CoAP-Based Streaming Control for IoT Applications

Joong-Hwa Jung ¹, Moneeb Gohar ² and Seok-Joo Koh ¹,*

¹ School of Computer Science and Engineering, Kyungpook National University, Daegu 41566, Korea; godopu16@gmail.com
² Department of Computer Science, Bahria University, Islamabad 44000, Pakistan; moneebgohar@gmail.com
* Correspondence: sjkoh@knu.ac.kr; Tel.: +82-53-950-7356

Received: 23 June 2020; Accepted: 13 August 2020; Published: 16 August 2020

Abstract: The Constrained Application Protocol (CoAP) is a representative messaging protocol for Internet of Things (IoT) applications. It is noted that a lot of IoT-based streaming applications have been recently deployed. Typically, CoAP uses User Datagram Protocol (UDP) as its underlying protocol for lightweight messaging. However, it cannot provide reliability, since it is based on UDP. Thus, the CoAP over Transmission Control Protocol (TCP) was recently proposed so as to provide reliability. However, the existing schemes do not provide the error handling and flow controls suitably for IoT-based streaming applications. This tends to induce throughput degradation in wireless lossy networks. In this paper, we propose a CoAP-based streaming control (CoAP-SC) scheme, which is an extension of CoAP over UDP with error handling and flow control for throughput enhancement. The proposed CoAP-SC scheme is designed by considering the sequence number of data message, the use of ACK messages, and the buffer size of sending buffer. To do this, a new CoAP option is defined. For performance analysis, the proposed scheme is implemented and compared with the existing schemes. From the testbed experimentations in various network environments, we see that the proposed CoAP-SC scheme can provide better throughput than the existing CoAP-based schemes by performing the error handling and flow control operations effectively.

Keywords: CoAP; error handling; flow control; IoT; streaming control

1. Introduction

With the growth of Internet-of-Things (IoT) services [1,2], a variety of streaming applications have been deployed, in which multimedia data measured by sensors will be delivered to the server by streaming transport [3]. The streaming transport is featured by the periodic and sequential data transmissions. It is reported that the conventional transport schemes using HTTP and TCP are not suitable for delivery of IoT applications, since these protocols are too heavy and complicated operations to support the small sensor devices in IoT networks. In the meantime, the Constrained Application Protocol (CoAP) was recently proposed. The CoAP is an application-layer protocol on top of UDP and it can be used to provide better communication performance to IoT-based constrained devices in wireless sensor networks. The CoAP is lightweight, compared to HTTP, and it provides a variety of functions for IoT services, such as resource discovery and block transfer.

However, the conventional CoAP scheme does not consider the error handling and flow controls for streaming transport, and thus the throughput performance tends to be degraded, in particular, in wireless sensor networks. For example, in CoAP over UDP, if a message is lost, retransmission will occur after a timeout event, and thus the error recovery mechanism may tend to increase a large transmission delay. The CoAP over TCP can recover the lost packet quickly by utilizing the TCP's fast retransmission, but TCP mechanism may add some overhead to the IoT environment. In addition, the CoAP over TCP inherits the complexity of TCP mechanisms that are not suitable for real-time streaming services in the IoT environment, as shown in the head-of-line (HOL) blocking problem.
To overcome these problems, we propose a CoAP-based streaming control (CoAP-SC), which is an extension of CoAP over UDP with error handling and flow controls for throughput enhancement. The proposed scheme is designed by considering the sequence number of data message, the use of ACK messages, and the buffer size of the sending buffer.

This paper is organized as follows. Section 2 briefly reviews the existing schemes for streaming transport. In Section 3, we describe the proposed CoAP-SC (CoAP with Streaming Control) scheme. Section 4 discusses the experimentation results for performance analysis. Finally, Section 5 concludes this paper.

2. Related Works

2.1. CoAP over UDP

The CoAP [4] is the widely used protocol in constrained network environments, such as sensor networks. Originally, CoAP based on UDP was developed to minimize network resource waste due to connection establishment and retransmission in network environments with low power, high loss, and low network bandwidth [5]. CoAP supports the Representational State Transfer (REST) architecture by considering the compatibility with the web services [6,7]. In addition, it provides essential functions for developing services that are not supported by UDP, such as reliable data transmission. The CoAP has been standardized in the IETF CoRE WG. Figure 1 shows the CoAP over UDP header format.

![CoAP over UDP header format](image)

The first 4 bits of the CoAP over UDP header refers to the version. The CoAP message has four types: Confirmable, Non-Confirmable, Acknowledgement, and Reset. It is expressed in the following 4 bits. The next 8 bits are the length of the token. TKL indicates a token length between 0 and 8, and 9~15 are reserved. The code field is split into a 3-bit class (most significant bits) and 5-bit detail (least significant bits). The code field indicates the message type such as GET, POST, PUT, DELETE in the request message, and the response code, such as 2.01 Created, in the response message. Message ID is used to detect duplication and also for optional reliability. Request-response message pairs have the same Message ID. If the token length is not 0, the token (indicated by the TKL) will be located after the Message ID field. The token value serves as a transaction ID. If large data is transmitted through a CoAP message, it is fragmented due to the characteristics of UDP. All the fragmented messages and the corresponding response message indicate that it is a chunk of data with the same token value. CoAP option is located between the CoAP basic header and the payload, and most CoAP extensions use this option field.

The CoAP provides many functions that UDP does not provide for service development. However, it is not suitable for streaming services [8,9]. For streaming transport, a wireless sensor (client or sender) transmits its sensing streaming data to the server (or receiver), periodically and sequentially. Basically, CoAP over UDP is designed for a simple message transport, and thus it has some limitations for streaming transport. This is because CoAP was designed based on the REST model and it uses UDP as its underlying protocol. CoAP provides only a simple error recovery mechanism using the CON and ACK types in data transmission.
Figure 2 shows the error handling mechanism of CoAP over UDP. In step (1), the ACK messages are transmitted for all CON messages. This mechanism may create unnecessary ACKs. As you can also see in step (2), this protocol does not have any field for sequence number.

![Packet analysis for CoAP over UDP.](image1)

In addition, CoAP over UDP does not provide any flow control mechanism to facilitate the streaming data transmission at the sender. These features tend to incur the degradation of throughput in wireless and lossy networks. Some works have been conducted to overcome the shortcomings of the basic CoAP model, which include the CoAP-Observe schemes [10–12]. Figure 3 shows the error handling scenario for CoAP over UDP. In this figure, ACK message for PUT message (/stream/20) has been lost and the PUT message is retransmitted after the PUT message (/stream/25) is sent. In addition, CoAP over UDP retransmits data messages if a timeout event occurs. Thus, it may give poor performance in case the messages must be processed sequentially.

![Packet analysis for error handling of CoAP over UDP.](image2)

In addition, CoAP over UDP does not provide any flow control mechanism to facilitate the streaming data transmission at the sender. These features tend to incur the degradation of throughput in wireless and lossy networks. Some works have been conducted to overcome the shortcomings of the basic CoAP model, which include the CoAP-Observe schemes [10–12]. Figure 4 shows the CoAP-Observe scenario. The receiver initiates the observation by sending a GET request message containing the CoAP-Observe option to the sender. The sender notifies the receiver of the changed status by sending a message including the Observe option, whenever the resource status changes. At this time, the Observe option can serve as a sequence number. However, these schemes...
do not address the error and flow controls effectively for streaming transport. The values included in the Observe option can only be used for reordering. That is, it cannot be used for other purposes, such as fast retransmission. Thus, they are still subject to the performance degradation in wireless lossy network.

As IoT services gradually grow, the research on convergence with web services is actively being conducted. With the demand of TCP support for IoT, the CoAP over TCP has been proposed [13]. As the CoAP message is delivered by using TCP, the reliable transmission is guaranteed, and thus the CoAP Confirmable and Acknowledge messages are no longer needed [14]. For this reason, the type field has been removed. Instead, the fields for expressing the length information of the message (Length Field and Extended Length Field) have been added, since the TCP header does not include the field for the length information. Figure 5 shows the CoAP over TCP header format.

If CoAP over TCP is used for streaming service, the TCP flow control and error control mechanisms will be used [15]. As shown in step (1) and (2) of Figure 6, it is possible to take advantage of the TCP cumulative ACK function. Thus, we expect that the CoAP over TCP can provide better performance than the CoAP over UDP, if data messages must be processed sequentially. However, for IoT devices
in a wireless network, the TCP three-way handshake messages (shown in step (3) of Figure 6) will become a burden. Hence, this may induce the performance degradation.

Figure 6. Packet analysis for CoAP over TCP transmission.

Figure 7 shows the error handling mechanism of CoAP over TCP. In step (1), the PUT message (/stream/10) has been lost, and a retransmission message was immediately sent by the fast retransmission algorithm of TCP. In addition, TCP may reduce the number of retransmissions through linear combinations of the data and cumulative ACK [16–18]. In step (2), we can see that the size of the retransmitted TCP packet has increased. This is because this packet includes the two PUT messages (/stream/10, /stream/11). We can also see the cumulative ACK in the figure.

Figure 7. Packet analysis for error handling of CoAP over TCP.

The ACK message of TCP can reduce the retransmission delay. However, it may increase the number of retransmissions due to the loss of the ACK message. Figure 8 illustrates this situation. In this figure, the PUT message (/stream/12) has been retransmitted because of ACK loss.
Another drawback of CoAP over TCP is that we have to modify the kernel to solve these problems [22].

Thus, the developer must perform additional works in the application layer. The delay caused by a three-way handshake or additional work makes TCP unsuitable for real-time streaming services.

Moreover, the CoAP over TCP tends to inherit the performance degradation issues of TCP, such as the Head of Line (HoL) blocking problem in wireless networks, since it operates on top of TCP [19–21]. Another drawback of CoAP over TCP is that we have to modify the kernel to solve these problems [22].

3. Proposed CoAP Streaming Control Scheme

Based on the analysis given in the previous section, this paper proposes an enhanced CoAP scheme with streaming control (CoAP-SC) for IoT streaming transport. Since UDP does not provide error handling and flow control functions, TCP may be used for the services that require the reliability. However, the CoAP over TCP is still subject to the kernel modification and the performance degradation for IoT streaming transport. Thus, in this paper, we design the CoAP-SC scheme based on CoAP over UDP.

In the proposed scheme, we assume that a sender (client) transmits the streaming data to a receiver (server) periodically and sequentially for IoT streaming transport. We propose the streaming control mechanisms for throughput enhancement. The proposed control mechanism performs the...
error handling and flow control functions. To do this, the sequence number (SN) is assigned to each data message by the sender, and the ACK number (AN) is given by the receiver to confirm successful reception of data message.

3.1. Initialization for CoAP-SC

The proposed scheme uses the existing CoAP initialization operation so as to arrange the resources for streaming transport between the two end nodes. These operations will be helpful to create a connection for streaming transport. Figure 10 shows the initialization process in the CoAP-SC.

In the figure, the sender first requests the creation of a resource to the receiver with a POST message. This message will include the parameters associated with the streaming service, such as authentication information. For CoAP-SC, the buffer size of the sender for streaming transport should also be included into the POST message. It is noted that the other information and operations are the same with those of the existing CoAP. When the resource is successfully created, the receiver returns the URL of the generated resource via the 2.01 response message. Then, the sender issues a GET request to the received URL, and the receiver responds with the 2.05 response message.

It is noted that the GET and its response messages should contain the sequence number (SN) and ACK number (AN) fields. SN is sequentially assigned for each data message by the sender, whereas AN is determined by the receiver to indicate that the corresponding data messages have been successfully received. Note that AN is used as a cumulative ACK number. In the initialization process, both SN and AN will be set to 0.

![CoAP-SC Initialization](image1.jpg)

**Figure 10.** CoAP-SC Initialization.

3.2. Error Handling for CoAP-SC

In CoAP with streaming control, the sender transmits data messages to the receiver, and the receiver responds to the sender with ACK messages, if necessary, as per the error handling and flow controls.

All data messages generated by the sender should contain the CoAP-SC option, which will be specified in Section III-D of this paper. The CoAP-SC option in a data message includes the SN and AN fields, which are denoted by dataMsg.SN and dataMsg.AN, respectively. The first data message will have dataMsg.SN = 1 and dataMsg.AN = 0.

Each time the receiver receives a data message from the sender, it will update its own AN value (denoted by receiver.AN and initially set to 0) as the largest SN value of data messages that have been received successfully and cumulatively. As done in the existing CoAP, if the receiver does not receive any data message for a specific time, it sends an ACK message to inform the sender of the AN status.
In the normal operation, the receiver will receive a data message with $dataMsg.SN = receiver.AN + 1$, and then it will update its $receiver.AN$ as $dataMsg.SN$.

Based on this description, in the error handling of CoAP-SC, the loss of data message is determined by the receiver, if the following condition is true:

$$dataMsg.SN - receiver.AN > 1$$

Note that the above condition indicates there may be some losses of data messages, since the receiver will expect the data message with $dataMsg.SN = receiver.AN + 1$. In this way, if a data loss is detected, the receiver sends an ACK message to the sender as a retransmission request. This ACK message includes the CoAP-SC option with the SNs of the data messages to be retransmitted.

Figure 11 shows an example of error handling operations for CoAP-SC. In the figure, the first data (SN = 1, AN = 0) is transmitted, and the receiver will update its $receiver.AN$ as 1. The second data message is lost, and the third one is successfully received by the receiver. In this case, $dataMsg.SN (=3) > receiver.AN (=1) + 1$, which indicates the loss of data message with SN = 2. This loss detection induces the receiver to generate the ACK message with SN = 2 and AN = 1. Such ACK message is repeatedly generated until the concerned data message is retransmitted and recovered, as shown in the figure.

**Figure 11.** Example of error handling operations for CoAP-SC.

In Figure 12, the receiver confirms that the PUT messages (SN: 39~43, AN: 36) have been lost, when it receives a PUT message (SN: 44, AN: 36) and sends the control messages (SN: 39~43, AN: 38) in order to request retransmission. The sender retransmits the requested messages. In CoAP-SC, the error handling mechanism can overcome the disadvantages of CoAP over UDP by providing the fast retransmission.
3.3. Flow Control for CoAP-SC

In the error handling, the receiver will generate the ACK message if a data loss is detected. The ACK messages will be also generated for flow control. This ACK message is purposed to provide the up-to-date AN information to the sender and thus facilitate the sender to transmit as much data as possible. This will result in the throughput enhancement.

The flow control for CoAP-SC is designed by considering the following two points. First, the ACK message of the receiver may be lost in the network. Secondly, ACK messages are helpful for throughput enhancement, whereas too much generation of ACK messages may rather degrade throughput performance. Thus, the ACK generation for flow control needs to be controlled appropriately.

Based on these considerations, the receiver will generate the ACK messages based on the SN and AN values of the data message and the buffer size of the sender. Note that the buffer size of the sender is already informed to the receiver in the initialization process (see Section 3.1). Specifically, on reception of a data message, the receiver will send an ACK message to the sender, if the following conditions are true:

(a) $\text{dataMsg.SN} - \text{dataMsg.AN} \geq \frac{1}{3} \text{bufsize}$ or
(b) $\text{dataMsg.SN} - \text{dataMsg.AN} \geq \frac{2}{3} \text{bufsize}$

When condition (a) becomes true, an ACK message is generated only once. On the other hand, whenever condition (b) is true, an ACK message is generated for each data message. Note that the condition (a) indicates a prior alarm for buffer fullness, whereas condition (b) represents a critical signal for buffer fullness, which may be derived from the loss of ACK message.

Figure 13 shows an example of flow control operations for CoAP-SC. In the figure, it is assumed that the sending buffer can store the maximum of six data messages (buffer size = 6). The first ACK
message for flow control is generated by condition (a), when the receiver receives the second data message (SN = 2, AN = 0). The second ACK message is also generated by condition (a) for the data message (SN = 4, AN = 2), but it is lost. The third ACK message is generated by condition (b) for the data message (SN = 6, AN = 2). The ACK messages for flow control contain the same SN and AN values, as shown in the figure.

Figure 13 shows an example of flow control operations for CoAP-SC. In the figure, it is assumed that the sending buffer can store the maximum of six data messages (buffer size = 6). The first ACK message for flow control is generated by condition (a), when the receiver receives the second data message (SN = 2, AN = 0). The second ACK message is also generated by condition (a) for the data message (SN = 4, AN = 2), but it is lost. The third ACK message is generated by condition (b) for the data message (SN = 6, AN = 2).

In Figure 14, the control message (SN: 21, AN: 21) has been lost. In the case of TCP, ACK loss causes retransmission. However, in CoAP-SC, the sender does not need to retransmit. In the figure, when the receiver receives the control message (SN: 25, AN: 18), it checks that the sender’s buffer is more than 2/3 of fullness and then transmits a control message so as to flush the buffer, each time it receives a data message. In this figure, the control messages (SN:25, AN:25/SN:26, AN:26, SN:27, AN:27/SN:28, AN:28) were transmitted. The sender who receives control messages flushes the buffer and updates its AN value. After receiving the control message (SN: 28, AN: 28), it can be confirmed that the AN value has been updated in the PUT message (SN: 29, AN: 28).
3.4. CoAP Option for CoAP-SC

For CoAP-SC, we define the CoAP-SC option header, as shown in Figure 15, which includes 8-byte Option Delta, 4-byte SN, and 4-byte AN fields. All data messages and ACK messages used for error and flow controls will include this CoAP-SC option. In this Letter, the CoAP-SC option number is arbitrarily set to 100.

![Figure 15. CoAP-SC option header format.](image)

4. Performance Analysis by Experimentations

For performance analysis, the proposed CoAP-SC scheme is implemented and compared with the existing schemes. The existing CoAP over TCP and CoAP over UDP schemes are experimented by using the go-coap open source libraries [23]. The proposed CoAP-SC scheme is also implemented by using the go-coap, and the resulting source codes are publically distributed [24].

For experimentation, Raspberry Pi was used as a sender, and a general-purpose personal computer was used as a receiver. Figure 16 shows the testbed environment. The senders and receivers are connected via the access point (AP). The bandwidth between APs was set to 1 Mbps. In order to simulate packet losses, we generate a packet loss event by using a randomly generated number at the AP every second, as shown in Figure 17.
To evaluate overall performance of CoAP-SC in the network with error rate 0.1, we first compared the average transmission delays for the three candidate schemes. In this experiment, a client measures the temperature at each 500 ms intervals, and it sends the measured data (via 100 messages) to a server. In total, we performed 10 experiments and got the average delays.

Figure 18 shows the average delays required for transmission of 100 messages sequentially over 10 trials. In this figure, we can see that CoAP over UDP gives larger latency than CoAP over TCP. This is due to the disadvantage of CoAP over UDP, which is retransmitted when a timeout event occurs for a packet loss. In addition, we can see that the proposed CoAP-SC scheme provides lower delays than the existing two schemes. This is because CoAP-SC provides fast retransmission and also because the retransmissions by ACK loss can be reduced.
From now on, we conduct some more various experimentations to evaluate the performance of the proposed CoAP-SC scheme. For streaming transport, the sender transmits totally 600 data messages (N), with the payload size of 150 bytes, to the receiver. The time interval between the two consecutive data messages is set to 500 ms. On the other hand, the different packet error rates and buffer sizes are employed for performance evaluation. The packet error rates (P(E)) in the network are configured by 0–0.3 (30%), and the buffer size of the sender is ranged from 1 to 10 data messages.

For each experiment, the three performance metrics are measured: Number of Retransmitted Packets (NRP), Total Blocking Time (TBT) and Total Transmission Delay (TTD).

4.1. Number of Retransmitted Packets (NRP)

NRP represents the total number of data messages that have been retransmitted by the sender during data transmission. It is noted that NRP will depend on how effectively the flow control is performed in streaming transport. Usually, the retransmission occurs when a data packet is lost. The retransmission will also occur unnecessarily, if an ACK packet is lost. Note that the proposed scheme was designed by considering the ACK loss.

Figure 19 shows the NRP performance for different error rates. In the figure, we see that NRPs get larger, as the packet error rates increase for all candidate schemes. However, we note that the proposed CoAP-SC scheme gives smaller NRPs than the existing CoAP schemes. The gaps of performance get larger, as the error rates increase. This is because the proposed scheme performs the error handling and flow controls by considering the ACK loss, whereas the existing schemes tend to perform unnecessary retransmissions. In the case of existing schemes, if the ACK message of the data message is lost, the data message is also retransmitted. Among the existing schemes, the TCP-based CoAP provides better performance than the UDP-based CoAP. This is because the CoAP over TCP scheme supports cumulative ACK, and thus the number of retransmissions due to loss of ACK packet can be reduced, compared to the CoAP over UDP scheme. From the results, we note that the proposed CoAP-SC scheme utilizes the advantages of selective ACK as well as cumulative ACK through the streaming control with SN and AN. Overall, we can see that the proposed CoAP-SC scheme gives better performance than the existing two schemes.

Figure 18. Average transmission delays.
Figure 20 compares the NRP performance for different buffer sizes. The CoAP over UDP scheme provides larger NRPs than CoAP over TCP and CoAP-SC. It is noted that CoAP-SC gives better performance than CoAP over TCP for larger buffer sizes. This is because the proposed scheme can reduce unnecessary retransmissions by using the error handling and flow controls. It is noted that the proposed CoAP-SC scheme performs the flow control, based on the sending buffer. So, the number of ACK messages can be reduced, since the receiver will check the sender’s buffer status by using the SN and AN values. The receiver can also transmit a control message actively so as to flush the sender’s buffer.

![Graph showing the number of retransmitted packets for different error rates.](image)

**Figure 19.** Number of retransmitted packets for different error rates.

![Graph showing the number of retransmitted packets for different buffer sizes.](image)

**Figure 20.** Number of retransmitted packets for different buffer size.
4.2. Total Blocking Time (TBT)

TBT means the time duration in which the sending buffer is in the fullness state, during which further data transmissions will be blocked. It is noted that TBT will depend on how effectively the flow control is performed in streaming transport.

Figure 21 shows the TBTs of candidate schemes for different error rates. All candidate schemes provide almost the same TBTs for low error rates. However, the proposed CoAP-SC scheme gives better performance than the existing two schemes for high error rates. This is because the proposed scheme can perform the flow control effectively even in the lossy network environments. In the CoAP over UDP scheme, the flow control is not performed. So, all messages are removed from the buffer when the corresponding ACK message is received. However, CoAP over TCP provides cumulative ACK with fast retransmission. This makes it less sensitive to the ACK loss event, compared to the CoAP over UDP. In the meantime, CoAP-SC provides cumulative ACK and fast retransmission, as done in TCP. In addition, since the receiver transmits a control message according to the buffer status of the sender, it is helpful to reduce the blocking time.

![Figure 21. Total blocking time for different error rates.](image)

Figure 22 compares the TBTs of the candidate schemes for different buffer sizes. As the buffer size gets larger, TBTs tend to decrease for all candidate schemes. In the meantime, we see that the proposed CoAP-SC scheme gives the best performance among the three candidate schemes by using the effective flow control, when the buffer size is small.

![Figure 22. Total blocking time for different buffer sizes.](image)
4.3. Total Transmission Delay (TTD)

TTD means the time duration in which packets have been delivered successfully. It is noted that TTD depends on how effectively the flow control and error handling are performed in streaming transport.

Figures 23 and 24 show the TTDs of three candidate schemes for different error rates and buffer sizes, respectively. From the figures, we can see that the proposed scheme provides lower TTDs than the two existing schemes for all experiments. This performance gain comes from the error and flow controls of the proposed scheme. The gaps of performance get larger, as the error rate and buffer size increase in the network.

CoAP over UDP performs timeout-dependent retransmission when an error occurs, but CoAP over TCP reduces delay by performing ACK-based fast retransmission. CoAP-SC also performs the fast retransmission. However, unlike TCP, retransmissions will be reduced in CoAP-SC, because the receiver requests retransmission only when the next data message is received. This tends to give the best performance in terms of the total transmission delay for all messages.
5. Conclusions

In this paper, we proposed a CoAP streaming control scheme with error handling and flow control for IoT streaming transport. From the experimentation results, we see that the proposed scheme provides better throughput than the existing UDP-based CoAP and TCP-based CoAP.

Figure 23. Total transmission delays for different error rates.

Figure 24. Total transmission delays for different buffer sizes.
5. Conclusions

In this paper, we proposed a CoAP streaming control scheme with error handling and flow control for IoT streaming transport. From the experimentation results, we see that the proposed scheme provides better throughput than the existing UDP-based CoAP and TCP-based CoAP schemes. It seems that this performance gain comes from the streaming control operations based on the sequence number (SN), ACK number (AN), and the sending buffer. In conclusion, the existing CoAP/UDP and CoAP/TCP schemes can be used for reliable services. However, the proposed scheme may also be considered as a candidate scheme for real-time streaming services, in particular, in the IoT networks with data losses, with the management of the associated parameters, such as SN and AN, etc.

On the other hand, it seems that the proposed scheme still requires some more works to reduce the packet size for IoT environment. For further study, some methods need to be investigated, which include linear combinations of the data, erasure coding and header compression [25].

Author Contributions: J.-H.J. wrote the initial manuscript; M.G. revised the manuscript; S.-J.K. proofread the manuscript; All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Commercialization Promotion Agency For R&D Outcomes (COMPA) grant funded by the Korea government (MSIT) (2020-0003).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. Futur. Gener. Comput. Syst. 2013, 29, 1645–1660. [CrossRef]
2. Xu, L.D.; He, W.; Li, S.C. Internet of Things in Industries: A Survey. IEEE Trans. Ind. Inf. 2014, 10, 2233–2243. [CrossRef]
3. Kapoor, A.; Bhat, S.I.; Shidnal, S.; Mehra, A. Implementation of IoT (Internet of Things) and image processing in smart agriculture. In Proceedings of the 2016 International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS), Bangalore, India, 6–8 October 2016; pp. 21–26.
4. Shelby, Z.; Hartke, K.; Bormann, C. The Constrained Application Protocol (CoAP). IETF, RFC 7252, 2014. Available online: https://datatracker.ietf.org/doc/rfc7252/ (accessed on 23 June 2020).
5. Silva, J.d.C.; Rodrigues, J.J.P.C.; Al-Muhtadi, J.; Rabelo, R.A.L.; Furtado, V. Management Platforms and Protocols for Internet of Things: A Survey. Sensors 2019, 19, 676. [CrossRef]
6. Castellani, A.P.; Gheda, M.; Bui, N.; Rossi, M.; Zorzi, M. Web services for the Internet of Things through CoAP and EXI. In Proceedings of the 2011 IEEE International Conference on Communications Workshops (ICC 2001), Kyoto, Japan, 5–9 June 2011; pp. 22–32.
7. Castro, M.; Jara, A.J.; Skarmeta, A.F. Enabling end-to-end coap based communications for the web of things. J. Netw. Comput. Appl. 2016, 59, 230–236. [CrossRef]
8. Khattak, H.A.; Ruta, M.; Di Sciascio, E. CoAP-based healthcare sensor networks: A survey. In Proceedings of the 2014 11th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Islamabad, Pakistan, 14–18 January 2014; pp. 499–503.
9. Van den Abeele, F.; Moerman, I.; Demeester, P.; Hoebbeke, J. Secure Service Proxy: A CoAP (s) Intermediary for a Securer and Smarter Web of Things. Sensors 2017, 17, 1609. [CrossRef]
10. Hartke, K. Observing Resources in the Constrained Application Protocol (CoAP). IETF, RFC 7641, 2015. Available online: https://tools.ietf.org/html/rfc7641 (accessed on 23 June 2020).
11. Bormann, C.; Shelby, Z. Block-Wise Transfers in the Constrained Application Protocol (CoAP). IETF, RFC 7959, 2016. Available online: https://tools.ietf.org/html/rfc7959 (accessed on 23 June 2020).
12. Choi, G.; Kim, D.; Yeom, J. Efficient Streaming over CoAP. In Proceedings of the 2016 International Conference on Information Networking (ICOIN), Kota Kinabalu, Malaysia, 13–15 January 2016; pp. 476–478.
13. Bormann, C.; Lemay, S.; Tschofenig, H.; Hartke, K.; Silverajan, B.; Raymor, B. CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets. IETF, RFC 8323, 2018. Available online: https://tools.ietf.org/html/rfc8323 (accessed on 23 June 2020).
14. Jarvinen, I.; Daniel, L.; Kojo, M. Experimental evaluation of alternative congestion control algorithms for Constrained Application Protocol (CoAP). In Proceedings of the Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum, Milan, Italy, 14–16 December 2015; pp. 453–458.
15. Postel, J. Transmission Control Protocol. JETF, RFC 793, 1981. Available online: https://tools.ietf.org/html/rfc793 (accessed on 23 June 2020).
16. Feizi, S.; Lucani, D.E.; Sorensen, C.W.; Makhdoumi, A.; Medard, M. Tunable sparse network coding for multicast networks. In Proceedings of the 2014 International Symposium on In Network Coding (NetCod), Aalborg Oest, Denmark, 27–28 June 2014; pp. 1–6.
17. Gligoroski, D.; Kralevska, K. Families of Optimal Binary Non-MDS Erasure Codes. In Proceedings of the 2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, 29 June–4 July 2014; pp. 3150–3154.
18. Biczók, G.; Chen, Y.; Kralevska, K.; Øverby, H. Combining forward error correction and network coding in bufferless networks: A case study for optical packet switching. In Proceedings of the 2016 IEEE 17th International Conference on High Performance Switching and Routing, Yokohama, Japan, 14–17 June 2016; pp. 61–68.
19. Zhang, J.; Ren, F.; Tang, L.; Lin, C. Modeling and solving TCP Incast problem in data center networks. IEEE Trans. Parallel Distrib. Syst. 2015, 26, 478–491. [CrossRef]
20. Zhang, M.; Mezzavilla, M.; Ford, R.; Rangan, S.; Panwar, S.; Mellios, E.; Kong, D.; Nix, A.; Zorzi, M. Transport layer performance in 5G mmWave cellular. In Proceedings of the 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), San Francisco, CA, USA, 10–15 April 2016; pp. 730–735.
21. Zhang, M.; Polese, M.; Mezzavilla, M.; Zhu, J.; Rangan, S.; Panwar, S.; Zorzi, M. Will TCP work in mmWave 5G cellular networks? IEEE Commun. Mag. 2019, 57, 65–71. [CrossRef]
22. Polese, M.; Chiariotti, F.; Bonetto, E.; Rigotto, F.; Zanella, A.; Zorzi, M. A survey on recent advances in transport layer protocols. IEEE Commun. Surv. Tuts. 2019, 21, 3584–3608. [CrossRef]
23. Go-Coap. Available online: https://github.com/go-ocf/go-coap (accessed on 14 August 2020).
24. Coap-Sc.Golang. Available online: https://github.com/Goopu/coap-sc.golang (accessed on 14 August 2020).
25. Gligoroski, D.; Kralevska, K.; Øverby, H. Minimal header overhead for random linear network coding. In Proceedings of the 2015 IEEE International Conference on Communication Workshop (ICCW), London, UK, 8–12 June 2015; pp. 680–685.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).