electronics

Article

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.

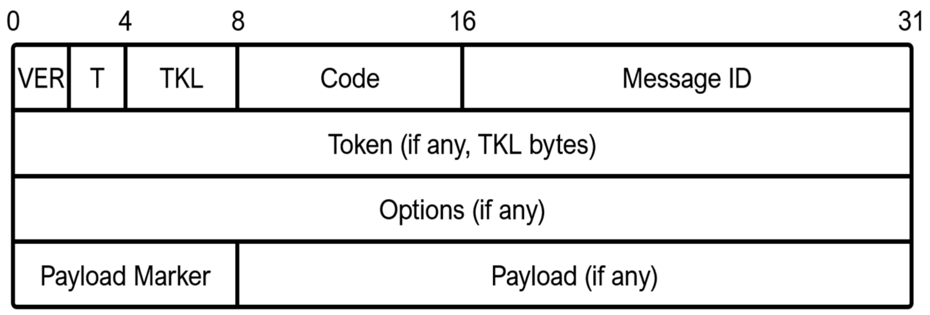
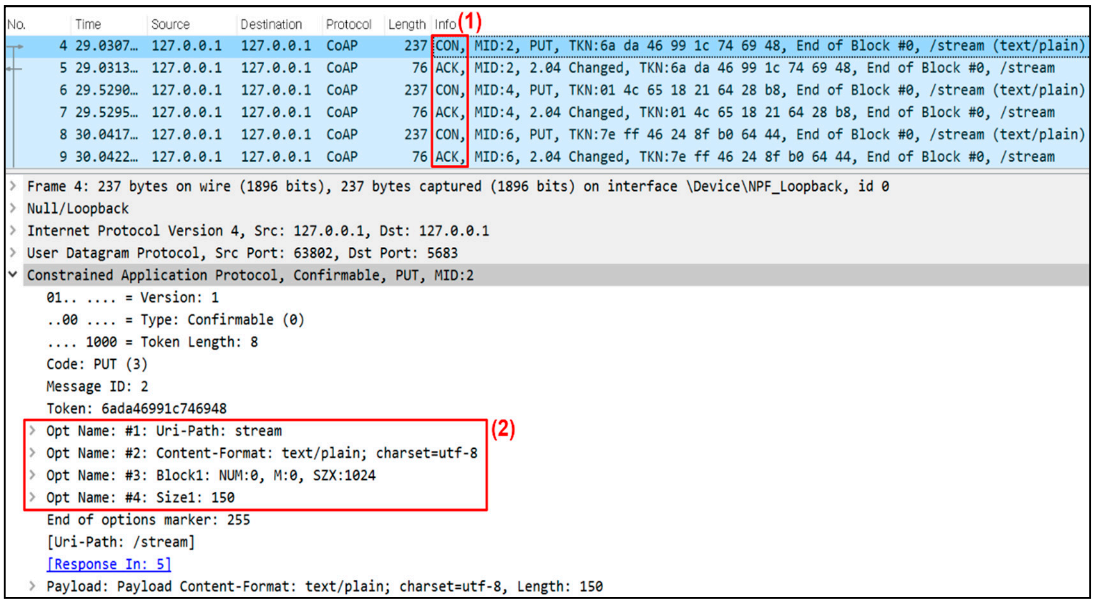


Figure 1. CoAP over UDP header format.

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.

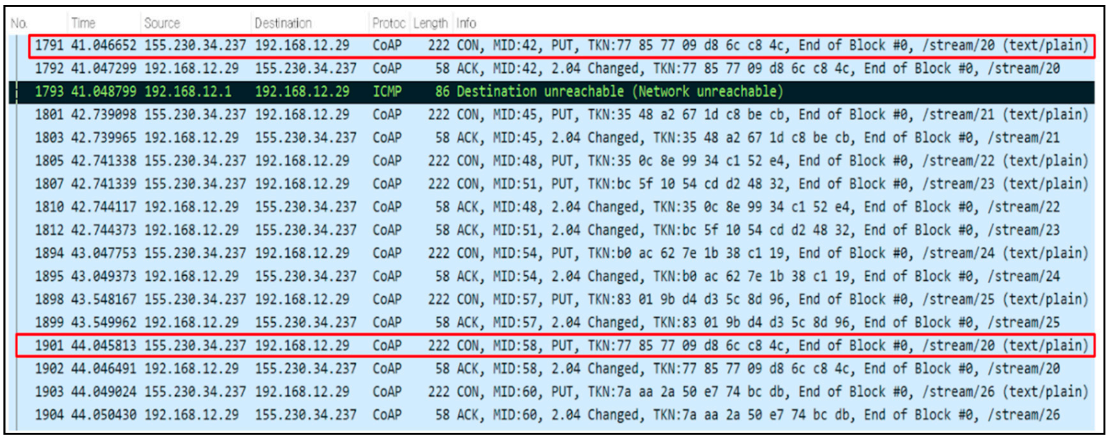
Figure 2 shows the error handling mechanism of CoAP over UDP. In step (1), the ACK messages are transmitted for all CON messages. This mechanism may create unnecessary ACKs. As you can also see in step (2), this protocol does not have any field for sequence number.



```
No.   Time       Source     Destination  Protocol  Length  Info (1)
  4 29.0307…  127.0.0.1  127.0.0.1    CoAP         237  CON, MID:2, PUT, TKN:6a da 46 99 1c 74 69 48, End of Block #0, /stream (text/plain)
  5 29.0313…  127.0.0.1  127.0.0.1    CoAP          76  ACK, MID:2, 2.04 Changed, TKN:6a da 46 99 1c 74 69 48, End of Block #0, /stream
  6 29.5290…  127.0.0.1  127.0.0.1    CoAP         237  CON, MID:4, PUT, TKN:01 4c 65 18 21 64 28 b8, End of Block #0, /stream (text/plain)
  7 29.5295…  127.0.0.1  127.0.0.1    CoAP          76  ACK, MID:4, 2.04 Changed, TKN:01 4c 65 18 21 64 28 b8, End of Block #0, /stream
  8 30.0417…  127.0.0.1  127.0.0.1    CoAP         237  CON, MID:6, PUT, TKN:7e ff 46 24 8f b0 64 44, End of Block #0, /stream (text/plain)
  9 30.0422…  127.0.0.1  127.0.0.1    CoAP          76  ACK, MID:6, 2.04 Changed, TKN:7e ff 46 24 8f b0 64 44, End of Block #0, /stream

> Frame 4: 237 bytes on wire (1896 bits), 237 bytes captured (1896 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 63802, Dst Port: 5683
v Constrained Application Protocol, Confirmable, PUT, MID:2
    01.. .... = Version: 1
    ..00 .... = Type: Confirmable (0)
    .... 1000 = Token Length: 8
    Code: PUT (3)
    Message ID: 2
    Token: 6ada46991c746948
  > Opt Name: #1: Uri-Path: stream                              (2)
  > Opt Name: #2: Content-Format: text/plain; charset=utf-8
  > Opt Name: #3: Block1: NUM:0, M:0, SZX:1024
  > Opt Name: #4: Size1: 150
    End of options marker: 255
    [Uri-Path: /stream]
    [Response In: 5]
  > Payload: Payload Content-Format: text/plain; charset=utf-8, Length: 150
```

Figure 2. Packet analysis for CoAP over UDP.

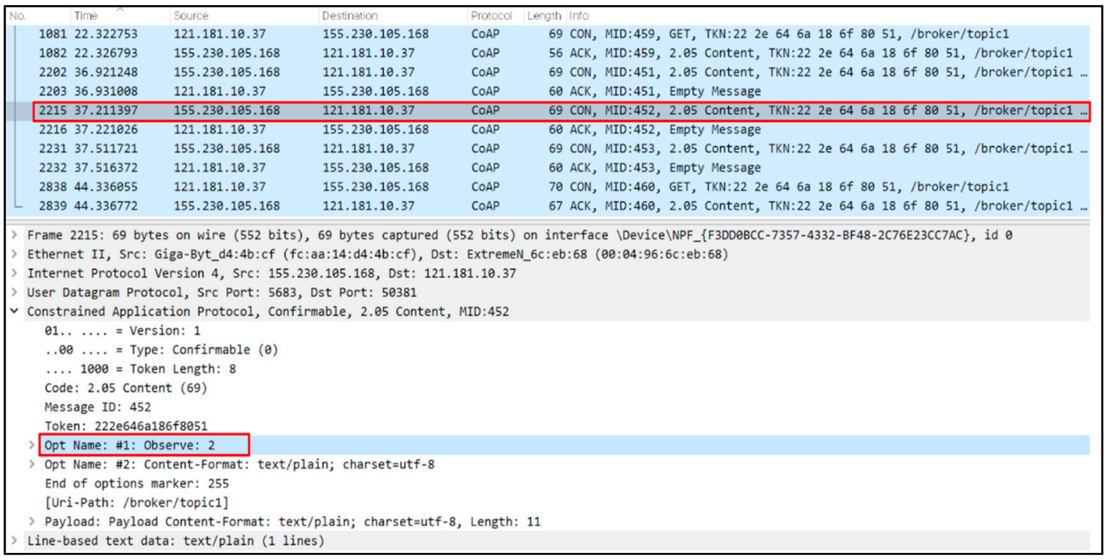
Figure 3 shows the error handling scenario for CoAP over UDP. In this figure, ACK message for PUT message (/stream/20) has been lost and the PUT message is retransmitted after the PUT message (/stream/25) is sent. In addition, CoAP over UDP retransmits data messages if a timeout event occurs. Thus, it may give poor performance in case the messages must be processed sequentially.



```
No.  Time      Source         Destination    Protoc Length Info
1791 41.046652 155.230.34.237 192.168.12.29  CoAP   222 CON, MID:42, PUT, TKN:77 85 77 09 d8 6c c8 4c, End of Block #0, /stream/20 (text/plain)
1792 41.047299 192.168.12.29  155.230.34.237 CoAP    58 ACK, MID:42, 2.04 Changed, TKN:77 85 77 09 d8 6c c8 4c, End of Block #0, /stream/20
1793 41.048799 192.168.12.1   192.168.12.29  ICMP    86 Destination unreachable (Network unreachable)
1801 42.739098 155.230.34.237 192.168.12.29  CoAP   222 CON, MID:45, PUT, TKN:35 48 a2 67 1d c8 be cb, End of Block #0, /stream/21 (text/plain)
1803 42.739965 192.168.12.29  155.230.34.237 CoAP    58 ACK, MID:45, 2.04 Changed, TKN:35 48 a2 67 1d c8 be cb, End of Block #0, /stream/21
1805 42.741338 155.230.34.237 192.168.12.29  CoAP   222 CON, MID:48, PUT, TKN:35 0c 8e 99 34 c1 52 e4, End of Block #0, /stream/22 (text/plain)
1807 42.741339 155.230.34.237 192.168.12.29  CoAP   222 CON, MID:51, PUT, TKN:bc 5f 10 54 cd d2 48 32, End of Block #0, /stream/23 (text/plain)
1810 42.744117 192.168.12.29  155.230.34.237 CoAP    58 ACK, MID:48, 2.04 Changed, TKN:35 0c 8e 99 34 c1 52 e4, End of Block #0, /stream/22
1812 42.744373 192.168.12.29  155.230.34.237 CoAP    58 ACK, MID:51, 2.04 Changed, TKN:bc 5f 10 54 cd d2 48 32, End of Block #0, /stream/23
1894 43.047753 155.230.34.237 192.168.12.29  CoAP   222 CON, MID:54, PUT, TKN:b0 ac 62 7e 1b 38 c1 19, End of Block #0, /stream/24 (text/plain)
1895 43.049373 192.168.12.29  155.230.34.237 CoAP    58 ACK, MID:54, 2.04 Changed, TKN:b0 ac 62 7e 1b 38 c1 19, End of Block #0, /stream/24
1898 43.548167 155.230.34.237 192.168.12.29  CoAP   222 CON, MID:57, PUT, TKN:83 01 9b d4 d3 5c 8d 96, End of Block #0, /stream/25 (text/plain)
1899 43.549962 192.168.12.29  155.230.34.237 CoAP    58 ACK, MID:57, 2.04 Changed, TKN:83 01 9b d4 d3 5c 8d 96, End of Block #0, /stream/25
1901 44.045813 155.230.34.237 192.168.12.29  CoAP   222 CON, MID:58, PUT, TKN:77 85 77 09 d8 6c c8 4c, End of Block #0, /stream/20 (text/plain)
1902 44.046491 192.168.12.29  155.230.34.237 CoAP    58 ACK, MID:58, 2.04 Changed, TKN:77 85 77 09 d8 6c c8 4c, End of Block #0, /stream/20
1903 44.049024 155.230.34.237 192.168.12.29  CoAP   222 CON, MID:60, PUT, TKN:7a aa 2a 50 e7 74 bc db, End of Block #0, /stream/26 (text/plain)
1904 44.050430 192.168.12.29  155.230.34.237 CoAP    58 ACK, MID:60, 2.04 Changed, TKN:7a aa 2a 50 e7 74 bc db, End of Block #0, /stream/26
```

Figure 3. Packet analysis for error handling of CoAP over UDP.

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 performance 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.



```
No.      Time         Source            Destination       Protocol  Length Info
   1081 22.322753    121.181.10.37     155.230.105.168   CoAP        69 CON, MID:459, GET, TKN:22 2e 64 6a 18 6f 80 51, /broker/topic1
   1082 22.326793    155.230.105.168   121.181.10.37     CoAP        56 ACK, MID:459, 2.05 Content, TKN:22 2e 64 6a 18 6f 80 51, /broker/topic1
   2202 36.921248    155.230.105.168   121.181.10.37     CoAP        69 CON, MID:451, 2.05 Content, TKN:22 2e 64 6a 18 6f 80 51, /broker/topic1 …
   2203 36.931008    121.181.10.37     155.230.105.168   CoAP        60 ACK, MID:451, Empty Message
   2215 37.211397    155.230.105.168   121.181.10.37     CoAP        69 CON, MID:452, 2.05 Content, TKN:22 2e 64 6a 18 6f 80 51, /broker/topic1 …
   2216 37.221026    121.181.10.37     155.230.105.168   CoAP        60 ACK, MID:452, Empty Message
   2231 37.511721    155.230.105.168   121.181.10.37     CoAP        69 CON, MID:453, 2.05 Content, TKN:22 2e 64 6a 18 6f 80 51, /broker/topic1 …
   2232 37.516372    121.181.10.37     155.230.105.168   CoAP        60 ACK, MID:453, Empty Message
   2838 44.336055    121.181.10.37     155.230.105.168   CoAP        70 CON, MID:460, GET, TKN:22 2e 64 6a 18 6f 80 51, /broker/topic1
   2839 44.336772    155.230.105.168   121.181.10.37     CoAP        67 ACK, MID:460, 2.05 Content, TKN:22 2e 64 6a 18 6f 80 51, /broker/topic1 …

> Frame 2215: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface \Device\NPF_{F3DD0BCC-7357-4332-BF48-2C76E23CC7AC}, id 0
> Ethernet II, Src: Giga-Byt_d4:4b:cf (fc:aa:14:d4:4b:cf), Dst: ExtremeN_6c:eb:68 (00:04:96:6c:eb:68)
> Internet Protocol Version 4, Src: 155.230.105.168, Dst: 121.181.10.37
> User Datagram Protocol, Src Port: 5683, Dst Port: 50381
v Constrained Application Protocol, Confirmable, 2.05 Content, MID:452
    01.. .... = Version: 1
    ..00 .... = Type: Confirmable (0)
    .... 1000 = Token Length: 8
    Code: 2.05 Content (69)
    Message ID: 452
    Token: 222e646a186f8051
  > Opt Name: #1: Observe: 2
  > Opt Name: #2: Content-Format: text/plain; charset=utf-8
    End of options marker: 255
    [Uri-Path: /broker/topic1]
  > Payload: Payload Content-Format: text/plain; charset=utf-8, Length: 11
> Line-based text data: text/plain (1 lines)
```

Figure 4. Packet analysis for CoAP-Observe in CoAP over UDP.

It is noted that the existing CoAP/UDP scheme gives a simple CON/ACK mechanism for reliability, whereas this will paper proposes a more elaborated streaming control mechanism by considering the sequence number of data message, the use of ACK messages, and the buffer size of the sending buffer. This will be helpful to give better performance in the networks with data losses.

2.2. *CoAP over TCP*

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.

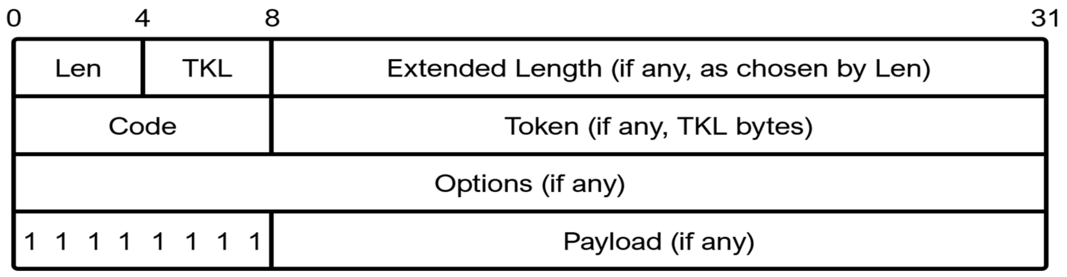
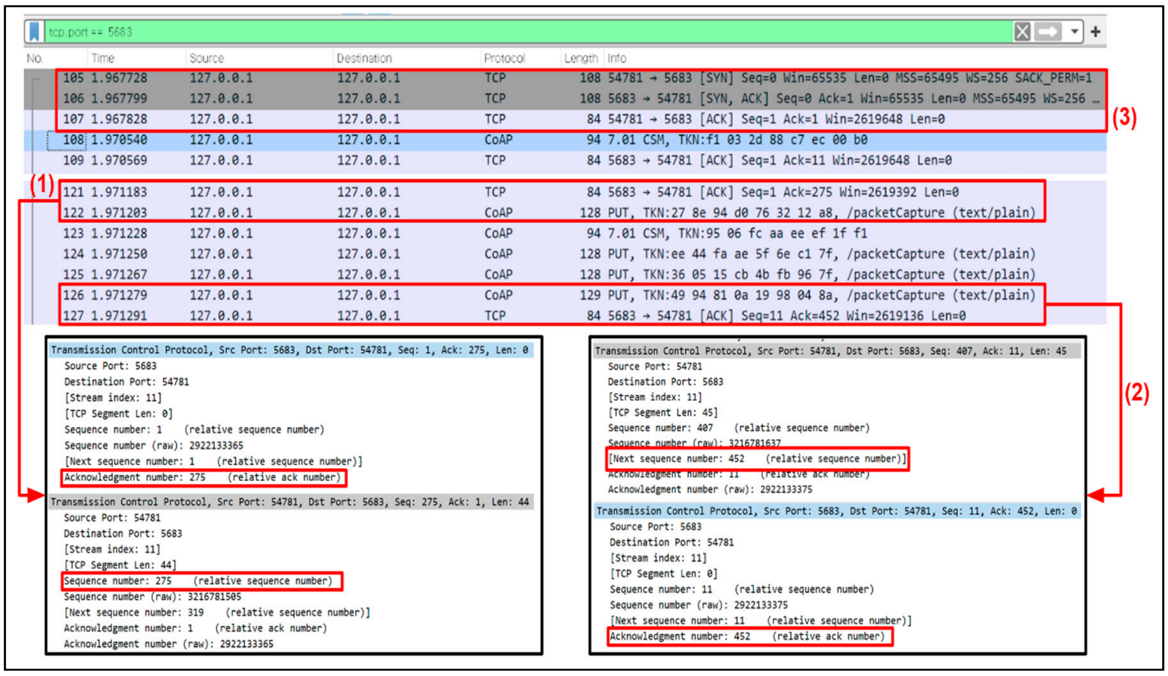

0
4
8
31
Len
TKL
Extended Length (if any, as chosen by Len)
Code
Token (if any, TKL bytes)
Options (if any)
1 1 1 1 1 1 1 1
Payload (if any)

Figure 5. 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.



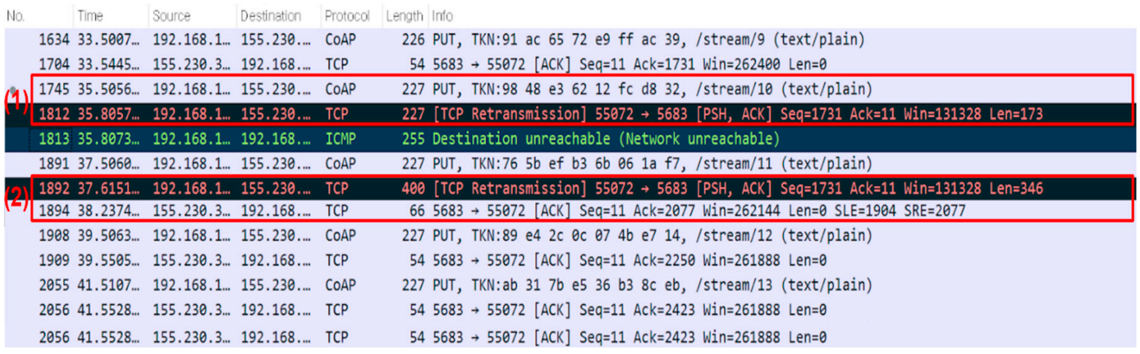
```
tcp.port == 5683

No.  Time      Source     Destination  Protocol  Length Info
105 1.967728   127.0.0.1  127.0.0.1    TCP       108 54781 → 5683 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
106 1.967799   127.0.0.1  127.0.0.1    TCP       108 5683 → 54781 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 …
107 1.967828   127.0.0.1  127.0.0.1    TCP        84 54781 → 5683 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
108 1.970540   127.0.0.1  127.0.0.1    CoAP       94 7.01 CSM, TKN:f1 03 2d 88 c7 ec 00 b0
109 1.970569   127.0.0.1  127.0.0.1    TCP        84 5683 → 54781 [ACK] Seq=1 Ack=11 Win=2619648 Len=0

121 1.971183   127.0.0.1  127.0.0.1    TCP        84 5683 → 54781 [ACK] Seq=1 Ack=275 Win=2619392 Len=0
122 1.971203   127.0.0.1  127.0.0.1    CoAP      128 PUT, TKN:27 8e 94 d0 76 32 12 a8, /packetCapture (text/plain)
123 1.971228   127.0.0.1  127.0.0.1    CoAP       94 7.01 CSM, TKN:95 06 fc aa ee ef 1f f1
124 1.971250   127.0.0.1  127.0.0.1    CoAP      128 PUT, TKN:ee 44 fa ae 5f 6e c1 7f, /packetCapture (text/plain)
125 1.971267   127.0.0.1  127.0.0.1    CoAP      128 PUT, TKN:36 05 15 cb 4b fb 96 7f, /packetCapture (text/plain)
126 1.971279   127.0.0.1  127.0.0.1    CoAP      129 PUT, TKN:49 94 81 0a 19 98 04 8a, /packetCapture (text/plain)
127 1.971291   127.0.0.1  127.0.0.1    TCP        84 5683 → 54781 [ACK] Seq=11 Ack=452 Win=2619136 Len=0
```

```
Transmission Control Protocol, Src Port: 5683, Dst Port: 54781, Seq: 1, Ack: 275, Len: 0
   Source Port: 5683
   Destination Port: 54781
   [Stream index: 11]
   [TCP Segment Len: 0]
   Sequence number: 1    (relative sequence number)
   Sequence number (raw): 2922133365
   [Next sequence number: 1    (relative sequence number)]
   Acknowledgment number: 275    (relative ack number)
Transmission Control Protocol, Src Port: 54781, Dst Port: 5683, Seq: 275, Ack: 1, Len: 44
   Source Port: 54781
   Destination Port: 5683
   [Stream index: 11]
   [TCP Segment Len: 44]
   Sequence number: 275    (relative sequence number)
   Sequence number (raw): 3216781505
   [Next sequence number: 319    (relative sequence number)]
   Acknowledgment number: 1    (relative ack number)
   Acknowledgment number (raw): 2922133365
```

```
Transmission Control Protocol, Src Port: 54781, Dst Port: 5683, Seq: 407, Ack: 11, Len: 45
   Source Port: 54781
   Destination Port: 5683
   [Stream index: 11]
   [TCP Segment Len: 45]
   Sequence number: 407    (relative sequence number)
   Sequence number (raw): 3216781637
   [Next sequence number: 452    (relative sequence number)]
   Acknowledgment number: 11    (relative ack number)
   Acknowledgment number (raw): 2922133375
Transmission Control Protocol, Src Port: 5683, Dst Port: 54781, Seq: 11, Ack: 452, Len: 0
   Source Port: 5683
   Destination Port: 54781
   [Stream index: 11]
   [TCP Segment Len: 0]
   Sequence number: 11    (relative sequence number)
   Sequence number (raw): 2922133375
   [Next sequence number: 11    (relative sequence number)]
   Acknowledgment number: 452    (relative ack number)
```

Figure 6. Packet analysis for CoAP over TCP transmission.

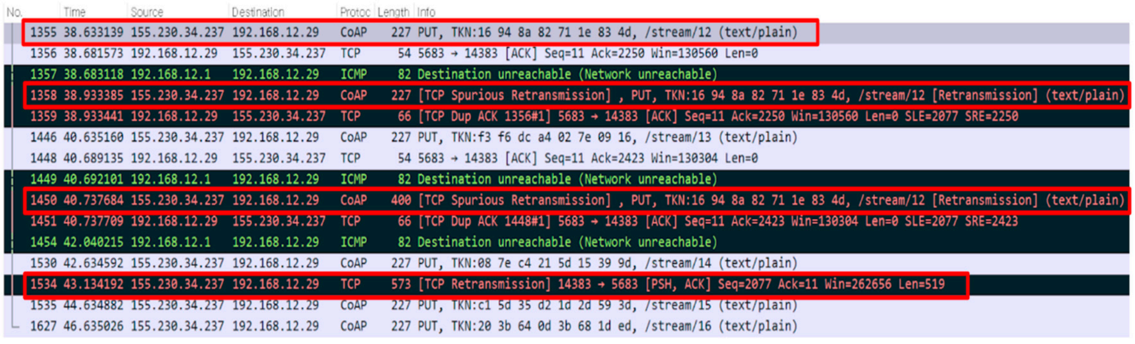
Figure 7 shows the error handling mechanism of CoAP over TCP. In step (1), the PUT message (/stream/10) has been lost, and a retransmission message was immediately sent by the fast retransmission algorithm of TCP. In addition, TCP may reduce the number of retransmissions through linear combinations of 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.



```
No.   Time       Source      Destination  Protocol  Length Info
1634 33.5007…  192.168.1…  155.230.…  CoAP   226 PUT, TKN:91 ac 65 72 e9 ff ac 39, /stream/9 (text/plain)
1704 33.5445…  155.230.3…  192.168.…  TCP     54 5683 → 55072 [ACK] Seq=11 Ack=1731 Win=262400 Len=0
1745 35.5056…  192.168.1…  155.230.…  CoAP   227 PUT, TKN:98 48 e3 62 12 fc d8 32, /stream/10 (text/plain)
1812 35.8057…  192.168.1…  155.230.…  TCP    227 [TCP Retransmission] 55072 → 5683 [PSH, ACK] Seq=1731 Ack=11 Win=131328 Len=173
1813 35.8073…  192.168.1…  192.168.…  ICMP   255 Destination unreachable (Network unreachable)
1891 37.5060…  192.168.1…  155.230.…  CoAP   227 PUT, TKN:76 5b ef b3 6b 06 1a f7, /stream/11 (text/plain)
1892 37.6151…  192.168.1…  155.230.…  TCP    400 [TCP Retransmission] 55072 → 5683 [PSH, ACK] Seq=1731 Ack=11 Win=131328 Len=346
1894 38.2374…  155.230.3…  192.168.…  TCP     66 5683 → 55072 [ACK] Seq=11 Ack=2077 Win=262144 Len=0 SLE=1904 SRE=2077
1908 39.5063…  192.168.1…  155.230.…  CoAP   227 PUT, TKN:89 e4 2c 0c 07 4b e7 14, /stream/12 (text/plain)
1909 39.5505…  155.230.3…  192.168.…  TCP     54 5683 → 55072 [ACK] Seq=11 Ack=2250 Win=261888 Len=0
2055 41.5107…  192.168.1…  155.230.…  CoAP   227 PUT, TKN:ab 31 7b e5 36 b3 8c eb, /stream/13 (text/plain)
2056 41.5528…  155.230.3…  192.168.…  TCP     54 5683 → 55072 [ACK] Seq=11 Ack=2423 Win=261888 Len=0
2056 41.5528…  155.230.3…  192.168.…  TCP     54 5683 → 55072 [ACK] Seq=11 Ack=2423 Win=261888 Len=0
```

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

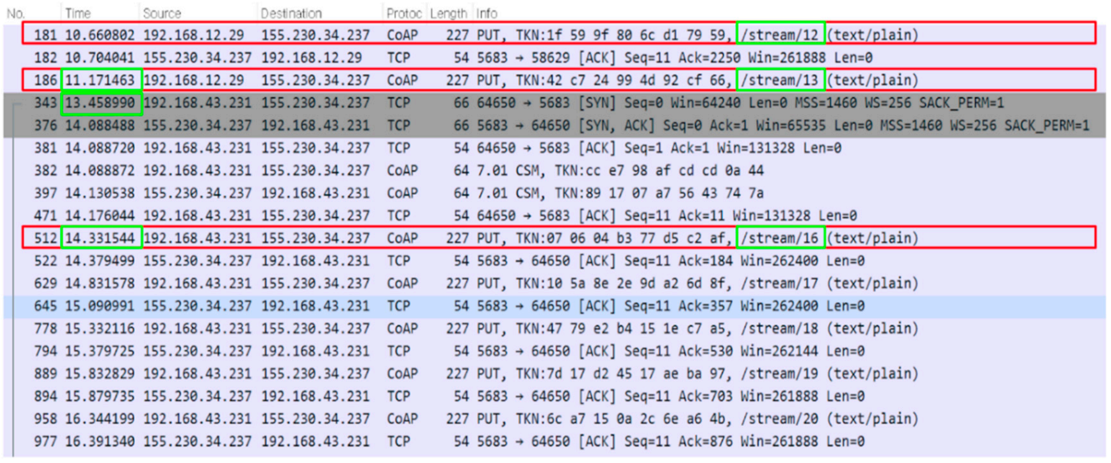
The ACK message of TCP can reduce the retransmission delay. However, it may increase the number of retransmissions due to the loss of the ACK message. Figure 8 illustrates this situation. In this figure, the PUT message (/stream/12) has been retransmitted because of ACK loss.



```
No.  Time      Source         Destination    Protoc Length Info
1355 38.633139 155.230.34.237 192.168.12.29  CoAP   227 PUT, TKN:16 94 8a 82 71 1e 83 4d, /stream/12 (text/plain)
1356 38.681573 192.168.12.29  155.230.34.237 TCP     54 5683 → 14383 [ACK] Seq=11 Ack=2250 Win=130560 Len=0
1357 38.683118 192.168.12.1   192.168.12.29  ICMP    82 Destination unreachable (Network unreachable)
1358 38.933385 155.230.34.237 192.168.12.29  CoAP   227 [TCP Spurious Retransmission] , PUT, TKN:16 94 8a 82 71 1e 83 4d, /stream/12 [Retransmission] (text/plain)
1359 38.933441 192.168.12.29  155.230.34.237 TCP     66 [TCP Dup ACK 1356#1] 5683 → 14383 [ACK] Seq=11 Ack=2250 Win=130560 Len=0 SLE=2077 SRE=2250
1446 40.635160 155.230.34.237 192.168.12.29  CoAP   227 PUT, TKN:f3 f6 dc a4 02 7e 09 16, /stream/13 (text/plain)
1448 40.689135 192.168.12.29  155.230.34.237 TCP     54 5683 → 14383 [ACK] Seq=11 Ack=2423 Win=130304 Len=0
1449 40.692101 192.168.12.1   192.168.12.29  ICMP    82 Destination unreachable (Network unreachable)
1450 40.737684 155.230.34.237 192.168.12.29  CoAP   400 [TCP Spurious Retransmission] , PUT, TKN:16 94 8a 82 71 1e 83 4d, /stream/12 [Retransmission] (text/plain)
1451 40.737709 192.168.12.29  155.230.34.237 TCP     66 [TCP Dup ACK 1448#1] 5683 → 14383 [ACK] Seq=11 Ack=2423 Win=130304 Len=0 SLE=2077 SRE=2423
1454 42.040215 192.168.12.1   192.168.12.29  ICMP    82 Destination unreachable (Network unreachable)
1530 42.634592 155.230.34.237 192.168.12.29  CoAP   227 PUT, TKN:08 7e c4 21 5d 15 39 9d, /stream/14 (text/plain)
1534 43.134192 155.230.34.237 192.168.12.29  TCP    573 [TCP Retransmission] 14383 → 5683 [PSH, ACK] Seq=2077 Ack=11 Win=262656 Len=519
1535 44.634882 155.230.34.237 192.168.12.29  CoAP   227 PUT, TKN:c1 5d 35 d2 1d 2d 59 3d, /stream/15 (text/plain)
1627 46.635026 155.230.34.237 192.168.12.29  CoAP   227 PUT, TKN:20 3b 64 0d 3b 68 1d ed, /stream/16 (text/plain)
```

Figure 8. Drawback of TCP error handling.

Moreover, the TCP three-way handshake mechanism will not be suitable for streaming transport of real-time IoT data in mobile networks, in which many re-connections may occur. In Figure 9, the sender is reconnected to the receiver due to a handover. At this time, a delay of about 500 ms occurred due to the three-way handshake process. In addition, we can see that the transmitted PUT message (/stream/14,/stream/15) was lost, until the sender confirmed that the connection was disconnected. TCP provides reliability during the connection, but it does not provide reliability when reconnection occurs. 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.



```
No. Time      Source         Destination    Protoc Length Info
181 10.660802 192.168.12.29  155.230.34.237 CoAP   227 PUT, TKN:1f 59 9f 80 6c d1 79 59, /stream/12 (text/plain)
182 10.704041 155.230.34.237 192.168.12.29  TCP     54 5683 → 58629 [ACK] Seq=11 Ack=2250 Win=261888 Len=0
186 11.171463 192.168.12.29  155.230.34.237 CoAP   227 PUT, TKN:42 c7 24 99 4d 92 cf 66, /stream/13 (text/plain)
343 13.458990 192.168.43.231 155.230.34.237 TCP     66 64650 → 5683 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
376 14.088488 155.230.34.237 192.168.43.231 TCP     66 5683 → 64650 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
381 14.088720 192.168.43.231 155.230.34.237 TCP     54 64650 → 5683 [ACK] Seq=1 Ack=1 Win=131328 Len=0
382 14.088872 192.168.43.231 155.230.34.237 CoAP    64 7.01 CSM, TKN:cc e7 98 af cd cd 0a 44
397 14.130538 155.230.34.237 192.168.43.231 CoAP    64 7.01 CSM, TKN:89 17 07 a7 56 43 74 7a
471 14.176044 192.168.43.231 155.230.34.237 TCP     54 64650 → 5683 [ACK] Seq=11 Ack=11 Win=131328 Len=0
512 14.331544 192.168.43.231 155.230.34.237 CoAP   227 PUT, TKN:07 06 04 b3 77 d5 c2 af, /stream/16 (text/plain)
522 14.379499 155.230.34.237 192.168.43.231 TCP     54 5683 → 64650 [ACK] Seq=11 Ack=184 Win=262400 Len=0
629 14.831578 192.168.43.231 155.230.34.237 CoAP   227 PUT, TKN:10 5a 8e 2e 9d a2 6d 8f, /stream/17 (text/plain)
645 15.090991 155.230.34.237 192.168.43.231 TCP     54 5683 → 64650 [ACK] Seq=11 Ack=357 Win=262400 Len=0
778 15.332116 192.168.43.231 155.230.34.237 CoAP   227 PUT, TKN:47 79 e2 b4 15 1e c7 a5, /stream/18 (text/plain)
794 15.379725 155.230.34.237 192.168.43.231 TCP     54 5683 → 64650 [ACK] Seq=11 Ack=530 Win=262144 Len=0
889 15.832829 192.168.43.231 155.230.34.237 CoAP   227 PUT, TKN:7d 17 d2 45 17 ae ba 97, /stream/19 (text/plain)
894 15.879735 155.230.34.237 192.168.43.231 TCP     54 5683 → 64650 [ACK] Seq=11 Ack=703 Win=261888 Len=0
958 16.344199 192.168.43.231 155.230.34.237 CoAP   227 PUT, TKN:6c a7 15 0a 2c 6e a6 4b, /stream/20 (text/plain)
977 16.391340 155.230.34.237 192.168.43.231 TCP     54 5683 → 64650 [ACK] Seq=11 Ack=876 Win=261888 Len=0
```

Figure 9. Drawback of TCP three-way handshaking mechanism.

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 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.

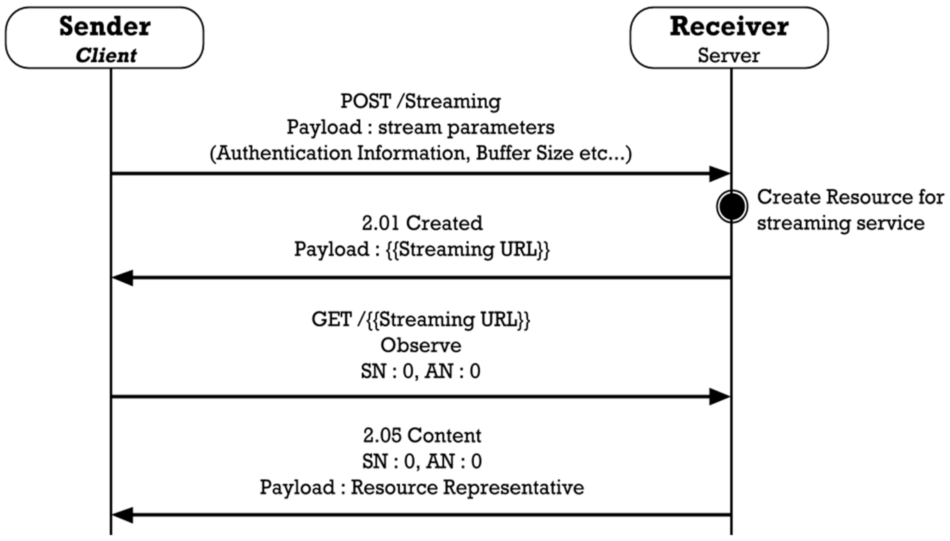

Sender
Client
Receiver
Server
POST /Streaming
Payload : stream parameters
(Authentication Information, Buffer Size etc...)
Create Resource for streaming service
2.01 Created
Payload : {{Streaming URL}}
GET /{{Streaming URL}}
Observe
SN : 0, AN : 0
2.05 Content
SN : 0, AN : 0
Payload : Resource Representative

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:

$$\text{dataMsg.SN} - \text{receiver.AN} > 1$$

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.

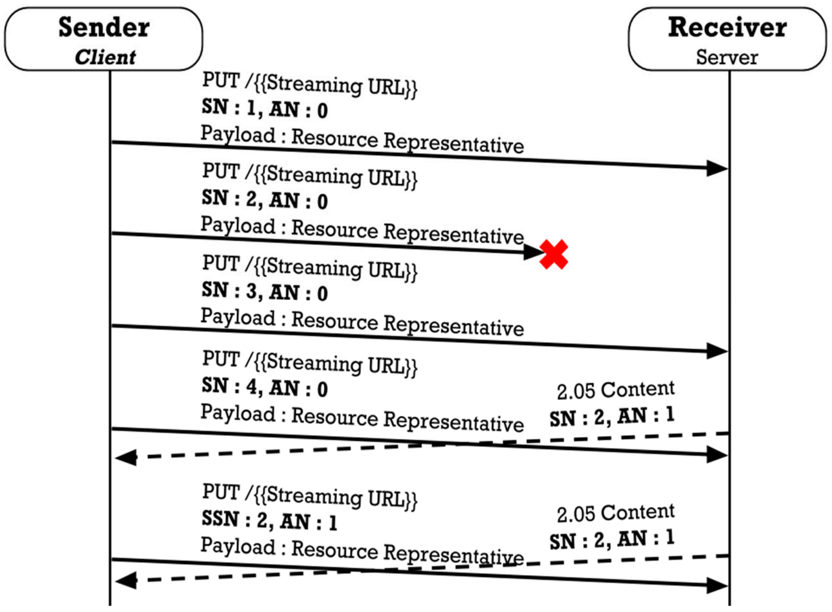
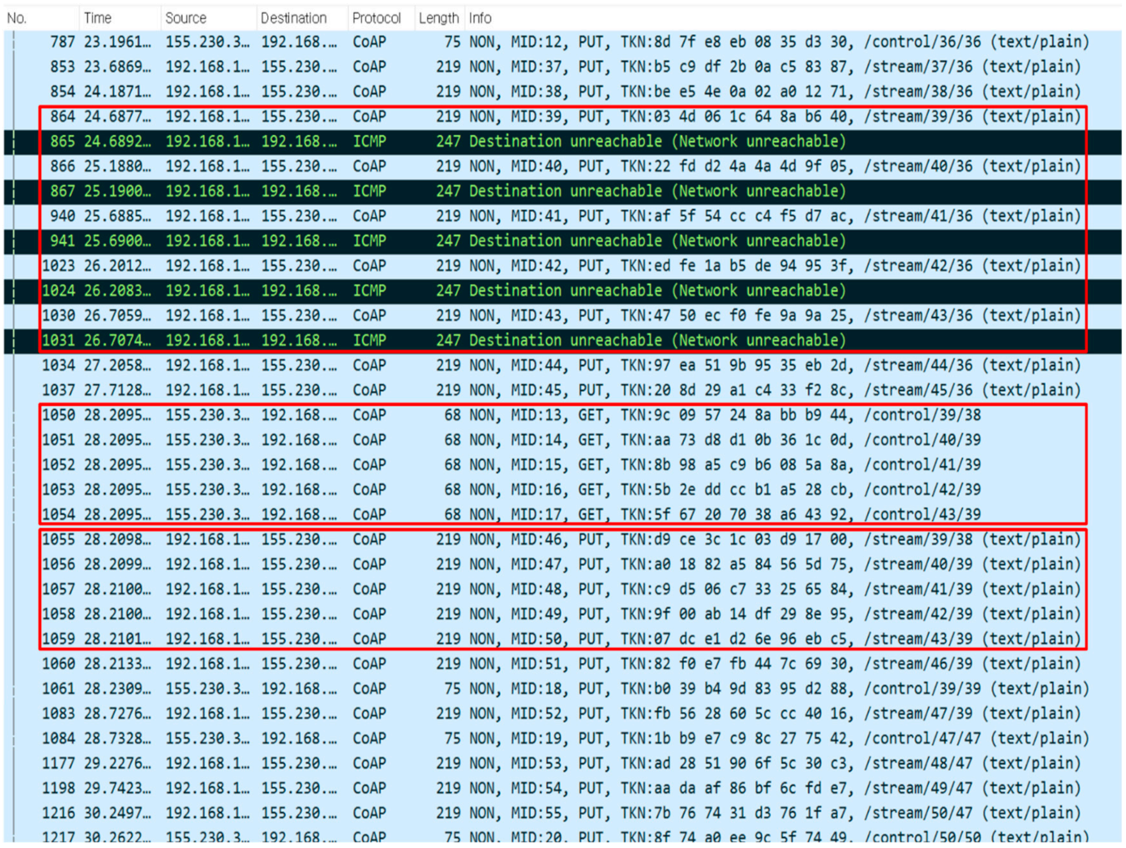


Figure 11. Example of error handling operations for CoAP-SC.

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.



```
No.    Time        Source        Destination   Protocol Length Info
  787 23.1961…  155.230.3…  192.168.…  CoAP      75 NON, MID:12, PUT, TKN:8d 7f e8 eb 08 35 d3 30, /control/36/36 (text/plain)
  853 23.6869…  192.168.1…  155.230.…  CoAP     219 NON, MID:37, PUT, TKN:b5 c9 df 2b 0a c5 83 87, /stream/37/36 (text/plain)
  854 24.1871…  192.168.1…  155.230.…  CoAP     219 NON, MID:38, PUT, TKN:be e5 4e 0a 02 a0 12 71, /stream/38/36 (text/plain)
  864 24.6877…  192.168.1…  155.230.…  CoAP     219 NON, MID:39, PUT, TKN:03 4d 06 1c 64 8a b6 40, /stream/39/36 (text/plain)
  865 24.6892…  192.168.1…  192.168.…  ICMP     247 Destination unreachable (Network unreachable)
  866 25.1880…  192.168.1…  155.230.…  CoAP     219 NON, MID:40, PUT, TKN:22 fd d2 4a 4a 4d 9f 05, /stream/40/36 (text/plain)
  867 25.1900…  192.168.1…  192.168.…  ICMP     247 Destination unreachable (Network unreachable)
  940 25.6885…  192.168.1…  155.230.…  CoAP     219 NON, MID:41, PUT, TKN:af 5f 54 cc c4 f5 d7 ac, /stream/41/36 (text/plain)
  941 25.6900…  192.168.1…  192.168.…  ICMP     247 Destination unreachable (Network unreachable)
 1023 26.2012…  192.168.1…  155.230.…  CoAP     219 NON, MID:42, PUT, TKN:ed fe 1a b5 de 94 95 3f, /stream/42/36 (text/plain)
 1024 26.2083…  192.168.1…  192.168.…  ICMP     247 Destination unreachable (Network unreachable)
 1030 26.7059…  192.168.1…  155.230.…  CoAP     219 NON, MID:43, PUT, TKN:47 50 ec f0 fe 9a 9a 25, /stream/43/36 (text/plain)
 1031 26.7074…  192.168.1…  192.168.…  ICMP     247 Destination unreachable (Network unreachable)
 1034 27.2058…  192.168.1…  155.230.…  CoAP     219 NON, MID:44, PUT, TKN:97 ea 51 9b 95 35 eb 2d, /stream/44/36 (text/plain)
 1037 27.7128…  192.168.1…  155.230.…  CoAP     219 NON, MID:45, PUT, TKN:20 8d 29 a1 c4 33 f2 8c, /stream/45/36 (text/plain)
 1050 28.2095…  155.230.3…  192.168.…  CoAP      68 NON, MID:13, GET, TKN:9c 09 57 24 8a bb b9 44, /control/39/38
 1051 28.2095…  155.230.3…  192.168.…  CoAP      68 NON, MID:14, GET, TKN:aa 73 d8 d1 0b 36 1c 0d, /control/40/39
 1052 28.2095…  155.230.3…  192.168.…  CoAP      68 NON, MID:15, GET, TKN:8b 98 a5 c9 b6 08 5a 8a, /control/41/39
 1053 28.2095…  155.230.3…  192.168.…  CoAP      68 NON, MID:16, GET, TKN:5b 2e dd cc b1 a5 28 cb, /control/42/39
 1054 28.2095…  155.230.3…  192.168.…  CoAP      68 NON, MID:17, GET, TKN:5f 67 20 70 38 a6 43 92, /control/43/39
 1055 28.2098…  192.168.1…  155.230.…  CoAP     219 NON, MID:46, PUT, TKN:d9 ce 3c 1c 03 d9 17 00, /stream/39/38 (text/plain)
 1056 28.2099…  192.168.1…  155.230.…  CoAP     219 NON, MID:47, PUT, TKN:a0 18 82 a5 84 56 5d 75, /stream/40/39 (text/plain)
 1057 28.2100…  192.168.1…  155.230.…  CoAP     219 NON, MID:48, PUT, TKN:c9 d5 06 c7 33 25 65 84, /stream/41/39 (text/plain)
 1058 28.2100…  192.168.1…  155.230.…  CoAP     219 NON, MID:49, PUT, TKN:9f 00 ab 14 df 29 8e 95, /stream/42/39 (text/plain)
 1059 28.2101…  192.168.1…  155.230.…  CoAP     219 NON, MID:50, PUT, TKN:07 dc e1 d2 6e 96 eb c5, /stream/43/39 (text/plain)
 1060 28.2133…  192.168.1…  155.230.…  CoAP     219 NON, MID:51, PUT, TKN:82 f0 e7 fb 44 7c 69 30, /stream/46/39 (text/plain)
 1061 28.2309…  155.230.3…  192.168.…  CoAP      75 NON, MID:18, PUT, TKN:b0 39 b4 9d 83 95 d2 88, /control/39/39 (text/plain)
 1083 28.7276…  192.168.1…  155.230.…  CoAP     219 NON, MID:52, PUT, TKN:fb 56 28 60 5c cc 40 16, /stream/47/39 (text/plain)
 1084 28.7328…  155.230.3…  192.168.…  CoAP      75 NON, MID:19, PUT, TKN:1b b9 e7 c9 8c 27 75 42, /control/47/47 (text/plain)
 1177 29.2276…  192.168.1…  155.230.…  CoAP     219 NON, MID:53, PUT, TKN:ad 28 51 90 6f 5c 30 c3, /stream/48/47 (text/plain)
 1198 29.7423…  192.168.1…  155.230.…  CoAP     219 NON, MID:54, PUT, TKN:aa da af 86 bf 6c fd e7, /stream/49/47 (text/plain)
 1216 30.2497…  192.168.1…  155.230.…  CoAP     219 NON, MID:55, PUT, TKN:7b 76 74 31 d3 76 1f a7, /stream/50/47 (text/plain)
 1217 30.2622…  155.230.3…  192.168.…  CoAP      75 NON, MID:20, PUT, TKN:8f 74 a0 ee 9c 5f 74 49, /control/50/50 (text/plain)
```

Figure 12. Packets of error handling for CoAP-SC.

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:

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

$$\text{(b) dataMsg.SN} - \text{dataMsg.AN} \;\geq\; \frac{2}{3} \cdot bufsize$$

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.

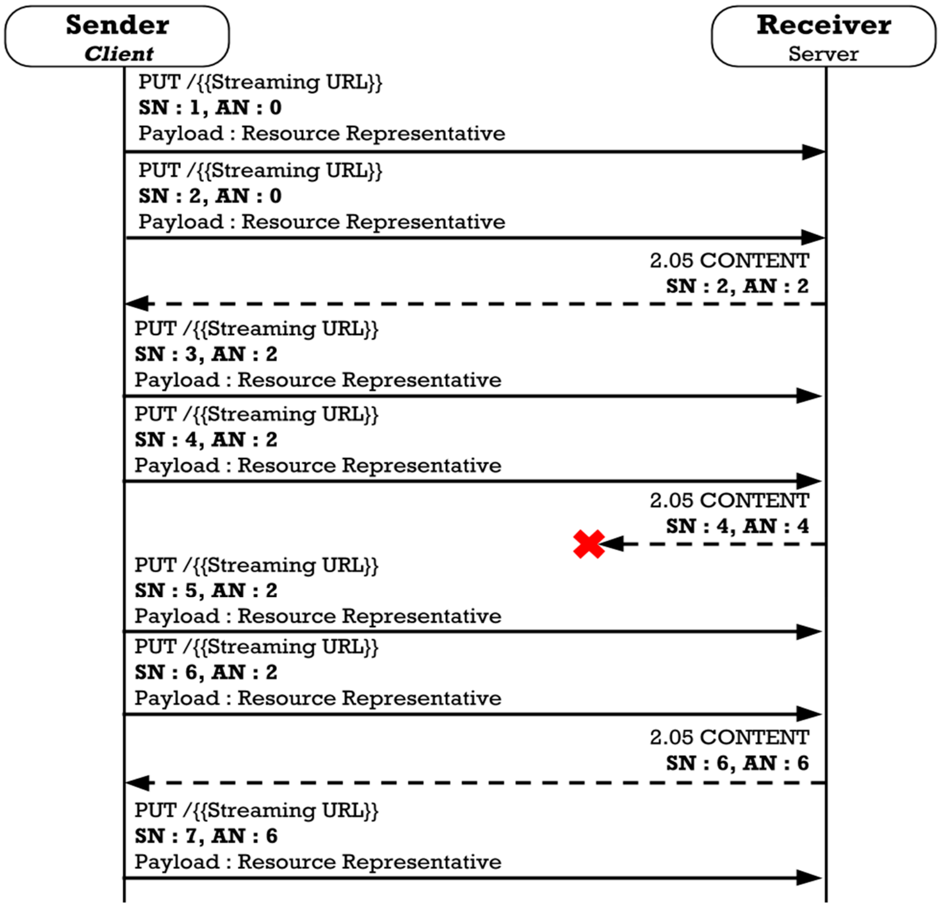
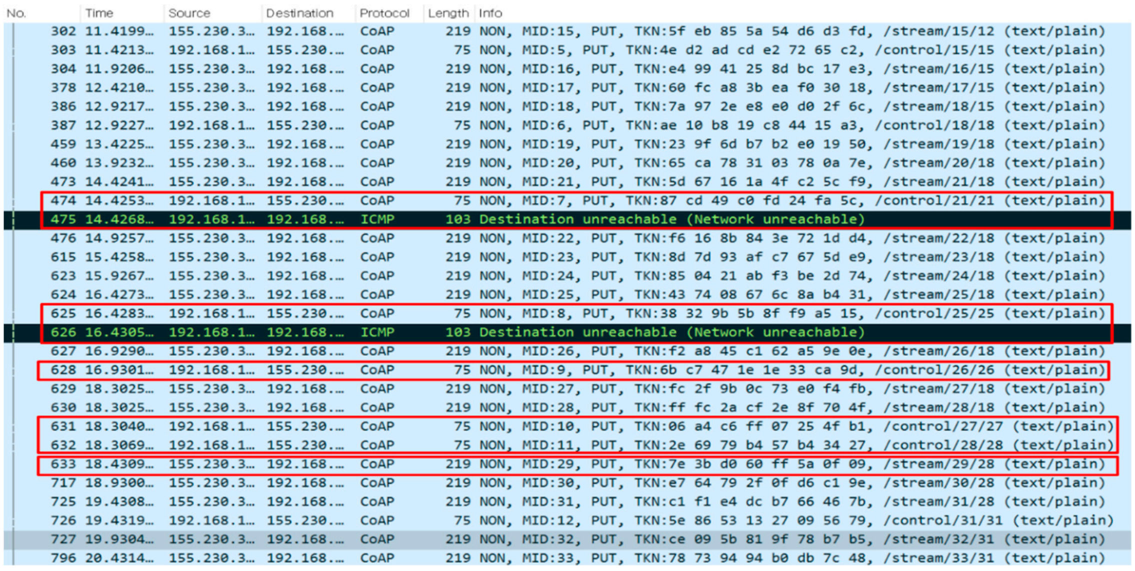


Figure 13. Example of flow control operations for CoAP-SC.

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).



```
No.     Time       Source       Destination  Protocol  Length Info
    302 11.4199…  155.230.3…  192.168.…   CoAP       219 NON, MID:15, PUT, TKN:5f eb 85 5a 54 d6 d3 fd, /stream/15/12 (text/plain)
    303 11.4213…  192.168.1…  155.230.…   CoAP        75 NON, MID:5, PUT, TKN:4e d2 ad cd e2 72 65 c2, /control/15/15 (text/plain)
    304 11.9206…  155.230.3…  192.168.…   CoAP       219 NON, MID:16, PUT, TKN:e4 99 41 25 8d bc 17 e3, /stream/16/15 (text/plain)
    378 12.4210…  155.230.3…  192.168.…   CoAP       219 NON, MID:17, PUT, TKN:60 fc a8 3b ea f0 30 18, /stream/17/15 (text/plain)
    386 12.9217…  155.230.3…  192.168.…   CoAP       219 NON, MID:18, PUT, TKN:7a 97 2e e8 e0 d0 2f 6c, /stream/18/15 (text/plain)
    387 12.9227…  192.168.1…  155.230.…   CoAP        75 NON, MID:6, PUT, TKN:ae 10 b8 19 c8 44 15 a3, /control/18/18 (text/plain)
    459 13.4225…  155.230.3…  192.168.…   CoAP       219 NON, MID:19, PUT, TKN:23 9f 6d b7 b2 e0 19 50, /stream/19/18 (text/plain)
    460 13.9232…  155.230.3…  192.168.…   CoAP       219 NON, MID:20, PUT, TKN:65 ca 78 31 03 78 0a 7e, /stream/20/18 (text/plain)
    473 14.4241…  155.230.3…  192.168.…   CoAP       219 NON, MID:21, PUT, TKN:5d 67 16 1a 4f c2 5c f9, /stream/21/18 (text/plain)
    474 14.4253…  192.168.1…  155.230.…   CoAP        75 NON, MID:7, PUT, TKN:87 cd 49 c0 fd 24 fa 5c, /control/21/21 (text/plain)
    475 14.4268…  192.168.1…  192.168.…   ICMP       103 Destination unreachable (Network unreachable)
    476 14.9257…  155.230.3…  192.168.…   CoAP       219 NON, MID:22, PUT, TKN:f6 16 8b 84 3e 72 1d d4, /stream/22/18 (text/plain)
    615 15.4258…  155.230.3…  192.168.…   CoAP       219 NON, MID:23, PUT, TKN:8d 7d 93 af c7 67 5d e9, /stream/23/18 (text/plain)
    623 15.9267…  155.230.3…  192.168.…   CoAP       219 NON, MID:24, PUT, TKN:85 04 21 ab f3 be 2d 74, /stream/24/18 (text/plain)
    624 16.4273…  155.230.3…  192.168.…   CoAP       219 NON, MID:25, PUT, TKN:43 74 08 67 6c 8a b4 31, /stream/25/18 (text/plain)
    625 16.4283…  192.168.1…  155.230.…   CoAP        75 NON, MID:8, PUT, TKN:38 32 9b 5b 8f f9 a5 15, /control/25/25 (text/plain)
    626 16.4305…  192.168.1…  192.168.…   ICMP       103 Destination unreachable (Network unreachable)
    627 16.9290…  155.230.3…  192.168.…   CoAP       219 NON, MID:26, PUT, TKN:f2 a8 45 c1 62 a5 9e 0e, /stream/26/18 (text/plain)
    628 16.9301…  192.168.1…  155.230.…   CoAP        75 NON, MID:9, PUT, TKN:6b c7 47 1e 1e 33 ca 9d, /control/26/26 (text/plain)
    629 18.3025…  155.230.3…  192.168.…   CoAP       219 NON, MID:27, PUT, TKN:fc 2f 9b 0c 73 e0 f4 fb, /stream/27/18 (text/plain)
    630 18.3025…  155.230.3…  192.168.…   CoAP       219 NON, MID:28, PUT, TKN:ff fc 2a cf 2e 8f 70 4f, /stream/28/18 (text/plain)
    631 18.3040…  192.168.1…  155.230.…   CoAP        75 NON, MID:10, PUT, TKN:06 a4 c6 ff 07 25 4f b1, /control/27/27 (text/plain)
    632 18.3069…  192.168.1…  155.230.…   CoAP        75 NON, MID:11, PUT, TKN:2e 69 79 b4 57 b4 34 27, /control/28/28 (text/plain)
    633 18.4309…  155.230.3…  192.168.…   CoAP       219 NON, MID:29, PUT, TKN:7e 3b d0 60 ff 5a 0f 09, /stream/29/28 (text/plain)
    717 18.9300…  155.230.3…  192.168.…   CoAP       219 NON, MID:30, PUT, TKN:e7 64 79 2f 0f d6 c1 9e, /stream/30/28 (text/plain)
    725 19.4308…  155.230.3…  192.168.…   CoAP       219 NON, MID:31, PUT, TKN:c1 f1 e4 dc b7 66 46 7b, /stream/31/28 (text/plain)
    726 19.4319…  192.168.1…  155.230.…   CoAP        75 NON, MID:12, PUT, TKN:5e 86 53 13 27 09 56 79, /control/31/31 (text/plain)
    727 19.9304…  155.230.3…  192.168.…   CoAP       219 NON, MID:32, PUT, TKN:ce 09 5b 81 9f 78 b7 b5, /stream/32/31 (text/plain)
    796 20.4314…  155.230.3…  192.168.…   CoAP       219 NON, MID:33, PUT, TKN:78 73 94 94 b0 db 7c 48, /stream/33/31 (text/plain)
```

Figure 14. Packets of flow control for CoAP-SC.

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.

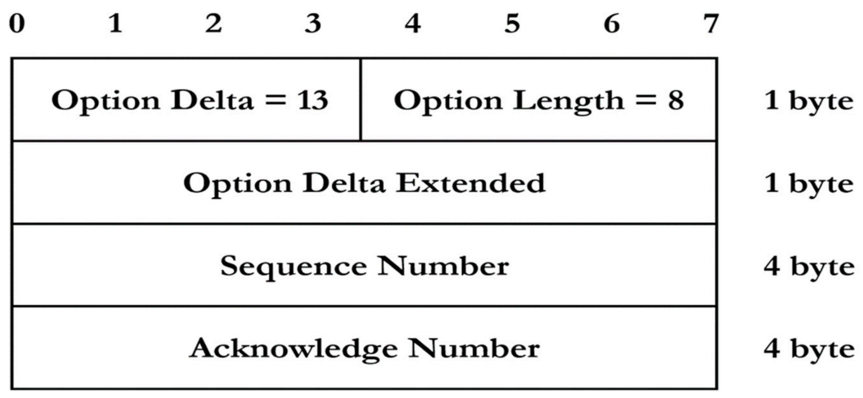


Figure 15. CoAP-SC option header format.

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.

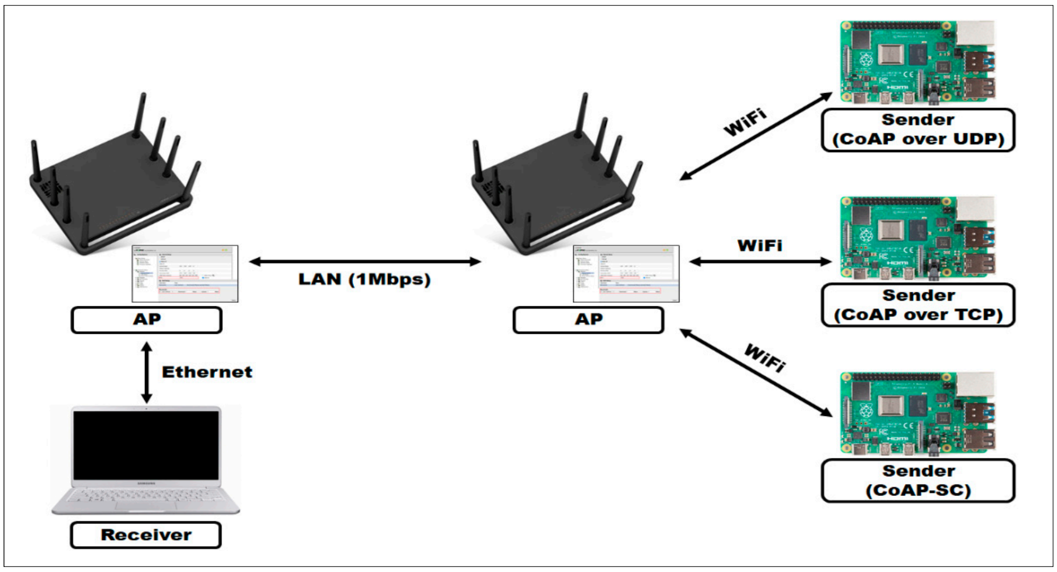
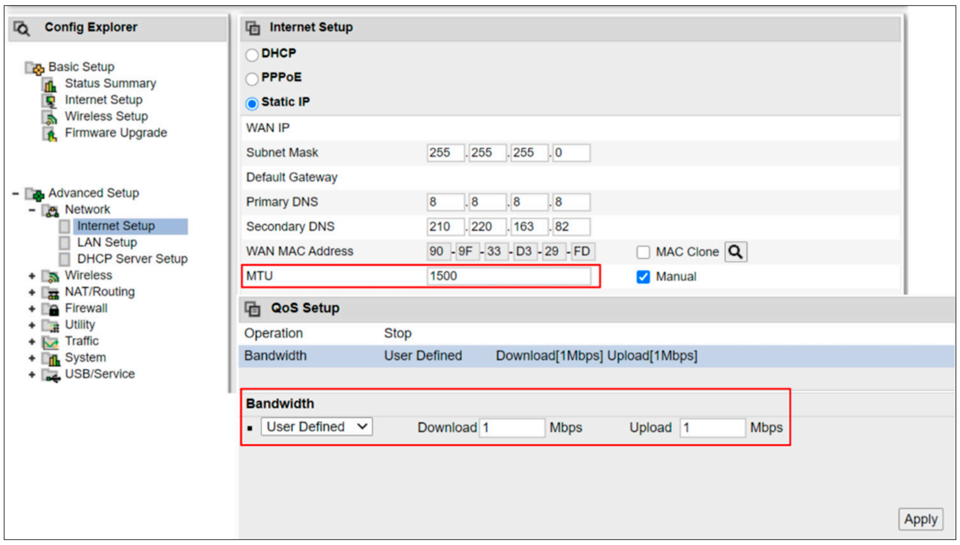


Figure 16. Testbed configuration.



Internet Setup

DHCP
PPPoE
[✓ Static IP]

WAN IP	
Subnet Mask	255 . 255 . 255 . 0
Default Gateway	
Primary DNS	8 . 8 . 8 . 8
Secondary DNS	210 . 220 . 163 . 82
WAN MAC Address	90 - 9F - 33 - D3 - 29 - FD ☐ MAC Clone
MTU	1500 ☑ Manual

QoS Setup

Operation	Stop	
Bandwidth	User Defined	Download[1Mbps] Upload[1Mbps]

Bandwidth
User Defined Download 1 Mbps Upload 1 Mbps

Figure 17. AP configuration and error event generation.

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.

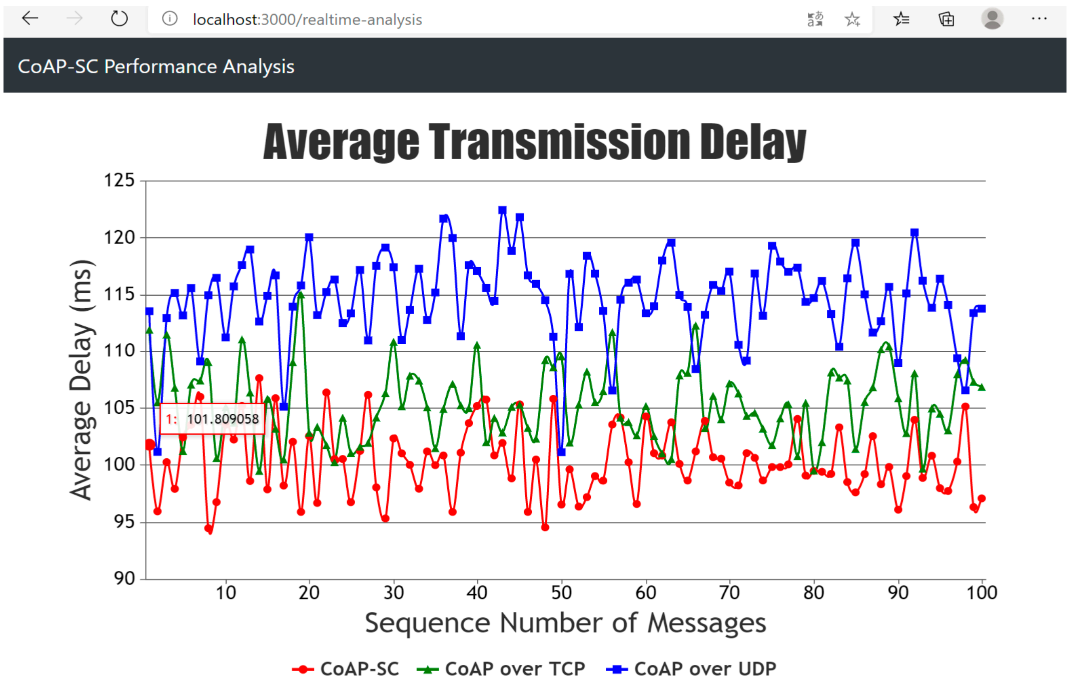


Figure 18. Average transmission delays.

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 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.

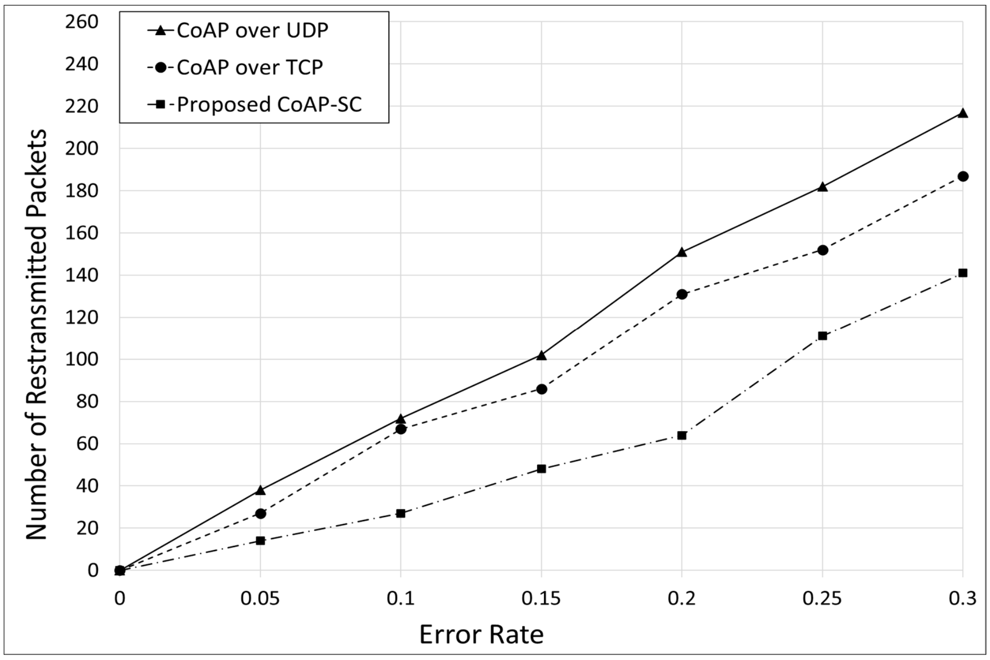


Figure 19. Number of retransmitted packets for different error rates.

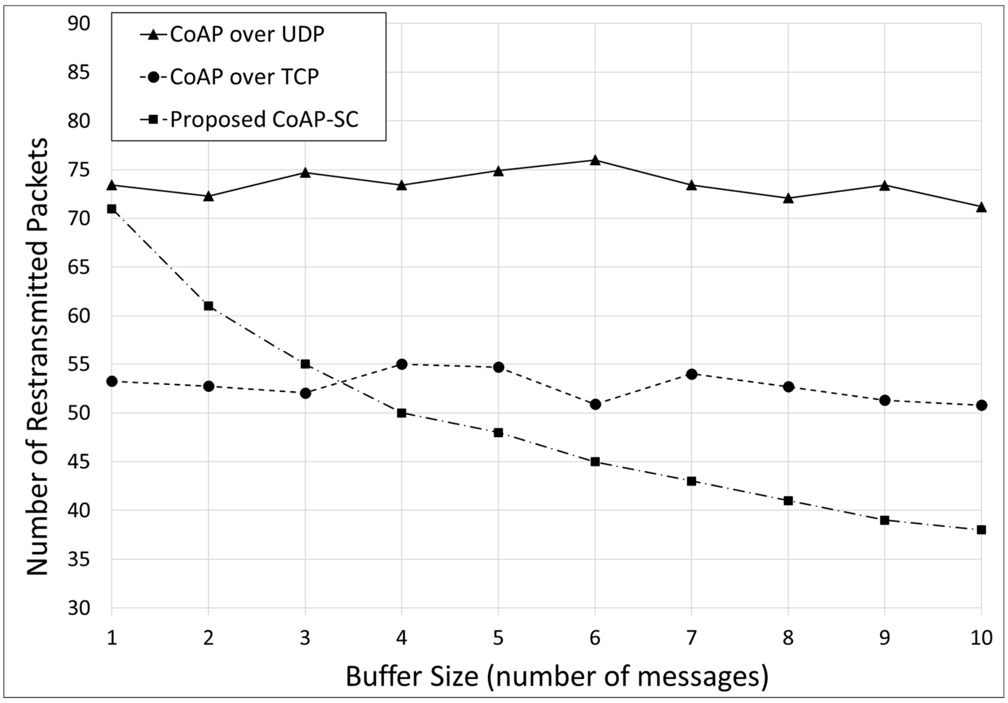


Figure 20. Number of retransmitted packets for different buffer size.

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.

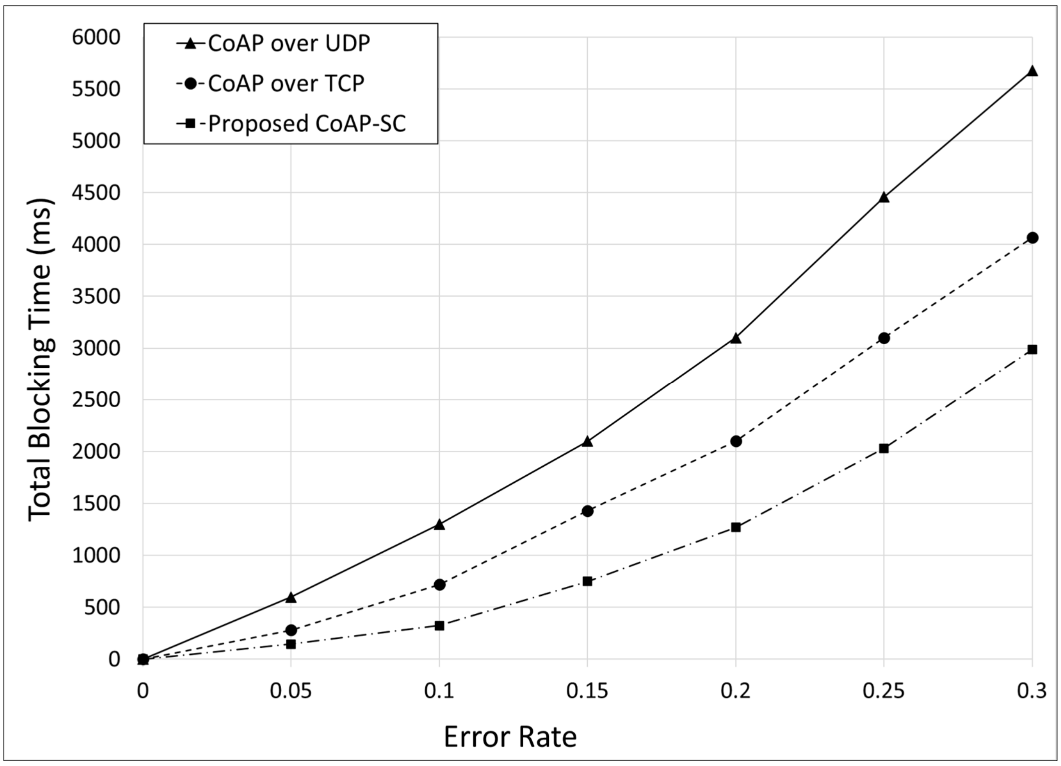


Figure 21. Total blocking time for different error rates.

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.

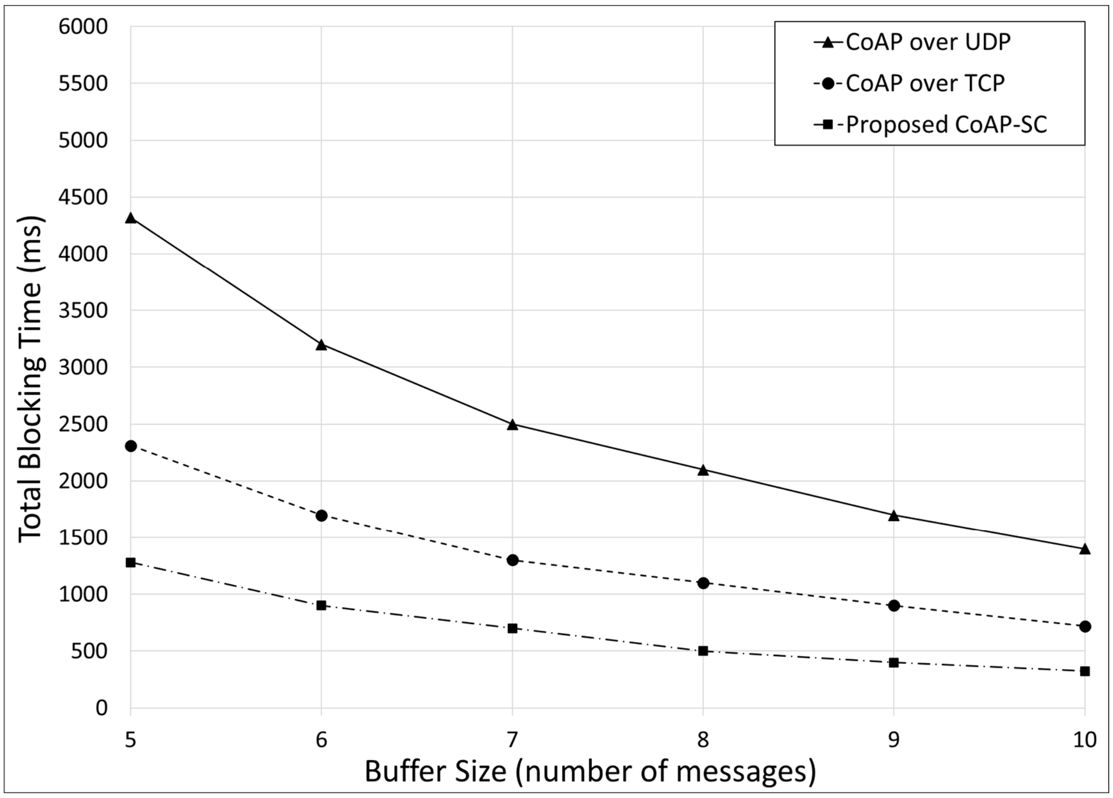


Figure 22. Total blocking times for different buffer sizes.

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.

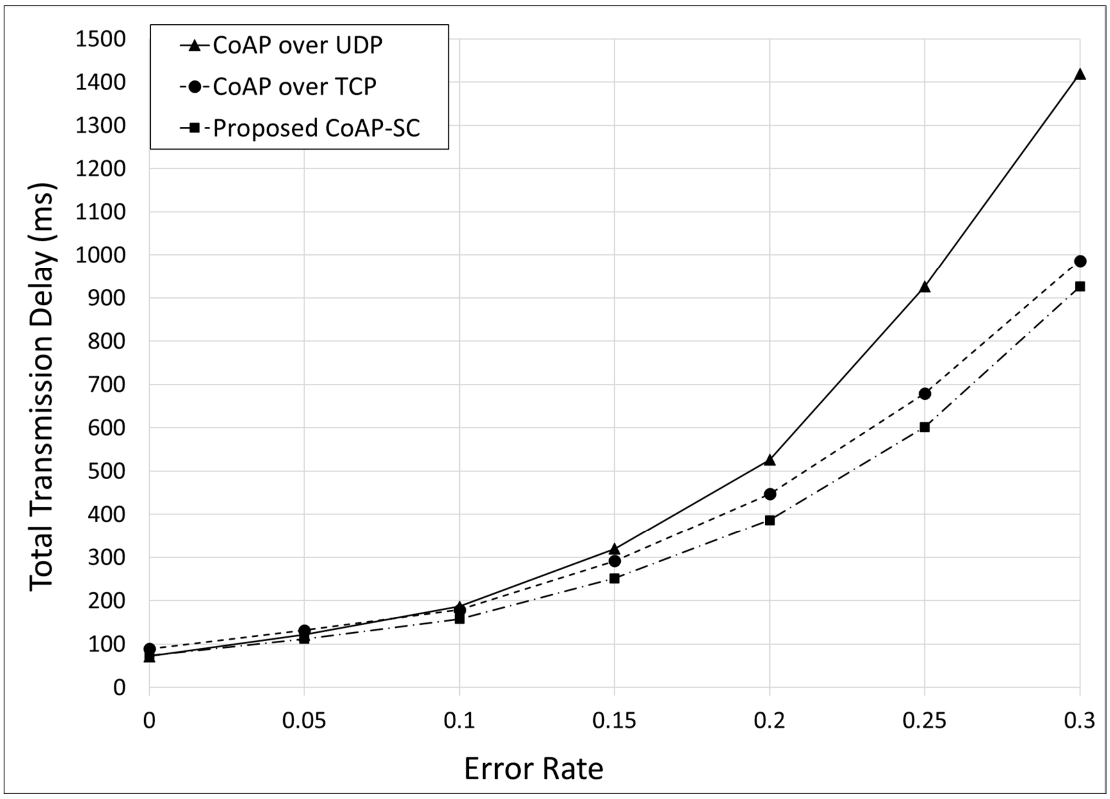


Figure 23. Total transmission delays for different error rates.

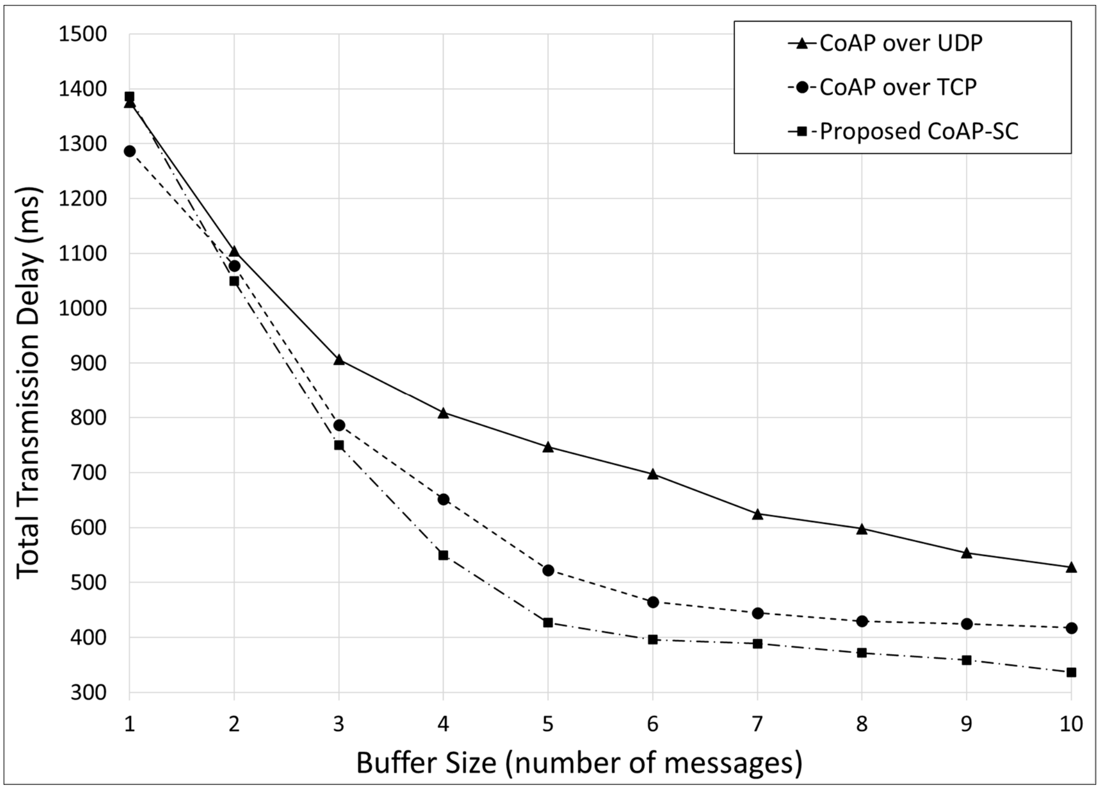


Figure 24. Total transmission delays for different buffer sizes.

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.; Hoebeke, 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, I. 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. *IETF, 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/Godopu/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.