   **Fast-Slow Retransmission Timeout and Congestion Control Algorithm for
                                  CoAP
                      draft-jarvinen-core-fasor-02**

Abstract

   This document specifies an alternative retransmission timeout and
   congestion control back off algorithm for the CoAP protocol, called
   Fast-Slow RTO (FASOR).

   The algorithm specified in this document employs an appropriate and
   large enough back off of Retransmission Timeout (RTO) as the major
   congestion control mechanism to allow acquiring unambiguous RTT
   samples with high probability and to prevent building a persistent
   queue when retransmitting. The algorithm also aims to retransmit
   quickly using an accurately managed retransmission timeout when link-
   errors are occuring, basing RTO calculation on unambiguous round-trip
   time (RTT) samples.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

   CoAP senders use retransmission timeout (RTO) to infer losses that
   have occurred in the network. For such a heuristic to be correct,
   the RTT estimate used for calculating the retransmission timeout must
   match to the real end-to-end path characteristics. Otherwise,
   unnecessary retransmission may occur. Both default RTO mechanism for
   CoAP [RFC7252] and CoCoA [I-D.ietf-core-cocoa] have issues in dealing
   with unnecessary retransmissions and in the worst-case the situation
   can persist causing congestion collapse [JRCK18a].

This document specifies FASOR retransmission timeout and congestion control algorithm [JRCK18b]. FASOR algorithm ensures unnecessary retransmissions that a sender may have sent due to an inaccurate RTT estimate will not persist avoiding the threat of congestion collapse. FASOR also aims to quickly restore the accuracy of the RTT estimate. Armed with an accurate RTT estimate, FASOR not only handles congestion robustly but also can quickly infer losses due to link errors.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

3. Problems with Existing CoAP Congestion Control Algorithms

Correctly inferring losses requires the retransmission timeout (RTO) to be longer than the real RTT in the network. Under certain circumstances the RTO may be incorrectly small. If the real end-to-end RTT is larger than the retransmission timeout, it is impossible for the sender to avoid making unnecessary retransmissions that duplicate data still existing in the network because the sender cannot receive any feedback in time. Unnecessary retransmissions cause two basic problems. First, they increase the perceived end-to-end RTT if the bottleneck has buffering capacity, and second, they prevent getting unambiguous RTT samples. Making unnecessary retransmissions is also a pre-condition for the congestion collapse [RFC0896], which may occur in the worst case if retransmissions are not well controlled [JRCK18a]. Therefore, the sender retransmission timeout algorithm should actively attempt to prevent unnecessary retransmissions from persisting under any circumstance.

Karn's algorithm [KP87] has prevented unnecessary retransmission from turning into congestion collapse for decades due to robust RTT estimation and retransmission timeout backoff handling. The recent CoAP congestion control algorithms, however, diverge from the principles of Karn's algorithm in significant ways and may pose a threat to the stability of the Internet due to those differences.

The default RTO mechanism for CoAP [RFC7252] uses only an initial RTO dithered between 2 and 3 seconds, while CoCoA [I-D.ietf-core-cocoa] measures RTT both from unambiguous and ambiguous RTT samples and applies a modified version of the TCP RTO algorithm [RFC6298]. The algorithm in RFC 7252 lacks solution to persistent congestion. The binary exponential back off used for the retransmission timeout does not properly address unnecessary retransmissions when RTT is larger

than the default RTO (ACK_TIMEOUT). If the CoAP sender performs
exchanges over an end-to-end path with such a high RTT, it
persistently keeps making unnecessary retransmissions for every
exchange wasting some fraction of the used resources (network
capacity, battery power).

CoCoA [I-D.ietf-core-cocoa] attempts to improve scenarios with link-
error related losses and solve persistent congestion by basing its
RTO value on an estimated RTT. However, there are couple of
exceptions when the RTT estimation is not available:

   - At the beginning of a flow where initial RTO of 2 seconds is
   used.

   - When RTT suddenly jumps high enough to trigger the rule in CoCoA
   that prevents taking RTT samples when more than two
   retransmissions are needed. This may also occur when the packet
   drop rate on the path is high enough.

When RTT estimate is too small, unnecessary retransmission will occur
also with CoCoA. CoCoA being unable to take RTT samples at all is a
particularly problematic phenomenon as it is similarly persisting
state as with the algorithm outlined in RFC 7252 and the network
remains in a congestion collapsed state due to persisting unnecessary
retransmissions.

4. FASOR Algorithm

FASOR [JRCK18b] is composed of three key components: RTO computation,
Slow RTO, and novel retransmission timeout back off logic.

4.1. Computing Normal RTO (FastRTO)

The FASOR algorithm measures the RTT for an CoAP message exchange
over an end-to-end path and computes the RTO value using the TCP RTO
algorithm specified in [RFC6298]. We call this normal RTO or
FastRTO. In contrast to the TCP RTO mechanism, FASOR SHOULD NOT use
1 second lower-bound when setting the RTO because RTO is only a
backup mechanisms for loss detection with TCP, whereas with CoAP RTO
is the primary and only loss detection mechanism. A lower-bound of 1
second would impact timeliness of the loss detection in low RTT
environments. The RTO value MAY be upper-bounded by at least 60
seconds. A CoAP sender using the FASOR algorithm SHOULD set initial
RTO to 2 seconds. The computed RTO value as well as the initial RTO
value is subject to dithering; they are dithered between RTO + 1/4 x
SRTT and RTO + SRTT. For dithering initial RTO, SRTT is unset;
therefore, SRTT is replaced with initial RTO / 3 which is derived
from the RTO formula and equals to a hypothetical initial RTT that

would yield the initial RTO using the SRTT and RTTVAR initialization
rule of RFC 6298. That is, for initial RTO of 2 seconds we use SRTT
value of 2/3 seconds.

FastRTO is updated only with unambiguous RTT samples. Therefore, it
closely tracks the actual RTT of the network and can quickly trigger
a retransmission when the network state is not dubious.
Retransmitting without extra delay is very useful when the end-to-end
path is subject to losses that are unrelated to congestion. When the
first unambiguous RTT sample is received, the RTT estimator is
initialized with that sample as specified in [RFC6298] except RTTVAR
that is set to R/2K.

4.2. Slow RTO

We introduce Slow RTO as a safe way to ensure that only a unique copy
of message is sent before at least one RTT has elapsed. To achieve
this the sender must ensure that its retransmission timeout is set to
a value that is larger than the path end-to-end RTT that may be
inflated by the unnecessary retransmission themselves. Therefore,
whenever a message needs to be retransmitted, we measure Slow RTO as
the elapsed time required for getting an acknowledgement. That is,
Slow RTO is measured starting from the original transmission of the
request message until the receipt of the acknowledgement, regardless
of the number of retransmissions. In this way, Slow RTO always
covers the worst-case RTT during which a number of unnecessary
retransmissions were made but the acknowledgement is received for the
original transmission. In contrast to computing normal RTO, Slow RTO
is not smoothed because it is derived from the sending pattern of the
retransmissions (that may turn out unnecessary). In order to drain
the potential unnecessary retransmissions successfully from the
network, it makes sense to wait for the time used for sending them
rather than some smoothed value. However, Slow RTO is multiplied by
a factor to allow some growth in load without making Slow RTO too
aggressive (by default the factor of 1.5 is used). FASOR then
applies Slow RTO as one of the backed off timer values used with the
next request message.

Slow RTO allows rapidly converging towards stable operating point
because 1) it lets the duplicate copies sent earlier to drain from
the network reducing the perceived end-to-end RTT, and 2) allows
enough time to acquire an unambiguous RTT sample for the RTO
computation. Robustly acquiring the RTT sample ensures that the next
RTO is set according to the recent measurement and further
unnecessary retransmissions are avoided. Slow RTO itself is a form
of back off because it includes the accumulated time from the
retransmission timeout back off of the previous exchange. FASOR uses
this for its advantage as the time included into Slow RTO is what is

needed to drain all unnecessary retransmissions possibly made during
the previous exchange. Assuming a stable RTT and that all of the
retransmissions were unnecessary, the time to drain them is the time
elapsed from the original transmission to the sending time of the
last retransmission plus one RTT. When the acknowledgement for the
original transmission arrives, one RTT has already elapsed, leaving
only the sending time difference still unaccounted for which is at
minimum the value for Slow RTO (when an RTT sample arrives
immediately after the last retransmission). Even if RTT would be
increasing, the draining still occurs rapidly due to exponentially
backed off frequency in sending the unnecessary retransmissions.

4.3. Retransmission Timeout Back Off Logic

4.3.1. Overview

FASOR uses normal RTO as the base for binary exponential back off
when no retransmission were needed for the previous CoAP message
exchange. When retransmission were needed for the previous CoAP
message exchange, the algorithm rules, however, are more complicated
than with the traditional RTO back off because Slow RTO is injected
into the back off series to reduce high impact of using Slow RTO.
FASOR logic chooses from three possible back off series alternatives:

   FAST back off: Perform traditional RTO back off with the normal
   RTO as the base. Applied when the previous message was not
   retransmitted.

   FAST_SLOW_FAST back off: First perform a probe using the normal
   RTO for the original transmission of the request message to
   improve cases with losses unrelated to congestion. If the probe
   for the original transmission of the request message is successful
   without retransmissions, continue with FAST back off for the next
   message exchange. If the request message needs to be
   retransmitted, continue by using Slow RTO for the first
   retransmission in order to respond to congestion and drain the
   network from the unnecessary retransmissions that were potentially
   sent for the previous exchange. If still further RTOs are needed,
   continue by backing off the normal RTO further on each timeout.
   FAST_SLOW_FAST back off is applied just once when the previous
   request message using FAST back off required one or more
   retransmissions.

   SLOW_FAST back off: Perform Slow RTO first for the original
   transmisssion to respond to congestion and to acquire an
   unambiguous RTT sample with high probability. Then, if the
   original request needs to be retransmitted, continue with the
   normal RTO-based RTO back off serie by backing off the normal RTO

on each timeout. SLOW_FAST back off is applied when the previous
request message using FAST_SLOW_FAST or SLOW_FAST back off
required one or more retransmissions. Once an acknowledgement for
the original transmission with unambigous RTT sample is received,
continue with FAST back off for the next message exchange.

For the initial message, FAST is used with INITIAL_RTO as the FastRTO
value. From there on, state is updated when an acknowledgement
arrives. Following unambiguous RTT samples, FASOR always uses FAST.
Whenever retransmissions are needed, the back off series selection is
first downgraded to FAST_SLOW_FAST back off and then to SLOW_FAST
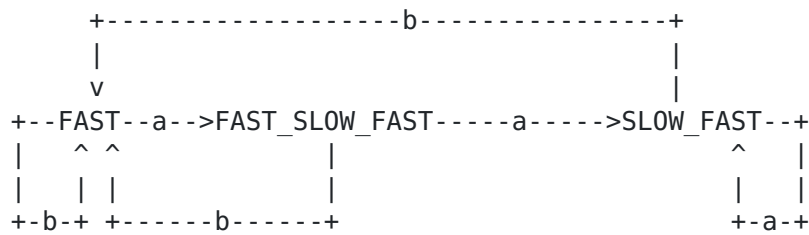back off if further retransmission are needed in FAST_SLOW_FAST.

When Slow RTO is used as the first RTO value, the sender is likely to
acquire unambiguous RTT sample even when the network has high delay
due to congestion because Slow RTO is based on a very recent
measurement of the worst-case RTT. However, using Slow RTO may
negatively impact the performance when losses unrelated to congestion
are occurring. Due to its potential high cost, FASOR algorithm
attempts to avoid using Slow RTO unnecessarily.

The CoAP protocol is often used by devices that are connected through
a wireless network where non-congestion related losses are much more
frequent than in their wired counterparts. This has implications for
the retransmission timeout algorithm. While it would be possible to
implement FASOR such that it immediately uses Slow RTO when a dubious
network state is detected, which would handle congestion very well,
it would do significant harm for performance when RTOs occur due to
non-congestion related losses. Instead, FASOR uses first normal RTO
for one transmission and only responds using Slow RTO if RTO expires
also for that request message. Such a pattern quickly probes if the
losses were unrelated to congestion and only slightly delays response
if real congestion event is taking place. To ensure that an
unambiguous RTT sample is also acquired on a congested network path,
FASOR then needs to use Slow RTO for the original transmission of the
subsequent packet if the probe was not successful.

4.3.2. Retransmission State Machine

FASOR consists of the three states discussed above while making
retranmission decisions, FAST, FAST_SLOW_FAST and SLOW_FAST. The
state machine of the FASOR algorithm is depicted in Figure 1.

```
      +-------------------b----------------+
      |                                    |
      v                                    |
  +--FAST--a-->FAST_SLOW_FAST-----a----->SLOW_FAST--+
  |   ^ ^             |                       ^   |
  |   | |             |                       |   |
  +-b-+ +------b------+                       +-a-+
```

    a: retransmission acknowledged, ambiguous RTT sample acquired;
    b: no retransmission, umambiguous RTT sample acquired;

Figure 1: State Machine of FASOR

In the FAST state, if the original transmission of the message has
not been acknowledged by the receiver within the time defined by
FastRTO, the sender will retransmit it. If there is still no
acknowledgement of the retransmitted packet within 2*FastRTO, the
sender performs the second retransmission and if necessary, each
further retransmission applying binary exponential back off of
FastRTO. The retransmission interval in this state is defined as
FastRTO, 2^1 * FastRTO, ..., 2^i * FastRTO.

When there is an acknowledgement after any retransmission, the sender
will calculate SlowRTO value based on the algorithm defined in
Section 4.2.

When these is an acknowledgement after any retransmission, the sender
will also switch to the second state, FAST_FLOW_FAST. In this state,
the retransmission interval is defined as FastRTO, Max(SlowRTO,
2*FastRTO), FastRTO * 2^1, ..., 2^i * FastRTO. The state will be
switched back to the FAST state once an acknowledgement is returned
within FastRTO, i.e., no retransmission happens for a message. This
is reasonable because it shows the network has recovered from
congestion or bloated queue.

If some retransmission has been made before the acknowledged arrives
in the FAST_SLOW_FAST state, the sender updates the SlowRTO value,
and moves to the third state, SLOW_FAST. The retransmission interval
in the SLOW_FAST state is defined as SlowRTO, FastRTO, FastRTO * 2^1,
..., 2^i * FastRTO.

In SLOW_FAST state, the sender switches back to the FAST state if an
unambiguous acknowledgement arrives. Otherwise, the sender stays in
the SLOW_FAST state if retransmission happens again.

4.4. Retransmission Count Option

When retransmissions are needed to deliver a CoAP message, it is not
possible to measure RTT for the RTO computation as the RTT sample
becomes ambiguous. Therefore, it would be beneficial to be able to
distinguish whether an acknowledgement arrives for the original
transmission of the message or for a retransmission of it. This
would allow reliably acquiring an RTT sample for every CoAP message
exchange and thereby compute a more accurate RTO even during periods
of congestion and loss.

The Retransmission Count Option is used to distinguish whether an
Acknowledgement message arrives for the original transmission or one
of the retransmissions of a Confirmable message. However, the
Retransmission Count Option cannot be used with an Empty
Acknowledgement (or Reset) message because the CoAP protocol
specification [RFC7252] does not allow adding options to an Empty
message. Therefore, Retransmission Count Option is useful only for
the common case of Piggybacked Response. In case of Empty
Acknowledgements the operation of FASOR is the same as without the
option.

```
+-----+---+---+---+---+------------+--------+--------+---------+
| No. | C | U | N | R |    Name    | Format | Length | Default |
+-----+---+---+---+---+------------+--------+--------+---------+
| TBD |   |   | X |   | Rexmit-Cnt |  uint  |  0-1   |    0    |
+-----+---+---+---+---+------------+--------+--------+---------+
```

           C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

              Table 1: Retransmission Count Option

Implementation of the Retransmission Count option is optional and it
is identified as elective. However, when it is present in a CoAP
message and a CoAP endpoint processes it, it MUST be processed as
described in this document. The Retransmission Count option MUST NOT
occur more than once in a single message.

The value of the Retransmission Count option is a variable-size (0 to
1 byte) unsigned integer. The default value for the option is the
number 0 and it is represented with an empty option value (a zero-
length sequence of bytes). However, when a client intents to use
Retransmit Count option, it MUST reserve space for it by limiting the
request message size also when the value is empty in order to fit the
full-sized option into retransmissions.

The Retransmission Count option can be present in both the request
and response message. When the option is present in a request it

indicates the ordinal number of the transmission for the request
message.

If the server supports (implements) the Retransmission Count option
and the option is present in a request, the server MUST echo the
option value in its Piggybacked Response unmodified. If the server
replies with an Empty Acknowledgement the server MUST silently ignore
the option and MUST NOT include it in a later separate response to
that request.

When Piggybacked Response carrying the Retransmission Count option
arrives, the client uses the option to match the response message to
the corresponding transmission of the request. In order to measure a
correct RTT, the client must store the timestamp for the original
transmission of the request as well as the timestamp for each
retransmission, if any, of the request. The resulting RTT sample is
used for the RTO computation. If the client retransmitted the
request without the option but the response includes the option, the
client MUST silently ignore the option.

The original transmission of a request is indicated with the number
0, except when sending the first request to a new destination
endpoint. The first original transmission of the request to a new
endpoint carries the number 255 (0xFF) and is interpreted the same as
an original transmission carrying the number 0. Retransmissions, if
any, carry the ordinal number of the retransmission. Once the first
Piggybacked Response from the new endpoint arrives the client learns
whether or not the other endpoint implements the option. If the
first response includes the echoed option, the client learns that the
other endpoint supports the option and may continue including the
option to each retransmitted request. From this point on the
original transmissions of requests implicitly include the option
number 0 and a zero-byte integer will be sent according to the CoAP
uint-encoding rules. If the first Piggybacked Response does not
include the option, the client SHOULD stop including the option into
the requests to that endpoint.

When the Retransmission Count option is in use, the client bases the
retransmission timeout for the normal RTO in the back off series as
follows:

   max(RTO, Previous-RTT-Sample)

Previous-RTT-Sample is the RTT sample acquired from the previous
message exchange. If no RTT sample was available with the previous
message exchange (e.g., the server replied with an Empty
Acknowledgement), RTO computed earlier is used like in case the
Retransmission Count option is not in use.

4.5. Alternatives for Exchanging Retransmission Count Information

An alternative way of exchanging the retransmission count information
between a client and server is to encode it in the Token. The Token
is a client-local identifier and a client solely decides how it
generates the Token. Therefore, including a varying Token value to
retransmissions of the same request is all possible as long as the
client can use the Token to differentiate between requests and match
a response to the corresponding request. The server is required to
make no assumptions about the content or structure of a Token and
always echo the Token unmodified in its response.

How exactly a client encodes the retransmission count into a Token is
an implementation issue. Note that the original transmission of a
request may carry a zero-length Token given that the rules for
generating a Token as specified in RFC 7252 [RFC7252] are followed.
This allows reducing the overhead of including the Token into the
requests in such cases where Token could otherwise be omitted.
However, similar to Retransmit Count option the maximum request
message size MUST be limited to accommodate the Token with retransmit
count into the retransmissions of the request.

5. Security Considerations

6. IANA Considerations

This memo includes no request to IANA.

7. References

7.1. Normative References

[RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
            Requirement Levels", BCP 14, RFC 2119,
            DOI 10.17487/RFC2119, March 1997,
            <https://www.rfc-editor.org/info/rfc2119>.

[RFC6298]   Paxson, V., Allman, M., Chu, J., and M. Sargent,
            "Computing TCP's Retransmission Timer", RFC 6298,
            DOI 10.17487/RFC6298, June 2011,
            <https://www.rfc-editor.org/info/rfc6298>.

[RFC7252]   Shelby, Z., Hartke, K., and C. Bormann, "The Constrained
            Application Protocol (CoAP)", RFC 7252,
            DOI 10.17487/RFC7252, June 2014,
            <https://www.rfc-editor.org/info/rfc7252>.

[7.2](#). Informative References

[I-D.ietf-core-cocoa]
            Bormann, C., Betzler, A., Gomez, C., and I. Demirkol,
            "CoAP Simple Congestion Control/Advanced", draft-ietf-
            core-cocoa-03 (work in progress), February 2018.

[JRCK18a]   Jarvinen, I., Raitahila, I., Cao, Z., and M. Kojo, "Is
            CoAP Congestion Safe?", Applied Networking Research
            Workshop (ANRW'18), July 2018.

[JRCK18b]   Jarvinen, I., Raitahila, I., Cao, Z., and M. Kojo, "FASOR
            Retransmission Timeout and Congestion Control Mechanism
            for CoAP?", Proceedings of IEEE Global Communications
            Conference (Globecom 2018), to appear, December 2018.

[KP87]   Karn, P. and C. Partridge, "Improving Round-trip Time
            Estimates in Reliable Transport Protocols", SIGCOMM'87
            Proceedings of the ACM Workshop on Frontiers in Computer
            Communications Technology, August 1987.

[RFC0896]   Nagle, J., "Congestion Control in IP/TCP Internetworks",
            RFC 896, DOI 10.17487/RFC0896, January 1984,
            <https://www.rfc-editor.org/info/rfc896>.

Appendix A. Pseudocode for Basic FASOR without Dithering

```
var state = NORMAL_RTO

rfc6298_init(var fastrto, 2 secs)

var slowrto
SLOWRTO_FACTOR = 1.5

var original_sendtime
var retransmit_count

/*
 * Sending Original Copy and Retransmitting 'req'
 */
send_request(req) {
  original_sendtime = time.now
  retransmit_count = 0

  arm_rto(calculate_rto())
  send(req)
}

rto_for(req) {
  retransmit_count += 1

  arm_rto(calculate_rto())
  send(req)
}

/*
 * ACK Processings
 */
ack() {
  sample = time.now - original_sendtime
  if (retransmit_count == 0)
    unambiguous_ack(sample)
  else
    ambiguous_ack(sample)
}

unambiguous_ack(sample) {
  k = 4                                // RFC6298 default K = 4
  if (rfc6298_is_first_sample(fastrto))
    k = 1
  rfc6298_update(fastrto, k, sample)  // Normal RFC6298 processing
  state = NORMAL_RTO
}
```

```
ambiguous_nextstate = {
  [NORMAL_RTO] = FAST_SLOW_FAST_RTO,
  [FAST_SLOW_FAST_RTO] = SLOW_FAST_RTO,
  [SLOW_FAST_RTO] = SLOW_FAST_RTO
}

ambiguous_ack(sample) {
  slowrto = sample * SLOWRTO_FACTOR
  state = ambiguous_nextstate[state]
}

/*
 * RTO Calculations
 */
calculate_rto() {
  return <state>_rtoseries()
}

normal_rtoseries() {
  switch (retransmit_count) {
    case 0: return fastrto_series_init()
    default: return fastrto_series_backoff()
  }
}

fastslowfast_rtoseries() {
  switch (retransmit_count) {
    case 0: return fastrto_series_init()
    case 1: return MAX(slowrto, 2*fastrto)
    default: return fastrto_series_backoff()
  }
}

slowfast_rtoseries() {
  switch (retransmit_count) {
    case 0: return slowrto
    case 1: return fastrto_series_init()
    default: return fastrto_series_backoff()
  }
}

var backoff_series_timer

fastrto_series_init() {
  backoff_series_timer = fastrto
  return backoff_series_timer
}
```

```
fastrto_series_backoff() {
  backoff_series_timer *= 2
  return backoff_series_timer
}
```

Figure 2

Authors' Addresses

    Ilpo Jarvinen
    University of Helsinki
    P.O. Box 68
    FI-00014 UNIVERSITY OF HELSINKI
    Finland

    EMail: ilpo.jarvinen@cs.helsinki.fi

    Markku Kojo
    University of Helsinki
    P.O. Box 68
    FI-00014 UNIVERSITY OF HELSINKI
    Finland

    EMail: markku.kojo@cs.helsinki.fi

    Iivo Raitahila
    University of Helsinki
    P.O. Box 68
    FI-00014 UNIVERSITY OF HELSINKI
    Finland

    EMail: iivo.raitahila@helsinki.fi

    Zhen Cao
    Huawei
    Beijing
    China

    EMail: zhencao.ietf@gmail.com