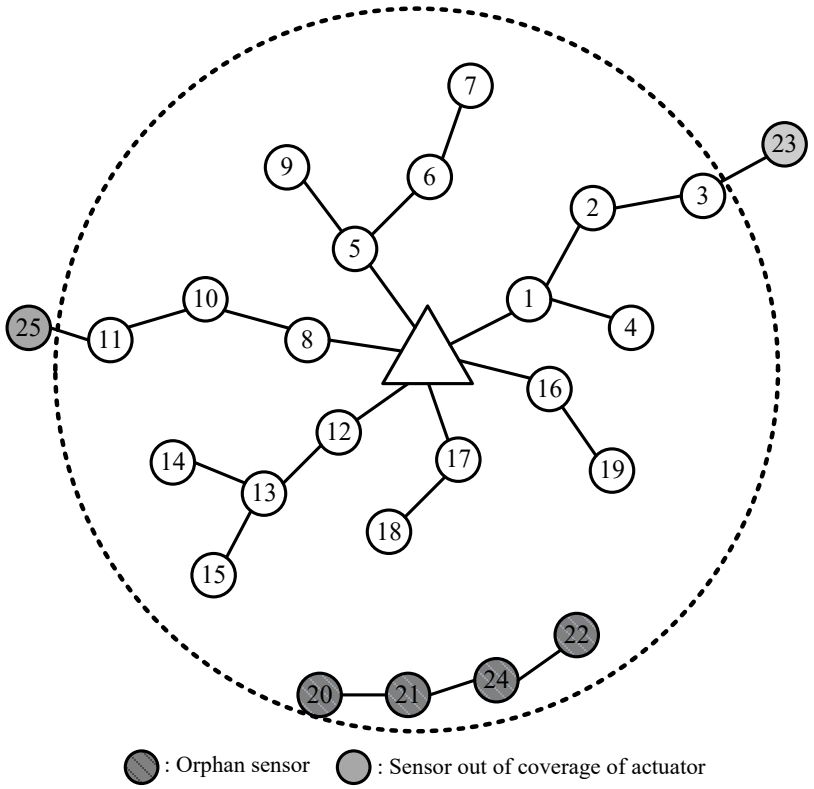

Figure 4. Coverage and connectivity problems [16].

FTEOCCAC is based on FTEOAL with CCAC-*k* [15-16]. Differed from FTEOAL, the clustering of FTEOCCAC is replaced the periodic, event-driven and query-based protocol, PEQ [18] with CCAC-*k* [16]. FTEOCCAC defined an aging waiting time (*AWT*) and an expected rate of correct events ordering (RC_{exp}), $0 \le RC_{exp} \le 1$. *AWT* was adjusted dynamically. The flowchart of FTEOCCAC is shown in Figure 5. The $t_{e_i}^t$ was calculated as $t_{e_i}^s + AWT_r$, where AWT_r denoted the redundant of *AWT*, if *AWT* was not expired. High aging learning time, ALT_{high}, was calculated as (1) [14]. ALT_{high} will be getting to DT_{max} defined as the maximal delay time. Low aging learning time, ALT_{low}, was calculated as (2) [15]. *AWT* was calculated as (3) [15]. *AWT* was set to $t_{AWT_{ex}} - t_{AWT_{st}}$, where $t_{AWT_{st}}$ and $t_{AWT_{ex}}$ denoted the start and expiration of *AWT*, respectively. Initially, ALT_{high} and ALT_{low} were set to 0 and DT_{max}. Based on [15], N_l and N_{cl} were also defined as the number of learning and the current N_l, respectively.

$$ALT_{high} = \begin{cases} t_{e_i}^a - t_{e_i}^s, \text{ if } t_{e_i}^a - t_{e_i}^s > ALT_{high} \\ ALT_{high}, \text{ if } t_{e_i}^a - t_{e_i}^s \le ALT_{high} \end{cases} \quad (1)$$

$$ALT_{low} = \begin{cases} AWT_{ex} - t_{e_i}^a, \text{ if } t_{AWT_{ex}} - t_{e_i}^a < ALT_{low} \\ ALT_{low}, \text{ if } t_{AWT_{ex}} - t_{e_i}^a \ge ALT_{low} \end{cases} \quad (2)$$

$$AWT = \begin{cases} AWT + ALT_{high} - ALT_{low}, \text{ if } RC < RC_{exp} \\ AWT - ALT_{low}, \text{ if } RC \ge RC_{exp} \end{cases} \quad (3)$$

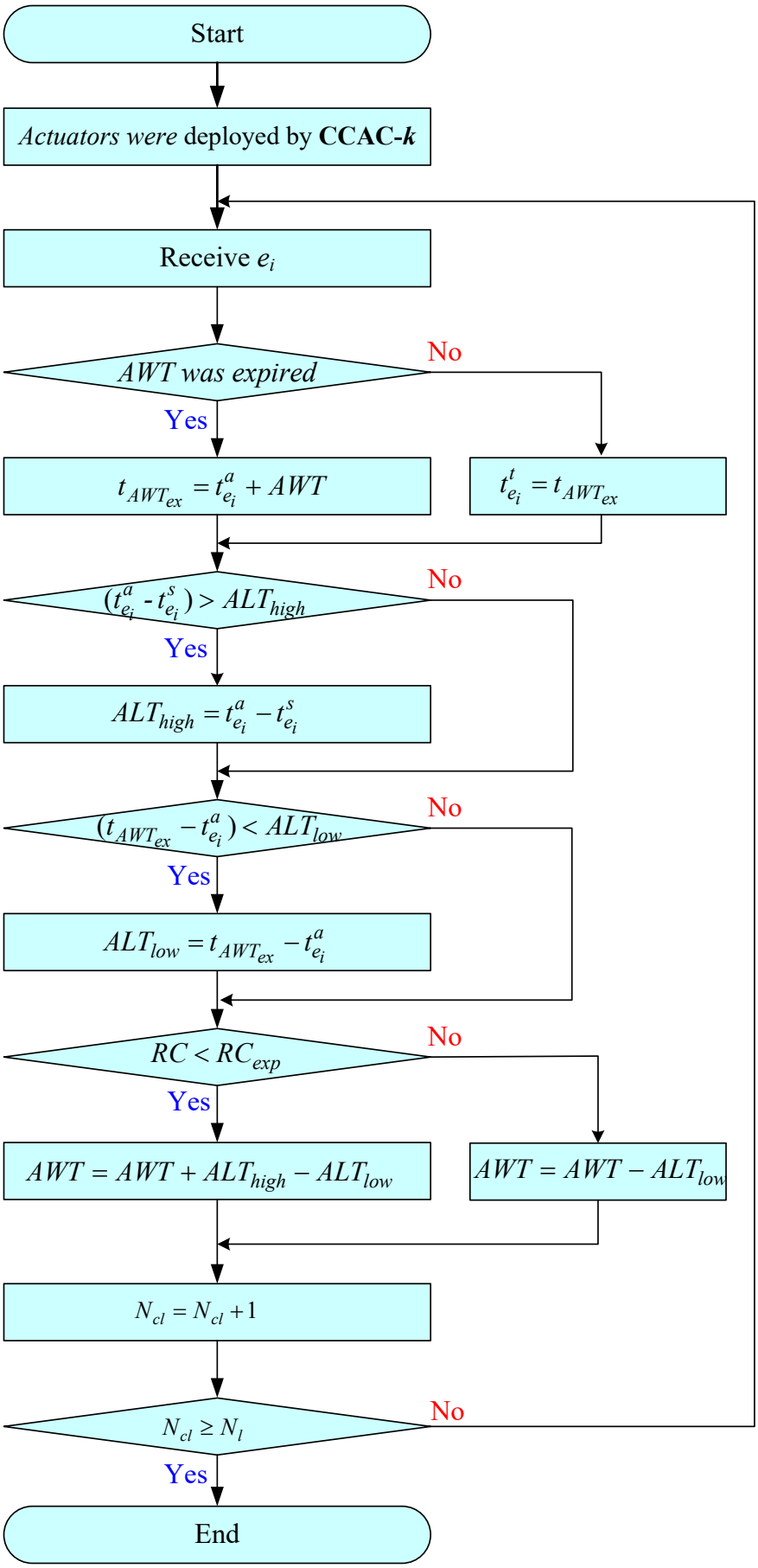

Figure 5. Flow chart of FTEOCCAC.

4. Simulation results

The simulation results were validated by C# custom simulator on the .NET platform. The simulation environment was listed in Table 2. In the simulation, it was performed on a WSAN subgroup as the cluster which the actuator acted as the cluster head in a distributed manner. Hence, the proposed scheme is designed in the distributed manner. The technical concepts and steps that are taken to demonstrate the model's validity will be explained in the following.

Delay time (*DT*) was set from 500 to 2000 ms. Delay time for generating an event, *DTG*, was set from 100 to 200 ms. P_d defined as the probability of an event with a delay in a link was set from 0% to 30%.

In the events ordering, correct ordering must be required. Thus the rate of correct events ordering (*RC*) was the main performance metric as (4) for events ordering [15]. While *RC* increased, the risk of performing error actions may increase. To address the executing time, the average time of ordering events (*AT*) was defined as (5) [15]. PT_{e_i} was defined as the period time of treating e_i as (6) [15]. Lower *AT* showed the lower executing time to order events correctly. Since FTEOCCAC could not send any confirmation message, FTEOCCAC does not have any additional overhead for events ordering. Hence, the performance evaluation of energy consumption was not considered in the simulation.

$$RC = \frac{N_e^c}{N_e} \times 100\%,\ N_e^c \le N_e \tag{4}$$

$$AT = \frac{\sum_{i=1}^{N_e} (PT_{e_i})}{N_e} \tag{5}$$

$$PT_{e_i}^t = t_{m_j}^a - t_{e_i}^s \tag{6}$$

For fault tolerance events ordering, we added a probability of an event meeting a fail link, P_l, from 10% to 30% to fit the real condition for fail link in wireless network. In the events ordering with fault tolerance, correct ordering was also required. Therefore, *RC* and *AT* were the performance metrics for FTEOCCAC. The simulation results will describe and demonstrate the models validity for FTEOCCAC.

Table 2. Simulation environment [15].

Name	Value
Simulation area, *A*	1000 × 1000 (m^2)
Number of sensors, N_s	250, 500, 1000, 1500, 2000
Number of events, N_e	100, 150, 200, 250, 300
Transmission radius of sensor, r_s	20 (m)
Transmission radius of actuator, r_a	140 (m)
Delay time, *DT*	500-2000 (ms)
Delay time for generating an event, *DTG*	100-200 (ms)
Propagation time in one hop, DT_{one}	20 (ms)
Packet size, S_p	50 (Bytes)
Probability of an event with a delay in a link, P_d	0-10%, 0-20%, 0-30%
Probability of an event meeting a fail link, P_l	0-10%, 0-20%, 0-30%
Expected value of *RC*, RC_{exp}	90%

P_l, N_e, N_l, DT, and DTG were varied to evaluate RC and AT under FTEOCCAC and FTEOAL. In Figure 6, RC was evaluated based on different P_l. DT and DTG were set to 2000 ms and 100 ms, respectively. It showed that RC in FTEOCCAC was close to RC_{exp}, such as 90%, but RC in FTEOAL was about 70% while P_l and N_e still increased. While the fail link and network delay existed together, FTEOAL cannot be executed. Hence FTEOCCAC was more better than FTEOAL for RC with different P_l. Because RC in FTEOAL was only influenced by P_l varied from 0% to 30%, RC in FTEOAL was random.

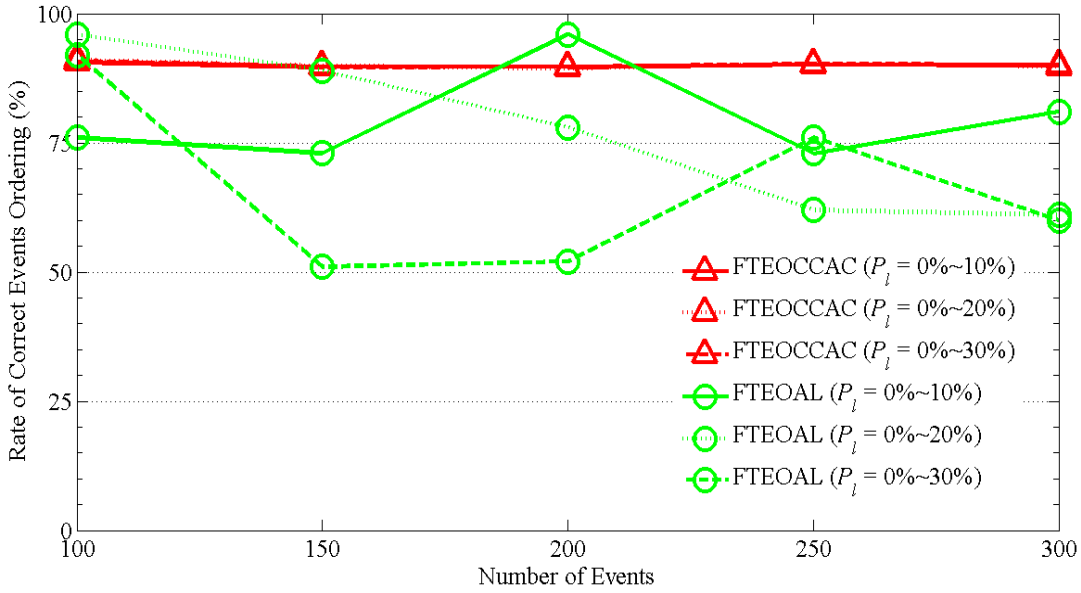

Figure 6. *RC* in FTEOCCAC and FTEOAL for different P_l (DTG = 100 ms, DT = 2000 ms, P_d = 0-30%).

In Figure 7, we could find that RC in FTEOCCAC was higher than RC in FTEOAL. While DT was increased, RC in FTEOCCAC was almost fixed. Figure 8 showed that RC in FTEOCCAC was close to RC_{exp} but RC in FTEOAL was lower than 75%. It proved that FTEOCCAC had higher RC than FTEOAL with different DTG. As DTG increased over time, RC in FTEOCCAC was almost the same. Hence, FTEOCCAC was better than FTEOAL for RC while DTG or DT increased.

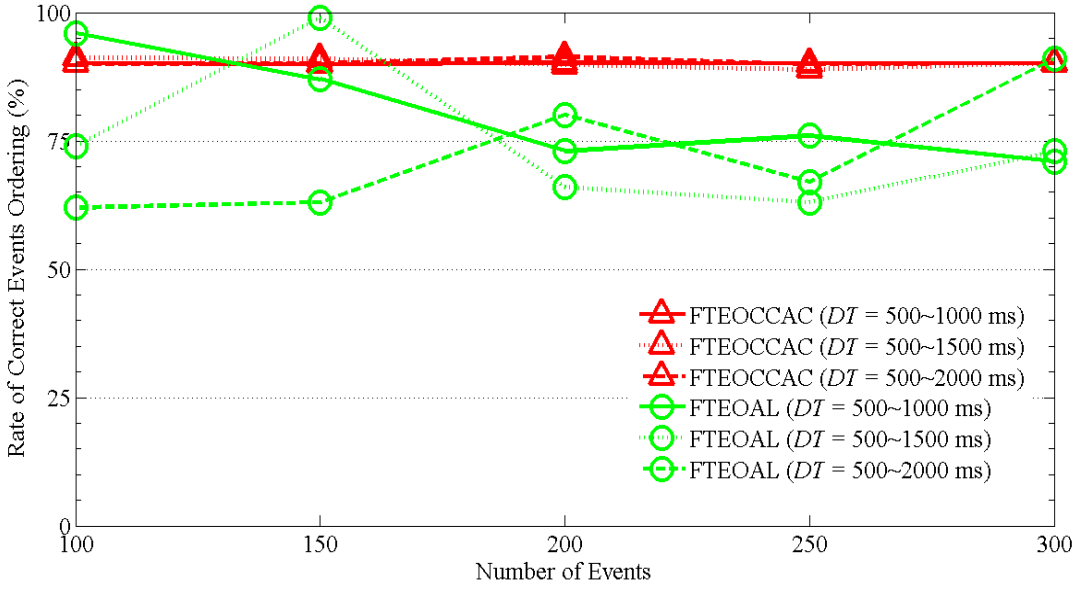

Figure 7. *RC* in FTEOCCAC and FTEOAL for different DT (P_l = 0-30%, DTG = 100 ms, P_l = 0-30%).

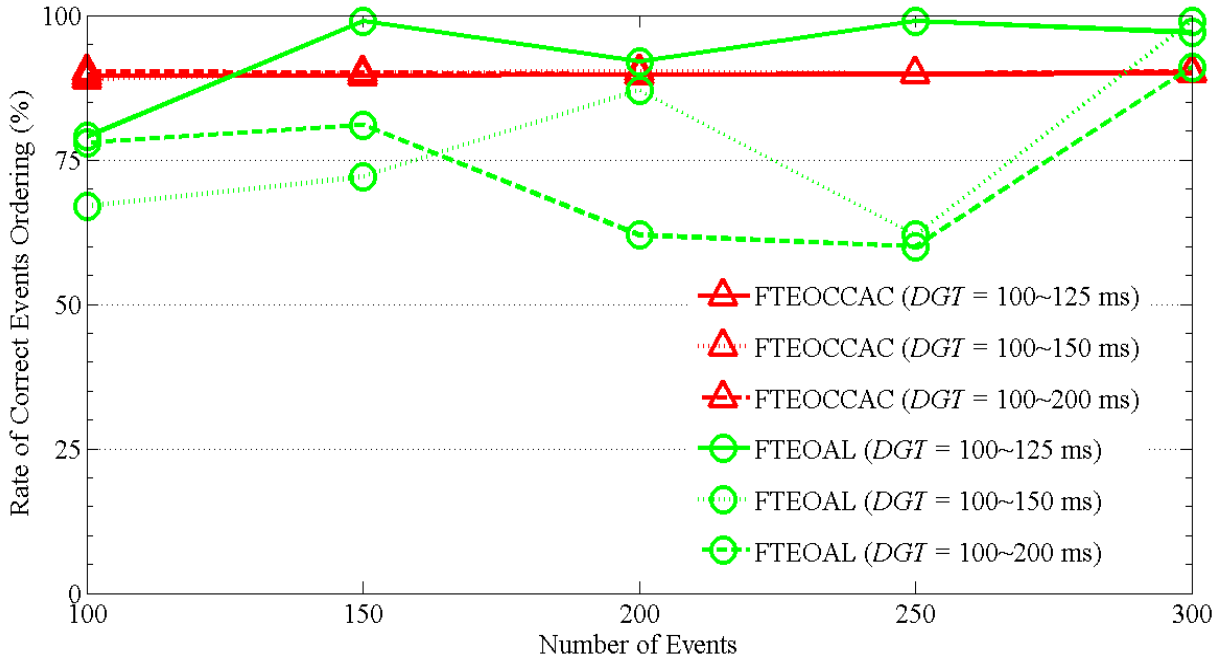

Figure 8. *RC* in FTEOCCAC and FTEOAL for different *DTG* (P_l = 0-30%, *DT* = 2000 ms, P_l = 0-30%).

Since some applications needed events treated timely, so average time of ordering events (*AT*) was also important. However, for *AT*, while a fail link was occurred in network, FTEOAL could not be executed. *AT* of FTEOAL was thus infinite. Hence, we only evaluated *AT* under FTEOCCAC and FTEOAL while only network delay existed without any fail link, such as P_l = 0%.

In Figure 9, we evaluated *AT* for different P_d under FTEOCCAC and FTEOAL. It showed that *AT* in FTEOCCAC was almost the same while P_d increased. *AT* in FTEOAL was increased while P_d was increased. In Figure 10, *AT* was evaluated in FTEOCCAC and FTEOAL with different *DT*. While *DT* was increased, *AT* in both FTEOCCAC and FTEOAL were increased. In Figure 11, it showed that *AT* in FTEOCCAC was higher than *AT* in FTEOAL with different *DTG*. However, *AT* in FTEOCCAC did not increase while *DTG* was increased.

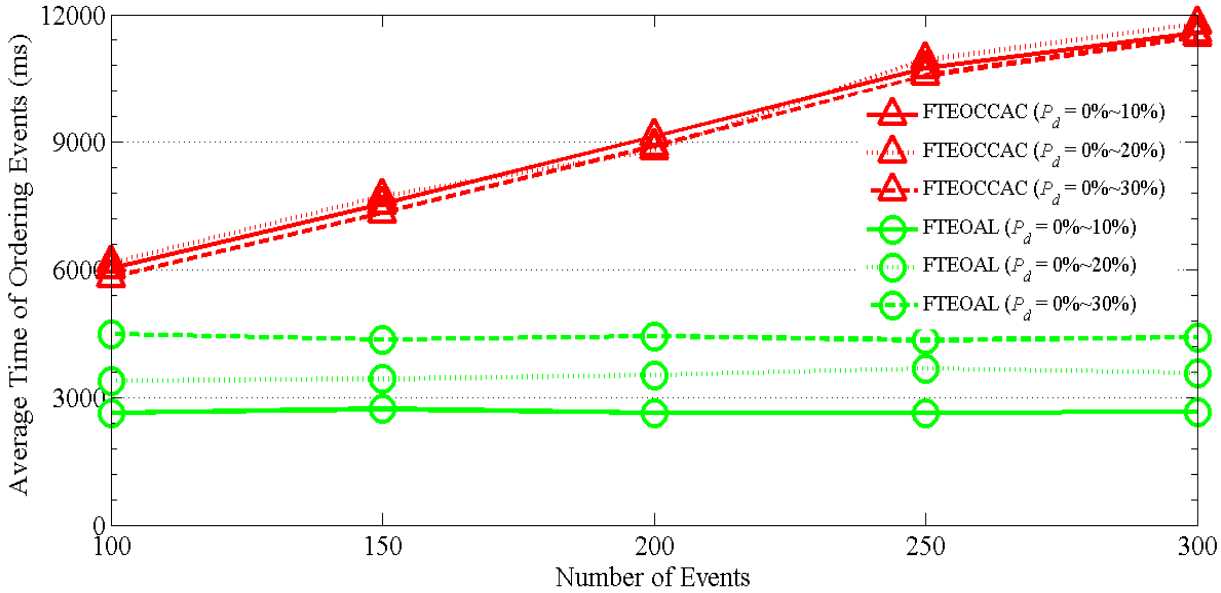

Figure 9. *AT* in FTEOCCAC and OBDC for different P_d (*DTG* = 100 ms, *DT* = 2000 ms, P_l = 0%).

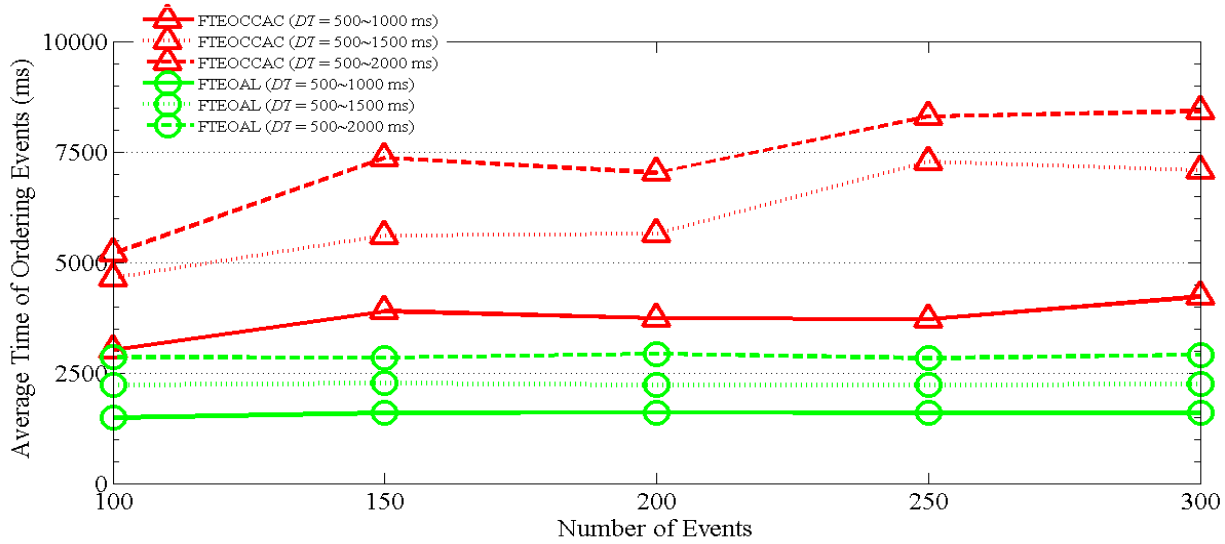

Figure 10. *AT* in FTEOCCAC and OBDC for different *DT* (P_d = 0-30%, *DTG* = 100 ms, P_l = 0%).

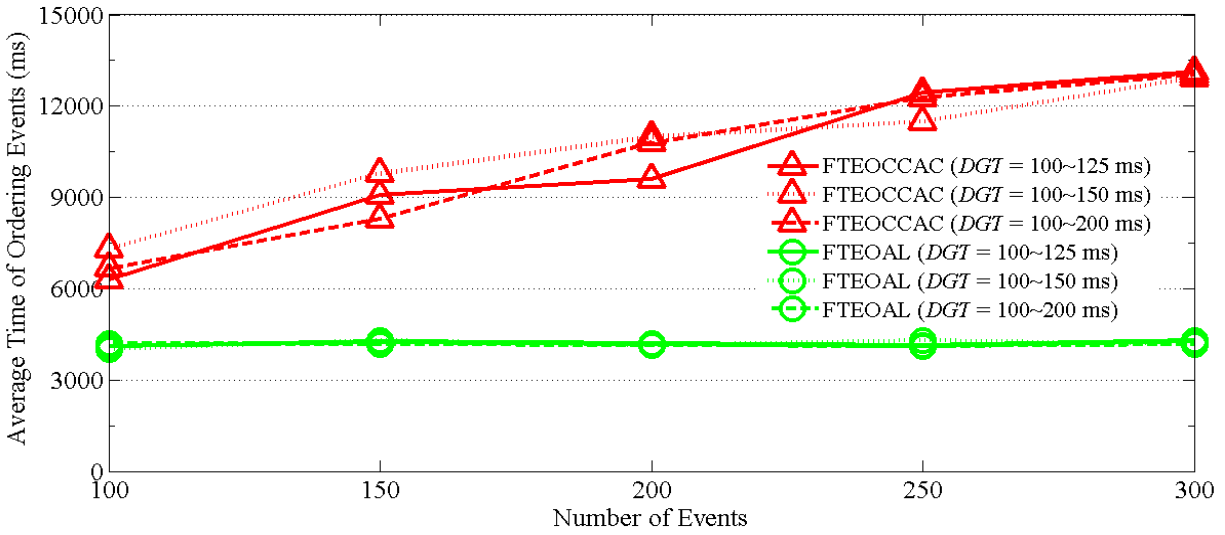

Figure 11. *AT* in FTEOCCAC and OBDC for different *DTG* (P_d = 0-30%, *DT* = 2000 ms, P_l = 0%).

For different P_l, *RC* in FTEOCCAC was higher 13% than *that* in FTEOAL. *RC* in FTEOCCAC was higher 17% than *that* in FTEOAL for different *DT*. For different *DTG*, *RC* in FTEOCCAC was higher 15% than *that* in FTEOAL. Hence, it showed that *RC* in FTEOCCAC was all higher than *RC* in FTEOAL, as P_l, *DTG*, and *DT* varied. In different P_d, *AT* in FTEOCCAC was higher 156% than *that* in FTEOAL. *AT* in FTEOCCAC was higher 160% than *AT* in FTEOAL in different *DT*. For different *DTG*, *AT* in FTEOCCAC was higher 150% than *AT* in FTEOAL.

In different *DTG*, *DT*, and P_l, *DT*, and *DTG*, *RC* of deviation of FTEOCCAC between the maximum and minimal cases was within 2%. In different P_l, *DT*, and *DTG* in FTEOAL, *RC* of deviation between the maximum and minimal cases was over 40%. It showed that FTEOAL addressed nothing for fail link issue. In different *DTG* and P_d, the *AT* of deviation of FTEOCCAC between the maximum and minimal cases was within 2%. *AT* of deviation between the maximum and minimal cases was over 15% for different *DT*. Hence, *AT* in FTEOCCAC was only affected by *DT*. For different *DTG* in FTEOAL, the *RC* of deviation in the maximum and minimal cases was within 2%. However, *AT* of deviation between the maximum and minimal cases was over 15% in different P_d and *DT*. It showed that *AT* in FTEOAL was affected by P_d and *DT*.

5. Conclusions

By the requirement of real-time and reaction, events may be co-related to each other. Without any delay, events must be treated by actuators in correct order. However, delay is easy to be caused from network dynamics, MAC layer protocol, multiple hops, and medium access. It is difficult to ensure there is no former event still transit over network while the later event arrives at actuators.

OBDC was a typical events ordering algorithm in WSAN. A corresponding duplicated confirmation message was broadcasted to leaf sensors in OBDC while an event arrived at actuator. While all messages routed back to the actuator, the events prior to this message will be removed from buffer and handed out to the applications. Hence, events could be treated in correct ordering by OBDC. However, OBDC only addressed the network delay without considering the fail link. Once a fail link in cluster existed, some confirmation messages cannot route back to the actuator. Thus the events ordering cannot be executed.

To solve the fail link problem, FTEOAL was proposed. However, FTEOAL assumed that the sensors are deployed manually. Hence, no sensor is non-covered by actuator and no orphan sensor existed. In fact, this assumption is impossible in the real situation. Once the sensors are scattered randomly, FTEOAL cannot be performed.

Therefore, we combined FTEOAL and CCAC-k to propose a fault tolerance events ordering with coverage and connectivity aware clustering, FTEOCCAC, in this paper. In FTEOCCAC, AWT and ALT were defined. RC_{exp} needed to be set. AWT and ALT were adjusted dynamically to let the new RC close to RC_{exp} with a number of aging learning, N_{al}. The redundant time in FTEOCCAC could be lower.

In the simulation results, RC in FTEOCCAC was converged to RC_{exp}, even N_e, DTG, DT, and P_l varied over time. While a sensor was not covered by an actuator or an orphan sensor existed, FTEOAL may not be executed. AT in FTEOAL was thus infinite. Hence, AT was evaluated while sensors are covered by an actuator or no orphan sensor existed. It showed that FTEOCCAC required higher AT than FTEOAL, while DT, DTG, and P_d, varied. Although FTEOCCAC required higher AT to make RC converged to RC_{exp}, FTEOCCAC had a maximum AT based on a threshold value of N_{al}. Thus the redundant time in FTEOCCAC could be decreased by aging learning. It proved that FTEOCCAC could be used with more time while the fail link and network delay are occurred jointly.

Although RC in FTEOCCAC could be converged to RC_{exp}, it still needs more time to treat events. For time-required applications, FTEOCCAC may not be suitable. Hence, we will investigate the new learning algorithms to improve FTEOCCAC.

Finally, the event may move in much monitored area, where each area is responsible by an actuator and a large number of sensors. Thus, events ordering may be addressed in multiple actuators. In this case, the coordination among the actuators and events ordering are all required. Therefore, we will investigate the mechanism of coordinating among actuators in the future. We will also study more cases carried out shall enhance the validity of the performance being produced by their methods.

References

[1] Salarian, H., Chin, K. W., and Naghdy, F. 2012. Coordination in Wireless Sensor-Actuator Networks: A Survey." *Journal of Parallel and Distributed Computing*, 72, 7: 856-867.

[2] Mo, L., Cao, X., Chen, J., and Sun, Y. 2014. Collaborative Estimation and Actuation for Wireless Sensor and Actuator Networks. *IFAC Proceedings*, 47, 3: 544-5549.

[3] Akyildiz, I. F. and Kasimoglu, I. H. 2004. Wireless Sensor and Actor Networks: Research Challenges. *Ad Hoc Networks*, 2, 4: 351-367.
[4] Kamali, M., Laibinis, L., Petre, L., and Sere, K. 2014. Formal Development of Wireless Sensor–Actor Networks. *Science of Computer Programming*, 80, A: 25-49.
[5] Cañete, E., Chen, J., Díaz, M., Llopis, L., and Rubio, B. 2011. A Service-Oriented Approach to Facilitate WSAN Application Development. *Ad Hoc Networks*, 9, 3: 430-452.
[6] Inoue, M., Ohnishi, M., Peng, C., Li, R., and Owada, Y. 2011. NerveNet: A Regional Platform Network for Context-Aware Services with Sensors and Actuators. *IEICE Transactions on Communications*, E94.B, 3: 618-629.
[7] Boukerche, A., Araujo, R. B., Silva, F. H. S., and Villas, L. 2007. Wireless Sensor and Actor Networks Context Interpretation for the Emergency Preparedness Class of Applications. Computer Communications, 30, 13: 2593-2602.
[8] Peng, L. and Wang, B. 2010. Complex Event Processing System for Wireless Sensor and Actor Networks. In *Proceedings of IEEE International Conference on Computing, Control and Industrial Engineering*, Wuhan, China, June 5-6: 337-340.
[9] Martirosyan, A. and Boukerche, A. F. 2012. Preserving Temporal Relationships of Events for Wireless Sensor Actor Networks. *IEEE Transactions on Computers*, 61, 8: 1203-1216.
[10] Boukerche, A. F. and Martirosyan, A. 2007. An Efficient Algorithm for Preserving Events' Temporal Relationships in Wireless Sensor Actor Networks. In *Proceedings of IEEE International Conference on Local Computer Networks*, Dublin, Ireland, October 15-18: 771-780.
[11] Boukerche, A. F., Araujo, R. B., and Silva, F. H. S. 2007. An Efficient Event Ordering Algorithm that Extends the Lifetime of Wireless Actor and Sensor Networks. *Performance Evaluation*, 64, 5: 480-494.
[12] Boukerche, A. F., Araujo, R. B., Silva, F. H. S., and Villas, L. 2007. Wireless Sensor and Actor Networks Context Interpretation for the Emergency Preparedness Class of Applications. *Computer Communications*, 30, 13: 2593-2602.
[13] Tuan, C. C. and Wu, Y. C. 2011. Event Ordering by Double Confirmation in Wireless Sensor and Actor Networks. *IEEE Sensors Journal*, 11, 3: 829-836.
[14] Tuan, C. C., and Wu, Y. C. 2013. Temporal Event Ordering with Fault Tolerance for Wireless Sensor and Actuator Networks. *Wireless Personal Communications*, 68, 3: 679-695.
[15] Wu, Y. C., and Tuan, C. C. 2017. Fault Tolerance Events Ordering by Aging Learning in Wireless Sensor and Actuator Networks, *IET Communications*, 11, 12: 1895-1902.
[16] Tuan, C. C., and Wu, Y. C. 2015. *K*-Hop Coverage and Connectivity Aware Clustering in Different Sensor Deployment Models for Wireless Sensor and Actuator Networks. *Wireless Personal Communications*, 85, 4: 2565-2579.
[17] Romer, K. 2003. Temporal Message Ordering in Wireless Sensor Networks. In *Proceedings of IFIP International Workshop on Annual Mediterranean Ad Hoc Networking*, Mahdia, Tunisia, June 25-27: 131-142.
[18] Boukerche, A., Pazzi, R. N. and Araujo, R.B. 2004. A Fast and Reliable Protocol for Wireless Sensor Networks in Critical Conditions Monitoring Applications. In *Proceedings of IEEE International Conference on Modeling, Analysis and Simulation of Wireless and Mobile System*, Venice, Italy, October 4-6: 157-164.