# Diagramas de interacción del protocolo RTPS

# Diagramas de interaccion con Estado

## Diagramas basados con la QoS Best Effort

### Best Effort Reader – Best Effort Writer con Key

1. El usuario DDS escribe datos por medio de la llamada a la operación *write* en el *DataWriter* DDS.
2. El *DataWriter* DDS llama a la operación *new\_change* en el *Writer* RTPS para crear un nuevo *CacheChange*. Cada uno de estos cambios es identificado únicamente por un *SequenceNumber*.
3. La operación *new\_change* retorna.
4. El *DataWriter* DDS utiliza la operación *add\_change* para almacenar el *CacheChange* dentro de *HistoryCache* del *Writer* RTPS.
5. El *HistoryCache* del *Writer* RTPS notifica el cambio por medio de la operación *notify\_change*  al *Publisher* DDS.
6. La operación *notify\_change*  retorna.
7. La operación *add\_change­* retorna.
8. La operación *write* retorna. El usuario ha completado la acción de escritura de datos.
9. El *HistoryCache*  del *Writer* DDS utiliza la operación *unsent\_changes* para informar al *ReaderProxy* que hay cambios o información no enviada.
10. La operación *unsent\_changes* retorna.
11. El *HistoryCache*  del *Writer* DDS utiliza la operación *can\_send* para informar al *ReaderProxy* que puede enviar los cambios.
12. La operación *can\_send* retorna.
13. El *DataWriter* DDS utiliza la operación *remove\_change*  en el *HistoryCache*  del *Writer* DDS para limpiar la cache. Esta operación puede ser realizada posteriormente.
14. La operación *remove\_change* retorna.
15. El *ReaderProxy* serializa la información mediante la operación *serialize* en el *Serializer*.
16. La operación *serialize*  retorna.
17. El *ReaderProxy*  envía el submensaje DATA al *MessageEncoder* para que sea encapsulado.
18. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
19. La operación *encoded­\_message retorna.*
20. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
21. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
22. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
23. La operación *doDecode* retorna.
24. El *MessageDecoder* envía el submensaje DATA al *WriterProxy*.
25. El *WriterProxy* llama a la operación *deserialize\_data* al *Deserializer*
26. La operación *deserialize\_data* retorna.
27. El *WriterProxy* llama a la operación *available\_change* dentro del *HistoryCache* del *Reader* RTPS, para la verificación de números de secuencia recibidos.
28. La operación *available\_change* retorna.
29. El *WriterProxy* genera un nuevo *CacheChange* con la operación *new\_change*  en el *Reader* RTPS.
30. La operación *new\_change* retorna.
31. El *Stateful Reader* añade el cambio al *HistoryCache* del *Reader* RTPS por medio de la operación *add\_change.*
32. El *HistoryCache* del *Reader* RTPS notifica el cambio al Suscriptor DDS por medio de la operación *notify\_change.*
33. La operación  *notify\_change* retorna.
34. La operación *add\_change* retorna.
35. El Usuario llama a la operación *take* para obtener los datos desde el *DataReader* DDS.
36. El *DataReader* DDS solicita los cambios por medio de la operación *get\_change*.
37. La operación *get\_change* retorna.
38. La operación *take* retorna. Los datos recibidos son entregados al usuario.
39. Una vez obtenido los cambios el *DataReader* DDS elimina los cambios mediante la operación *remove\_change*.
40. La operación *remove\_change* retorna.

### Best Effort Reader – BestEffort Writer sin Key

### Best Effort Reader – BestEffort Writer con Key (Packet Failure)

1. El usuario DDS escribe datos mediante la operación *write* en el DataWriter DDS
2. El DataWriter DDS crea un CacheChange mediante la operación new\_change al Stateful Writer.
3. La operación new\_change retorna.
4. El DataWriter DDS añade el cambio mediante la operación add\_change al HistoryCache del Writer RTPS.
5. El HistoryCache del Writer RTPS notifica al Publisher mediante la operación notify\_change.
6. La operación notify\_change retorna.
7. La operación add\_change retorna.
8. La operación *write* retorna.
9. El HistoryCache del Writer RTPS envía los cambios no enviados mediante la operación unsent\_changes al ReaderProxy.
10. La operación unsent\_changes retorna.
11. El HistoryCache del Writer RTPS le informa que todos los cambios han sido enviados por medio de la operación can\_send.
12. La operación *can\_send* retorna.
13. El ReaderProxy serializa los datos mediante la operación *serialize*.
14. La operación serialize retorna.
15. El *DataWriter* DDS elimina el cambio enviado mediante la operación *remove\_change.*
16. La operación *remove\_change* retorna.
17. El ReaderProxy envía el submensaje DATA al MessageEncoder, para que el mensaje sea encapsulado.
18. El MessageEncoder envía el mensaje encapsulado mediante la operación encoded\_message
19. La operación encoded\_message retorna.
20. El UDPTransmitter envía el mensaje UDP a la red.
21. El usuario intenta obtener datos mediante la operación take al DataReader DDS.
22. El DataReader DDS intenta obtener los cambios mediante la operación get\_change al HistoryCache del Reader RTPS.
23. La operación get\_change retorna.
24. La operación take retorna.

## Diagramas basados con la QoS Reliable

### Reliable Reader—Reliable Writer con Key

1. El usuario DDS escribe datos mediante la operación *write* en el DataWriter DDS
2. El DataWriter DDS crea un CacheChange mediante la operación new\_change al Stateful Writer.
3. La operación new\_change retorna.
4. El DataWriter DDS añade el cambio mediante la operación add\_change al HistoryCache del Writer RTPS.
5. El HistoryCache del Writer RTPS notifica al Publisher mediante la operación notify\_change.
6. La operación notify\_change retorna.
7. La operación add\_change retorna.
8. La operación *write* retorna.
9. El HistoryCache del Writer RTPS envía los cambios no enviados mediante la operación unsent\_changes al ReaderProxy.
10. La operación unsent\_changes retorna.
11. El HistoryCache del Writer RTPS le informa que todos los cambios han sido enviados por medio de la operación can\_send.
12. La operación *can\_send* retorna.
13. El ReaderProxy serializa los datos mediante la operación *serialize*.
14. La operación serialize retorna.
15. El *ReaderProxy* envía al *MessageEncoder* los submensajes GAP, DATA y dependiendo de la política de QoS se envía también un INFO\_TIMESTAMP.
16. El *HistoryCache* del *Writer* RTPS reenvía los números de secuencia de los cambios que no han sido confirmados mediante la operación *unacked\_changes*, para que sean añadidos al submensaje HEARTBEAT.
17. La operación *unacked\_changes* retorna.
18. El *ReaderProxy* envía al *MessageEncoder* el submensaje HEARTBEAT.
19. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
20. La operación *encoded­\_message retorna.*
21. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
22. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
23. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
24. La operación *doDecode* retorna.
25. El *WriterProxy* recibe el submensaje HEARTBEAT desde el *MessageDecoder.*
26. El *WriterProxy* deserializa el submensaje por medio de la operación *deserialize\_data.*
27. La operación  *deserialize\_data* retorna.
28. El *WriterProxy* obtiene una lista de los cambios que se han perdido por medio de la operación *missing\_changes* al *HistoryCache* del *Reader* RTPS.
29. La operación *missing\_changes* retorna.
30. Los números de secuencia faltantes son serializados en el *WriterProxy* mediante la operación *serialize.*
31. La operación *serialize* retorna.
32. El *WriterProxy* envía los submensajes INFO\_DESTINATION y ACKNACK con la confirmación de recepción o pérdida de paquetes.
33. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
34. La operación *encoded­\_message retorna.*
35. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
36. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
37. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
38. La operación *doDecode* retorna.
39. El *ReaderProxy* recibe los submensajes INFO\_DESTINATION y ACKNACK desde el *MessageEncoder.* El submensaje INFO\_DESTINATION contiene el destino el cual ha confirmado el cambio.
40. El *ReaderProxy* deserializa el submensaje por medio de la operación *deserialize\_data.*
41. La operación  *deserialize\_data* retorna.
42. El *ReaderProxy* envía al *Stateful Writer* los cambios que han sido confirmados mediante la operación *acked\_changes*.
43. La operación *acked\_changes* retorna.
44. El *DataWriter* DDS consulta mediante la operación *is\_acked\_by\_all* al *Stateful Writer* que todos los suscriptores tienen los cambios.
45. La operación *is\_acked\_by\_all* retorna.
46. El *DataWriter* DDS elimina los cambios cuando todos suscriptores han recibido los cambios por medio de la operación *remove\_change* al *HistoryCache* del *Writer* RTPS.
47. La operación  *remove\_change* retorna.
48. Este literal toma lugar después del punto 25, luego de recibir el HEARTBEAT en el lado del suscriptor. El *WriterProxy* recibe los submensajes GAP, INFO\_TIMESTAMP y DATA del *MessageDecoder.*
49. El *WriterProxy* genera un nuevo *CacheChange* con la operación *new\_change*  en el *Reader* RTPS.
50. La operación *new\_change* retorna.
51. El *Stateful Reader* añade el cambio al *HistoryCache* del *Reader* RTPS por medio de la operación *add\_change.*
52. El *HistoryCache* del *Reader* RTPS notifica el cambio al Suscriptor DDS por medio de la operación *notify\_change.*
53. La operación  *notify\_change* retorna.
54. La operación *add\_change* retorna.
55. El Usuario llama a la operación *take* para obtener los datos desde el *DataReader* DDS.
56. El *DataReader* DDS solicita los cambios por medio de la operación *get\_change*.
57. La operación *get\_change* retorna.
58. La operación *take* retorna. Los datos recibidos son entregados al usuario.
59. El usuario indica al *DataReader* DDS que ya obtuvo el cambio mediante la operación *return\_loan.*
60. El *DataReader* DDS pregunta al *HistoryCache* del *Reader* RTPS si el cambio indicado es relevante mediante la operación *a\_change\_is\_relevant.*
61. La operación *a\_change\_is\_relevant* retorna.
62. Dependiendo si el cambio es relevante el *DataReader* DDS elimina los cambios mediante la operación *remove\_change*.
63. La operación *remove\_change* retorna.
64. La operación *return\_loan* retorna.

### Reliable Reader—Reliable Writer con Key con fragmentación

1. El usuario DDS escribe datos mediante la operación *write* en el DataWriter DDS
2. El DataWriter DDS crea un CacheChange mediante la operación new\_change al Stateful Writer.
3. La operación new\_change retorna.
4. El DataWriter DDS añade el cambio mediante la operación add\_change al HistoryCache del Writer RTPS.
5. El HistoryCache del Writer RTPS notifica al Publisher mediante la operación notify\_change.
6. La operación notify\_change retorna.
7. La operación add\_change retorna.
8. La operación *write* retorna.
9. El HistoryCache del Writer RTPS envía los cambios no enviados mediante la operación unsent\_changes al ReaderProxy.
10. La operación unsent\_changes retorna.
11. El HistoryCache del Writer RTPS le informa que todos los cambios han sido enviados por medio de la operación can\_send.
12. La operación *can\_send* retorna.
13. El ReaderProxy serializa los datos mediante la operación *serialize*, y se realiza el proceso de fragmentación de la información.
14. La operación serialize retorna.
15. El *ReaderProxy* envía al *MessageEncoder* los submensajes GAP, DATA\_FRAG y dependiendo de la política de QoS se envía también un INFO\_TIMESTAMP.
16. El *HistoryCache* del *Writer* RTPS reenvía los números de secuencia de los cambios que no han sido confirmados mediante la operación *unacked\_changes*, para que sean añadidos a los submensajes HEARTBEAT\_FRAG.
17. La operación *unacked\_changes* retorna.
18. El *ReaderProxy* envía al *MessageEncoder* los submensajes HEARTBEAT\_FRAG.
19. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
20. La operación *encoded­\_message retorna.*
21. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
22. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
23. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
24. La operación *doDecode* retorna.
25. El *WriterProxy* recibe los submensajes HEARTBEAT\_FRAG desde el *MessageDecoder.*
26. El *WriterProxy* deserializa el submensaje por medio de la operación *deserialize\_data.*
27. La operación  *deserialize\_data* retorna.
28. El *WriterProxy* obtiene una lista de los cambios que se han perdido por medio de la operación *missing\_changes* al *HistoryCache* del *Reader* RTPS.
29. La operación *missing\_changes* retorna.
30. Los números de secuencia faltantes son serializados en el *WriterProxy* mediante la operación *serialize.*
31. La operación *serialize* retorna.
32. El *WriterProxy* envía los submensajes INFO\_DESTINATION y NACK\_FRAG con la confirmación de recepción o pérdida de paquetes.
33. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
34. La operación *encoded­\_message retorna.*
35. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
36. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
37. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
38. La operación *doDecode* retorna.
39. El *ReaderProxy* recibe los submensajes INFO\_DESTINATION y NACK\_FRAG desde el *MessageEncoder.* El submensaje INFO\_DESTINATION contiene el destino el cual ha confirmado el cambio.
40. El *ReaderProxy* deserializa el submensaje por medio de la operación *deserialize\_data.*
41. La operación  *deserialize\_data* retorna.
42. El *ReaderProxy* envía al *Stateful Writer* los cambios que han sido confirmados mediante la operación *acked\_changes*.
43. La operación *acked\_changes* retorna.
44. El *DataWriter* DDS consulta mediante la operación *is\_acked\_by\_all* al *Stateful Writer* que todos los suscriptores tienen los cambios.
45. La operación *is\_acked\_by\_all* retorna.
46. El *DataWriter* DDS elimina los cambios cuando todos suscriptores han recibido los cambios por medio de la operación *remove\_change* al *HistoryCache* del *Writer* RTPS.
47. La operación  *remove\_change* retorna.
48. Este literal toma lugar después del punto 25, luego de recibir el HEARTBEAT en el lado del suscriptor. El *WriterProxy* recibe los submensajes GAP, INFO\_TIMESTAMP y DATA\_FRAG del *MessageDecoder.*
49. El *WriterProxy* genera un nuevo *CacheChange* con la operación *new\_change*  en el *Reader* RTPS.
50. La operación *new\_change* retorna.
51. El *Stateful Reader* añade el cambio al *HistoryCache* del *Reader* RTPS por medio de la operación *add\_change.*
52. El *HistoryCache* del *Reader* RTPS notifica el cambio al Suscriptor DDS por medio de la operación *notify\_change.*
53. La operación  *notify\_change* retorna.
54. La operación *add\_change* retorna.
55. El Usuario llama a la operación *take* para obtener los datos desde el *DataReader* DDS.
56. El *DataReader* DDS solicita los cambios por medio de la operación *get\_change*.
57. La operación *get\_change* retorna.
58. La operación *take* retorna. Los datos recibidos son entregados al usuario.
59. El usuario indica al *DataReader* DDS que ya obtuvo el cambio mediante la operación *return\_loan.*
60. El *DataReader* DDS pregunta al *HistoryCache* del *Reader* RTPS si el cambio indicado es relevante mediante la operación *a\_change\_is\_relevant.*
61. La operación *a\_change\_is\_relevant* retorna.
62. Dependiendo si el cambio es relevante el *DataReader* DDS elimina los cambios mediante la operación *remove\_change*.
63. La operación *remove\_change* retorna.
64. La operación *return\_loan* retorna.

### Reliable Reader—Reliable Writer con Key (Communication Error)

1. El usuario DDS escribe datos mediante la operación *write* en el DataWriter DDS
2. El DataWriter DDS crea un CacheChange mediante la operación new\_change al Stateful Writer.
3. La operación new\_change retorna.
4. El DataWriter DDS añade el cambio mediante la operación add\_change al HistoryCache del Writer RTPS.
5. El HistoryCache del Writer RTPS notifica al Publisher mediante la operación notify\_change.
6. El Publisher DDS indica la disponibilidad de datos al *ReaderProxy* mediante la operación *data\_available.*
7. El *ReaderProxy* serializa la notificación mediante la operación *serialize.*
8. La operación *serialize* retorna.
9. El *ReaderProxy* envía un submensajeHEARTBEAT en modo *waiting* al *MessageEncoder.*
10. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
11. La operación *encoded­\_message retorna.*
12. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
13. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
14. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
15. La operación *doDecode* retorna.
16. El *WriterProxy* recibe el HEARTBEAT en modo *waiting* del *MessageDecoder.* El envío de este tipo HEARTBEAT sirve solamente para confirmar la presencia de un participante.
17. El *WriterProxy* deserializa el submensaje mediante la operación *deserialize\_data.*
18. La operación  *deserialize\_data* retorna.
19. La operación *notify\_change*  retorna.
20. La operación *add\_change­* retorna.
21. La operación *write* retorna. El usuario ha completado la acción de escritura de datos.
22. El *HistoryCache*  del *Writer* DDS utiliza la operación *unsent\_changes* para informar al *ReaderProxy* que hay cambios o información no enviada.
23. La operación *unsent\_changes* retorna.
24. El *HistoryCache*  del *Writer* DDS utiliza la operación *can\_send* para informar al *ReaderProxy* que puede enviar los cambios.
25. La operación *can\_send* retorna.
26. El ReaderProxy serializa los datos mediante la operación *serialize*.
27. La operación serialize retorna.
28. El *ReaderProxy* envía al *MessageEncoder* los submensajes GAP, DATA y dependiendo de la política de QoS se envía también un INFO\_TIMESTAMP.
29. El *HistoryCache* del *Writer* RTPS reenvía los números de secuencia de los cambios que no han sido confirmados mediante la operación *unacked\_changes*, para que sean añadidos al submensaje HEARTBEAT.
30. La operación *unacked\_changes* retorna.
31. El *ReaderProxy* envía al *MessageEncoder* el submensaje HEARTBEAT. Este HEARTBEAT tiene un temporizador llamado *Heartbeat period* en el cual se debería recibir las confirmaciones.
32. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
33. La operación *encoded­\_message retorna.*
34. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos. En este punto se ha simulado una falla en la comunicación por tanto el mensaje no llega a su destino.
35. El *ReaderProxy* reenvía al *MessageEncoder* los submensajes GAP, DATA y dependiendo de la política de QoS se envía también un INFO\_TIMESTAMP.
36. El *ReaderProxy* reenvía al *MessageEncoder* el submensaje HEARTBEAT.
37. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
38. La operación *encoded­\_message retorna.*
39. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
40. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
41. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
42. La operación *doDecode* retorna.
43. El *WriterProxy* recibe el submensaje HEARTBEAT desde el *MessageDecoder.*
44. El *WriterProxy* deserializa el submensaje por medio de la operación *deserialize\_data.*
45. La operación  *deserialize\_data* retorna.
46. El *WriterProxy* obtiene una lista de los cambios que se han perdido por medio de la operación *missing\_changes* al *HistoryCache* del *Reader* RTPS.
47. La operación *missing\_changes* retorna.
48. Los números de secuencia faltantes son serializados en el *WriterProxy* mediante la operación *serialize.*
49. La operación *serialize* retorna.
50. El *WriterProxy* envía los submensajes INFO\_DESTINATION y ACKNACK con la confirmación de recepción o pérdida de paquetes.
51. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
52. La operación *encoded­\_message retorna.*
53. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
54. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
55. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
56. La operación *doDecode* retorna.
57. El *ReaderProxy* recibe los submensajes INFO\_DESTINATION y ACKNACK desde el *MessageEncoder.* El submensaje INFO\_DESTINATION contiene el destino el cual ha confirmado el cambio.
58. El *ReaderProxy* deserializa el submensaje por medio de la operación *deserialize\_data.*
59. La operación  *deserialize\_data* retorna.
60. El *ReaderProxy* envía al *Stateful Writer* los cambios que han sido confirmados mediante la operación *acked\_changes*.
61. La operación *acked\_changes* retorna.
62. El *DataWriter* DDS consulta mediante la operación *is\_acked\_by\_all* al *Stateful Writer* que todos los suscriptores tienen los cambios.
63. La operación *is\_acked\_by\_all* retorna.
64. El *DataWriter* DDS elimina los cambios cuando todos suscriptores han recibido los cambios por medio de la operación *remove\_change* al *HistoryCache* del *Writer* RTPS.
65. La operación  *remove\_change* retorna.
66. Este literal toma lugar después del punto 43, luego de recibir el HEARTBEAT en el lado del suscriptor. El *WriterProxy* recibe los submensajes GAP, INFO\_TIMESTAMP y DATA del *MessageDecoder.*
67. El *WriterProxy* genera un nuevo *CacheChange* con la operación *new\_change*  en el *Reader* RTPS.
68. La operación *new\_change* retorna.
69. El *Stateful Reader* añade el cambio al *HistoryCache* del *Reader* RTPS por medio de la operación *add\_change.*
70. El *HistoryCache* del *Reader* RTPS notifica el cambio al Suscriptor DDS por medio de la operación *notify\_change.*
71. La operación  *notify\_change* retorna.
72. La operación *add\_change* retorna.
73. El Usuario llama a la operación *take* para obtener los datos desde el *DataReader* DDS.
74. El *DataReader* DDS solicita los cambios por medio de la operación *get\_change*.
75. La operación *get\_change* retorna.
76. La operación *take* retorna. Los datos recibidos son entregados al usuario.
77. El usuario indica al *DataReader* DDS que ya obtuvo el cambio mediante la operación *return\_loan.*
78. El *DataReader* DDS pregunta al *HistoryCache* del *Reader* RTPS si el cambio indicado es relevante mediante la operación *a\_change\_is\_relevant.*
79. La operación *a\_change\_is\_relevant* retorna.
80. Dependiendo si el cambio es relevante el *DataReader* DDS elimina los cambios mediante la operación *remove\_change*.
81. La operación *remove\_change* retorna.
82. La operación *return\_loan* retorna.

### Reliable Reader—Reliable Writer con Key (Packet Failure)

1. El usuario DDS escribe datos mediante la operación *write* en el DataWriter DDS
2. El DataWriter DDS crea un CacheChange mediante la operación new\_change al Stateful Writer.
3. La operación new\_change retorna.
4. El DataWriter DDS añade el cambio mediante la operación add\_change al HistoryCache del Writer RTPS.
5. El HistoryCache del Writer RTPS notifica al Publisher mediante la operación notify\_change.
6. El Publisher DDS indica la disponibilidad de datos al *ReaderProxy* mediante la operación *data\_available.*
7. El *ReaderProxy* serializa la notificación mediante la operación *serialize.*
8. La operación *serialize* retorna.
9. El *ReaderProxy* envía un submensajeHEARTBEAT en modo *waiting* al *MessageEncoder,* que será dirigido al participante uno.
10. El *ReaderProxy* envía un submensajeHEARTBEAT en modo *waiting* al *MessageEncoder,* que será dirigido al participante dos.
11. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
12. La operación *encoded­\_message retorna.*

**Del punto 13 al 18 pertenecen al participante 2**

1. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos, dirigido hacia el participante dos.
2. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
3. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
4. La operación *doDecode* retorna.
5. El *WriterProxy* recibe el HEARTBEAT en modo *waiting* del *MessageDecoder.* El envío de este tipo HEARTBEAT sirve solamente para confirmar la presencia de un participante.
6. El *WriterProxy* deserializa el submensaje mediante la operación *deserialize\_data.*
7. La operación  *deserialize\_data* retorna.

**Del punto 20 al 26 pertenecen al participante 1**

1. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos, dirigido hacia el participante dos.
2. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
3. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
4. La operación *doDecode* retorna.
5. El *WriterProxy* recibe el HEARTBEAT en modo *waiting* del *MessageDecoder.* El envío de este tipo HEARTBEAT sirve solamente para confirmar la presencia de un participante.
6. El *WriterProxy* deserializa el submensaje mediante la operación *deserialize\_data.*
7. La operación  *deserialize\_data* retorna.
8. La operación *notify\_change*  retorna.
9. La operación *add\_change­* retorna.
10. La operación *write* retorna. El usuario ha completado la acción de escritura de datos.
11. El *HistoryCache*  del *Writer* DDS utiliza la operación *unsent\_changes* para informar al *ReaderProxy* que hay cambios o información no enviada.
12. La operación *unsent\_changes* retorna.
13. El *HistoryCache*  del *Writer* DDS utiliza la operación *can\_send* para informar al *ReaderProxy* que puede enviar los cambios.
14. La operación *can\_send* retorna.
15. El ReaderProxy serializa los datos mediante la operación *serialize*.
16. La operación serialize retorna.
17. El *ReaderProxy* envía al *MessageEncoder* los submensajes GAP, DATA y dependiendo de la política de QoS se envía también un INFO\_TIMESTAMP, va dirigido al suscriptor uno.
18. El *ReaderProxy* envía al *MessageEncoder* los submensajes GAP, DATA y dependiendo de la política de QoS se envía también un INFO\_TIMESTAMP, va dirigido al suscriptor dos.
19. El *HistoryCache* del *Writer* RTPS reenvía los números de secuencia de los cambios que no han sido confirmados mediante la operación *unacked\_changes*, para que sean añadidos al submensaje HEARTBEAT.
20. La operación *unacked\_changes* retorna.
21. El *ReaderProxy* envía al *MessageEncoder* el submensaje HEARTBEAT. Este HEARTBEAT tiene un temporizador llamado *Heartbeat period* en el cual se debería recibir las confirmaciones, va dirigido al suscriptor uno.
22. El *ReaderProxy* envía al *MessageEncoder* el submensaje HEARTBEAT. Este HEARTBEAT tiene un temporizador llamado *Heartbeat period* en el cual se debería recibir las confirmaciones, va dirigido al suscriptor dos.

**Del punto 42 al 59 pertenece al suscriptor 2**

1. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
2. La operación *encoded­\_message retorna.*
3. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
4. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
5. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
6. La operación *doDecode* retorna.
7. El *WriterProxy* recibe el submensaje HEARTBEAT desde el *MessageDecoder.*
8. El *WriterProxy* deserializa el submensaje por medio de la operación *deserialize\_data.*
9. La operación  *deserialize\_data* retorna.
10. El *WriterProxy* obtiene una lista de los cambios que se han perdido por medio de la operación *missing\_changes* al *HistoryCache* del *Reader* RTPS.
11. La operación *missing\_changes* retorna.
12. Los números de secuencia faltantes son serializados en el *WriterProxy* mediante la operación *serialize.*
13. La operación *serialize* retorna.
14. El *WriterProxy* envía los submensajes INFO\_DESTINATION y ACKNACK con la confirmación de recepción o pérdida de paquetes.
15. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
16. La operación *encoded­\_message retorna.*
17. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
18. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.

**Del punto 60 al 77 pertenece al suscriptor 1**

1. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
2. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos. El mensaje se corrompe.
3. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
4. La operación *doDecode* retorna.
5. El *WriterProxy* recibe el submensaje HEARTBEAT desde el *MessageDecoder.*
6. El *WriterProxy* deserializa el submensaje por medio de la operación *deserialize\_data.*
7. La operación  *deserialize\_data* retorna.
8. El *WriterProxy* obtiene una lista de los cambios que se han perdido por medio de la operación *missing\_changes* al *HistoryCache* del *Reader* RTPS.
9. La operación *missing\_changes* retorna.
10. Los números de secuencia faltantes son serializados en el *WriterProxy* mediante la operación *serialize.*
11. La operación *serialize* retorna.
12. El *WriterProxy* envía los submensajes INFO\_DESTINATION y ACKNACK con la confirmación de recepción o pérdida de paquetes.
13. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
14. La operación *encoded­\_message retorna.*
15. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
16. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
17. Una vez recibido los mensajes UDP, el *UDPReceiver* desencapsula los mensaje mediante la operación *doDecode*  en el *MessageDecoder.*
18. La operación *doDecode* retorna.
19. El *ReaderProxy* recibe los submensajes INFO\_DESTINATION y ACKNACK desde el *MessageEncoder,* del suscriptor uno.El submensaje INFO\_DESTINATION contiene el destino el cual ha confirmado el cambio.
20. El *ReaderProxy* recibe los submensajes INFO\_DESTINATION y ACKNACK desde el *MessageEncoder,* del suscriptor dos.El submensaje INFO\_DESTINATION contiene el destino el cual ha confirmado el cambio.
21. El *ReaderProxy* deserializa el submensaje por medio de la operación *deserialize\_data.*
22. La operación  *deserialize\_data* retorna.
23. El *HistoryCache* del *Writer* RTPS pregunta si hay cambios confirmados mediante la operación *requested\_changes* al *ReaderProxy.*
24. El  *ReaderProxy* envía los cambios confirmados y no confirmados al *Stateful Writer* mediante la operación *acked\_changes.*
25. La operación *acked\_changes* retorna.
26. La operación *requested\_changes* retorna.

**Del punto 86 al 102 pertenece al suscriptor 2**

1. Luego de recibir el HEARTBEAT en el lado del suscriptor. El *WriterProxy* recibe los submensajes GAP, INFO\_TIMESTAMP y DATA del *MessageDecoder.*
2. El *WriterProxy* genera un nuevo *CacheChange* con la operación *new\_change*  en el *Reader* RTPS.
3. La operación *new\_change* retorna.
4. El *Stateful Reader* añade el cambio al *HistoryCache* del *Reader* RTPS por medio de la operación *add\_change.*
5. El *HistoryCache* del *Reader* RTPS notifica el cambio al Suscriptor DDS por medio de la operación *notify\_change.*
6. La operación  *notify\_change* retorna.
7. La operación *add\_change* retorna.
8. El Usuario llama a la operación *take* para obtener los datos desde el *DataReader* DDS.
9. El *DataReader* DDS solicita los cambios por medio de la operación *get\_change*.
10. La operación *get\_change* retorna.
11. La operación *take* retorna. Los datos recibidos son entregados al usuario.
12. El usuario indica al *DataReader* DDS que ya obtuvo el cambio mediante la operación *return\_loan.*
13. El *DataReader* DDS pregunta al *HistoryCache* del *Reader* RTPS si el cambio indicado es relevante mediante la operación *a\_change\_is\_relevant.*
14. La operación *a\_change\_is\_relevant* retorna.
15. Dependiendo si el cambio es relevante el *DataReader* DDS elimina los cambios mediante la operación *remove\_change*.
16. La operación *remove\_change* retorna.
17. La operación *return\_loan* retorna.

**Desde el punto 103 pertenece al suscriptor 1**

1. Luego de recibir el HEARTBEAT en el lado del suscriptor. El *WriterProxy* recibe los submensajes GAP, INFO\_TIMESTAMP y DATA del *MessageDecoder.*
2. El *HistoryCache* del *Writer* RTPS reenvía los números de secuencia de los cambios que no han sido confirmados mediante la operación *unacked\_changes*, para que sean añadidos al submensaje HEARTBEAT.
3. La operación *unacked\_changes* retorna.
4. El *ReaderProxy* reenvía al *MessageEncoder* los submensajes GAP, DATA y dependiendo de la política de QoS se envía también un INFO\_TIMESTAMP.
5. El *ReaderProxy* reenvía al *MessageEncoder* el submensaje HEARTBEAT.
6. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
7. La operación *encoded­\_message retorna.*
8. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
9. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
10. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
11. La operación *doDecode* retorna.
12. El *WriterProxy* recibe el submensaje HEARTBEAT desde el *MessageDecoder.*
13. El *WriterProxy* deserializa el submensaje por medio de la operación *deserialize\_data.*
14. La operación  *deserialize\_data* retorna.
15. El *WriterProxy* obtiene una lista de los cambios que se han perdido por medio de la operación *missing\_changes* al *HistoryCache* del *Reader* RTPS.
16. La operación *missing\_changes* retorna.
17. Los números de secuencia faltantes son serializados en el *WriterProxy* mediante la operación *serialize.*
18. La operación *serialize* retorna.
19. El *WriterProxy* envía los submensajes INFO\_DESTINATION y ACKNACK con la confirmación de recepción o pérdida de paquetes.
20. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
21. La operación *encoded­\_message retorna.*
22. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
23. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
24. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
25. La operación *doDecode* retorna.
26. El *ReaderProxy* recibe los submensajes INFO\_DESTINATION y ACKNACK desde el *MessageEncoder.* El submensaje INFO\_DESTINATION contiene el destino el cual ha confirmado el cambio.
27. El *ReaderProxy* deserializa el submensaje por medio de la operación *deserialize\_data.*
28. La operación  *deserialize\_data* retorna.
29. El *ReaderProxy* envía al *Stateful Writer* los cambios que han sido confirmados mediante la operación *acked\_changes*.
30. La operación *acked\_changes* retorna.
31. El *DataWriter* DDS consulta mediante la operación *is\_acked\_by\_all* al *Stateful Writer* que todos los suscriptores tienen los cambios.
32. La operación *is\_acked\_by\_all* retorna.
33. El *DataWriter* DDS elimina los cambios cuando todos suscriptores han recibido los cambios por medio de la operación *remove\_change* al *HistoryCache* del *Writer* RTPS.
34. La operación  *remove\_change* retorna.
35. Este literal toma lugar después del punto 114, luego de recibir el HEARTBEAT en el lado del suscriptor. El *WriterProxy* recibe los submensajes GAP, INFO\_TIMESTAMP y DATA del *MessageDecoder.*
36. El *WriterProxy* genera un nuevo *CacheChange* con la operación *new\_change*  en el *Reader* RTPS.
37. La operación *new\_change* retorna.
38. El *Stateful Reader* añade el cambio al *HistoryCache* del *Reader* RTPS por medio de la operación *add\_change.*
39. El *HistoryCache* del *Reader* RTPS notifica el cambio al Suscriptor DDS por medio de la operación *notify\_change.*
40. La operación  *notify\_change* retorna.
41. La operación *add\_change* retorna.
42. El Usuario llama a la operación *take* para obtener los datos desde el *DataReader* DDS.
43. El *DataReader* DDS solicita los cambios por medio de la operación *get\_change*.
44. La operación *get\_change* retorna.
45. La operación *take* retorna. Los datos recibidos son entregados al usuario.
46. El usuario indica al *DataReader* DDS que ya obtuvo el cambio mediante la operación *return\_loan.*
47. El *DataReader* DDS pregunta al *HistoryCache* del *Reader* RTPS si el cambio indicado es relevante mediante la operación *a\_change\_is\_relevant.*
48. La operación *a\_change\_is\_relevant* retorna.
49. Dependiendo si el cambio es relevante el *DataReader* DDS elimina los cambios mediante la operación *remove\_change*.
50. La operación *remove\_change* retorna.
51. La operación *return\_loan* retorna.

## Diagramas basados con la QoS reliable – Best Effort

### Reliable Writer—Best Effort Reader con Key

1. El usuario DDS escribe datos mediante la operación *write* en el DataWriter DDS
2. El DataWriter DDS crea un CacheChange mediante la operación new\_change al Stateful Writer.
3. La operación new\_change retorna.
4. El DataWriter DDS añade el cambio mediante la operación add\_change al HistoryCache del Writer RTPS.
5. El HistoryCache del Writer RTPS notifica al Publisher mediante la operación notify\_change.
6. El Publisher DDS indica la disponibilidad de datos al *ReaderProxy* mediante la operación *data\_available.*
7. El *ReaderProxy* serializa la notificación mediante la operación *serialize.*
8. La operación *serialize* retorna.
9. El *ReaderProxy* envía un submensajeHEARTBEAT en modo *waiting* al *MessageEncoder.*
10. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
11. La operación *encoded­\_message retorna.*
12. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
13. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
14. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
15. La operación *doDecode* retorna.
16. El *WriterProxy* recibe el HEARTBEAT en modo *waiting* del *MessageDecoder.* El envío de este tipo HEARTBEAT sirve solamente para confirmar la presencia de un participante.
17. El *WriterProxy* deserializa el submensaje mediante la operación *deserialize\_data.*
18. La operación  *deserialize\_data* retorna.
19. La operación *notify\_change*  retorna.
20. La operación *add\_change­* retorna.
21. La operación *write* retorna. El usuario ha completado la acción de escritura de datos.
22. El *HistoryCache*  del *Writer* DDS utiliza la operación *unsent\_changes* para informar al *ReaderProxy* que hay cambios o información no enviada.
23. La operación *unsent\_changes* retorna.
24. El *HistoryCache*  del *Writer* DDS utiliza la operación *can\_send* para informar al *ReaderProxy* que puede enviar los cambios.
25. La operación *can\_send* retorna.
26. El ReaderProxy serializa los datos mediante la operación *serialize*.
27. La operación serialize retorna.
28. El *ReaderProxy* envía al *MessageEncoder* los submensajes GAP, DATA y dependiendo de la política de QoS se envía también un INFO\_TIMESTAMP.
29. El *HistoryCache* del *Writer* RTPS reenvía los números de secuencia de los cambios que no han sido confirmados mediante la operación *unacked\_changes*, para que sean añadidos al submensaje HEARTBEAT. Esta operación asume que todos los mensajes son confirmados cuando se trabaja con suscriptores con mejor esfuerzo.
30. La operación *unacked\_changes* retorna.
31. El *ReaderProxy* envía al *MessageEncoder* el submensaje HEARTBEAT. Este HEARTBEAT tiene un temporizador llamado *Heartbeat period* en el cual se debería recibir las confirmaciones.
32. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
33. La operación *encoded­\_message retorna.*
34. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
35. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
36. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
37. La operación *doDecode* retorna.
38. El *WriterProxy* recibe el submensaje HEARTBEAT desde el *MessageDecoder.*
39. El *WriterProxy* recibe el submensaje GAP Y DATA desde el *MessageDecoder.*
40. El *WriterProxy* llama a la operación *deserialize\_data* al *Deserializer*
41. La operación *deserialize\_data* retorna.
42. El *WriterProxy* genera un nuevo *CacheChange* con la operación *new\_change*  en el *Reader* RTPS.
43. La operación *new\_change* retorna.
44. El *Stateful Reader* añade el cambio al *HistoryCache* del *Reader* RTPS por medio de la operación *add\_change.*
45. El *HistoryCache* del *Reader* RTPS notifica el cambio al Suscriptor DDS por medio de la operación *notify\_change.*
46. La operación  *notify\_change* retorna.
47. La operación *add\_change* retorna.
48. El Usuario llama a la operación *take* para obtener los datos desde el *DataReader* DDS.
49. El *DataReader* DDS solicita los cambios por medio de la operación *get\_change*.
50. La operación *get\_change* retorna.
51. La operación *take* retorna. Los datos recibidos son entregados al usuario.
52. Una vez obtenido los cambios el *DataReader* DDS elimina los cambios mediante la operación *remove\_change*.
53. La operación *remove\_change* retorna.
54. El *DataWriter* DDS consulta mediante la operación *is\_acked\_by\_all* al *Stateful Writer* que todos los suscriptores tienen los cambios.
55. La operación *is\_acked\_by\_all* retorna.
56. El *DataWriter* DDS elimina los cambios cuando todos suscriptores han recibido los cambios por medio de la operación *remove\_change* al *HistoryCache* del *Writer* RTPS.
57. La operación  *remove\_change* retorna.

### Reliable Writer—Best Effort Reader con Key (Packet Failure)

1. El usuario DDS escribe datos mediante la operación *write* en el DataWriter DDS
2. El DataWriter DDS crea un CacheChange mediante la operación new\_change al Stateful Writer.
3. La operación new\_change retorna.
4. El DataWriter DDS añade el cambio mediante la operación add\_change al HistoryCache del Writer RTPS.
5. El HistoryCache del Writer RTPS notifica al Publisher mediante la operación notify\_change.
6. El Publisher DDS indica la disponibilidad de datos al *ReaderProxy* mediante la operación *data\_available.*
7. El *ReaderProxy* serializa la notificación mediante la operación *serialize.*
8. La operación *serialize* retorna.
9. El *ReaderProxy* envía un submensajeHEARTBEAT en modo *waiting* al *MessageEncoder.*
10. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
11. La operación *encoded­\_message retorna.*
12. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
13. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
14. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
15. La operación *doDecode* retorna.
16. El *WriterProxy* recibe el HEARTBEAT en modo *waiting* del *MessageDecoder.* El envío de este tipo HEARTBEAT sirve solamente para confirmar la presencia de un participante.
17. El *WriterProxy* deserializa el submensaje mediante la operación *deserialize\_data.*
18. La operación  *deserialize\_data* retorna.
19. La operación *notify\_change*  retorna.
20. La operación *add\_change­* retorna.
21. La operación *write* retorna. El usuario ha completado la acción de escritura de datos.
22. El *HistoryCache*  del *Writer* DDS utiliza la operación *unsent\_changes* para informar al *ReaderProxy* que hay cambios o información no enviada.
23. La operación *unsent\_changes* retorna.
24. El *HistoryCache*  del *Writer* DDS utiliza la operación *can\_send* para informar al *ReaderProxy* que puede enviar los cambios.
25. La operación *can\_send* retorna.
26. El ReaderProxy serializa los datos mediante la operación *serialize*.
27. La operación serialize retorna.
28. El *ReaderProxy* envía al *MessageEncoder* los submensajes GAP, DATA y dependiendo de la política de QoS se envía también un INFO\_TIMESTAMP.
29. El *HistoryCache* del *Writer* RTPS reenvía los números de secuencia de los cambios que no han sido confirmados mediante la operación *unacked\_changes*, para que sean añadidos al submensaje HEARTBEAT. Esta operación asume que todos los mensajes son confirmados cuando se trabaja con suscriptores con mejor esfuerzo.
30. La operación *unacked\_changes* retorna.
31. El *ReaderProxy* envía al *MessageEncoder* el submensaje HEARTBEAT. Este HEARTBEAT tiene un temporizador llamado *Heartbeat period* en el cual se debería recibir las confirmaciones.
32. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
33. La operación *encoded­\_message retorna.*
34. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
35. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos. Se simula un paquete corrupto.
36. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
37. La operación *doDecode* retorna.
38. El *WriterProxy* recibe el submensaje HEARTBEAT desde el *MessageDecoder.*
39. El *WriterProxy* recibe el submensaje GAP Y DATA desde el *MessageDecoder.*
40. El Usuario llama a la operación *take* para obtener los datos desde el *DataReader* DDS.
41. El *DataReader* DDS solicita los cambios por medio de la operación *get\_change*.
42. La operación *get\_change* retorna.
43. La operación *take* retorna. Los datos recibidos son entregados al usuario.
44. El *DataWriter* DDS consulta mediante la operación *is\_acked\_by\_all* al *Stateful Writer* que todos los suscriptores tienen los cambios.
45. La operación *is\_acked\_by\_all* retorna.
46. El *DataWriter* DDS elimina los cambios cuando todos suscriptores han recibido los cambios por medio de la operación *remove\_change* al *HistoryCache* del *Writer* RTPS.
47. La operación  *remove\_change* retorna.

# Diagramas de interacción sin estado

## Diagrama basado con la QoS Best Effort

### BestEffort Reader –- Best Effort Writer con KEY

1. El usuario DDS escribe datos por medio de la llamada a la operación *write* en el *DataWriter* DDS.
2. El *DataWriter* DDS llama a la operación *new\_change* en el *Writer* RTPS para crear un nuevo *CacheChange*. Cada uno de estos cambios es identificado únicamente por un *SequenceNumber*.
3. La operación *new\_change* retorna.
4. El *DataWriter* DDS utiliza la operación *add\_change* para almacenar el *CacheChange* dentro de *HistoryCache* del *Writer* RTPS.
5. El *HistoryCache* del *Writer* RTPS notifica el cambio por medio de la operación *notify\_change*  al *Publisher* DDS.
6. La operación *notify\_change*  retorna.
7. La operación *add\_change­* retorna.
8. La operación *write* retorna. El usuario ha completado la acción de escritura de datos.
9. El *HistoryCache*  del *Writer* DDS utiliza la operación *unsent\_changes* para informar al *ReaderLocator* que hay cambios o información no enviada.
10. La operación *unsent\_changes* retorna.
11. El *HistoryCache*  del *Writer* DDS utiliza la operación *can\_send* para informar al *ReaderLocator* que puede enviar los cambios.
12. La operación *can\_send* retorna.
13. El *DataWriter* DDS utiliza la operación *remove\_change*  en el *HistoryCache*  del *Writer* DDS para limpiar la cache. Esta operación puede ser realizada posteriormente.
14. La operación *remove\_change* retorna.
15. El *ReaderLocator* serializa la información mediante la operación *serialize* en el *Serializer*.
16. La operación *serialize*  retorna.
17. El *ReaderLocator* envía al *MessageEncoder* el submensaje DATA y dependiendo de la política de QoS se envía también un INFO\_TIMESTAMP.
18. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
19. La operación *encoded­\_message retorna.*
20. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
21. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
22. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder.*
23. La operación *doDecode* retorna.
24. El *MessageDecoder* envía el submensaje DATA y dependiendo de la política de QoS se envía también un INFO\_TIMESTAMP al *Reader* RTPS.
25. El *Reader* RTPS llama a la operación *deserialize\_data* en el *Deserializer.*
26. La operación *deserialize\_data* retorna.
27. El *Stateless Reader* añade el cambio al *HistoryCache* del *Reader* RTPS por medio de la operación *add\_change.*
28. El *HistoryCache* del *Reader* RTPS notifica el cambio al Suscriptor DDS por medio de la operación *notify\_change.*
29. La operación  *notify\_change* retorna.
30. La operación *add\_change* retorna.
31. El Usuario llama a la operación *take* para obtener los datos desde el *DataReader* DDS.
32. El *DataReader* DDS solicita los cambios por medio de la operación *get\_change* al *HistoryCache* del *Reader* RTPS.
33. La operación *get\_change* retorna.
34. La operación *take* retorna. Los datos recibidos son entregados al usuario.
35. Una vez obtenido los cambios el *DataReader* DDS elimina los cambios mediante la operación *remove\_change*.
36. La operación *remove\_change* retorna.

## Diagrama basado con la QoS Reliable – Best Effort

### Reliable Writer – Best Effort Reader con KEY

1. El usuario DDS escribe datos mediante la operación *write* en el DataWriter DDS
2. El DataWriter DDS crea un CacheChange mediante la operación *new\_change* al Stateful Writer.
3. La operación *new\_change* retorna.
4. El DataWriter DDS añade el cambio mediante la operación *add\_change* al HistoryCache del Writer RTPS.
5. El HistoryCache del Writer RTPS notifica al Publisher mediante la operación *notify\_change*.
6. El Publisher DDS indica la disponibilidad de datos al *ReaderLocator* mediante la operación *data\_available.*
7. El *ReaderLocator* serializa la notificación mediante la operación *serialize.*
8. La operación *serialize* retorna.
9. El *ReaderLocator* envía un submensajeHEARTBEAT en modo *waiting* al *MessageEncoder.*
10. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
11. La operación *encoded­\_message retorna.*
12. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
13. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
14. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
15. La operación *doDecode* retorna.
16. El *Stateless Reader* recibe el HEARTBEAT en modo *waiting* del *MessageDecoder.* El envío de este tipo HEARTBEAT sirve solamente para confirmar la presencia de un participante.
17. El *Stateless Reader* deserializa el submensaje mediante la operación *deserialize\_data.*
18. La operación  *deserialize\_data* retorna.
19. La operación *notify\_change*  retorna.
20. La operación *add\_change­* retorna.
21. La operación *write* retorna. El usuario ha completado la acción de escritura de datos.
22. El *HistoryCache*  del *Writer* DDS utiliza la operación *unsent\_changes* para informar al *ReaderProxy* que hay cambios o información no enviada.
23. La operación *unsent\_changes* retorna.
24. El *HistoryCache*  del *Writer* DDS utiliza la operación *can\_send* para informar al *ReaderLocator* que puede enviar los cambios.
25. La operación *can\_send* retorna.
26. El *ReaderLocator* serializa los datos mediante la operación *serialize*.
27. La operación serialize retorna.
28. El *ReaderLocator* envía al *MessageEncoder* los submensajes DATA.
29. El *ReaderLocator* envía al *MessageEncoder* el submensaje HEARTBEAT. Este HEARTBEAT tiene un temporizador llamado *Heartbeat period* en el cual se debería recibir las confirmaciones.
30. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
31. La operación *encoded­\_message retorna.*
32. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
33. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
34. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
35. La operación *doDecode* retorna.
36. El *Stateless Reader* recibe el submensaje HEARTBEAT desde el *MessageDecoder.*
37. El *Stateless Reader* recibe el submensaje DATA desde el *MessageDecoder.*
38. El *Stateless Reader* llama a la operación *deserialize\_data* al *Deserializer*
39. La operación *deserialize\_data* retorna.
40. El *Stateless Reader* añade el cambio al *HistoryCache* del *Reader* RTPS por medio de la operación *add\_change.*
41. El *HistoryCache* del *Reader* RTPS notifica el cambio al Suscriptor DDS por medio de la operación *notify\_change.*
42. La operación  *notify\_change* retorna.
43. La operación *add\_change* retorna.
44. El Usuario llama a la operación *take* para obtener los datos desde el *DataReader* DDS.
45. El *DataReader* DDS solicita los cambios por medio de la operación *get\_change*.
46. La operación *get\_change* retorna.
47. La operación *take* retorna. Los datos recibidos son entregados al usuario.
48. El *HistoryCache* del *Writer* RTPS solicita los cambios no confirmados al *ReaderLocator* mediante la operación *requested\_changes.* Como se está trabajando con un *Reader* sin estado con mejor esfuerzo, este no confirma ningún cambio por lo cual en esta operación se assume que todo está confirmado.
49. La operación *requested\_changes* retorna.
50. El *ReaderLocator* envía al *MessageEncoder* el submensaje GAP. Este submensaje en este caso no solicitará ningún número de secuencia.
51. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
52. La operación *encoded­\_message retorna.*
53. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
54. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
55. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
56. La operación *doDecode* retorna.
57. El *Stateless Reader* recibe el submensaje GAP desde el *MessageDecoder.*
58. El *Stateless Reader* llama a la operación *deserialize\_data* al *Deserializer*
59. La operación *deserialize\_data* retorna.
60. Una vez obtenido los cambios el *DataReader* DDS elimina los cambios mediante la operación *remove\_change*.
61. La operación *remove\_change* retorna.

## Diagramas híbridos (con estado y sin estado)

### Reliable Stateless Writer – Reliable Stateful Reader con Key

1. El usuario DDS escribe datos mediante la operación *write* en el DataWriter DDS
2. El DataWriter DDS crea un CacheChange mediante la operación *new\_change* al Stateful Writer.
3. La operación *new\_change* retorna.
4. El DataWriter DDS añade el cambio mediante la operación *add\_change* al HistoryCache del Writer RTPS.
5. El HistoryCache del Writer RTPS notifica al Publisher mediante la operación *notify\_change*.
6. El Publisher DDS indica la disponibilidad de datos al *ReaderLocator* mediante la operación *data\_available.*
7. El *ReaderLocator* serializa la notificación mediante la operación *serialize.*
8. La operación *serialize* retorna.
9. El *ReaderLocator* envía un submensajeHEARTBEAT en modo *waiting* al *MessageEncoder.*
10. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
11. La operación *encoded­\_message retorna.*
12. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
13. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
14. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
15. La operación *doDecode* retorna.
16. El *WriterProxy* recibe el HEARTBEAT en modo *waiting* del *MessageDecoder.* El envío de este tipo HEARTBEAT sirve solamente para confirmar la presencia de un participante.
17. El *WriterProxy* deserializa el submensaje mediante la operación *deserialize\_data.*
18. La operación  *deserialize\_data* retorna.
19. La operación *notify\_change*  retorna.
20. La operación *add\_change­* retorna.
21. La operación *write* retorna. El usuario ha completado la acción de escritura de datos.
22. El *HistoryCache*  del *Writer* DDS utiliza la operación *unsent\_changes* para informar al *ReaderProxy* que hay cambios o información no enviada.
23. La operación *unsent\_changes* retorna.
24. El *HistoryCache*  del *Writer* DDS utiliza la operación *can\_send* para informar al *ReaderLocator* que puede enviar los cambios.
25. La operación *can\_send* retorna.
26. El *ReaderLocator* serializa los datos mediante la operación *serialize*.
27. La operación serialize retorna.
28. El *ReaderLocator* envía al *MessageEncoder* los submensajes DATA.
29. El *ReaderLocator* envía al *MessageEncoder* el submensaje HEARTBEAT. Este HEARTBEAT tiene un temporizador llamado *Heartbeat period* en el cual se debería recibir las confirmaciones.
30. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
31. La operación *encoded­\_message retorna.*
32. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
33. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
34. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
35. La operación *doDecode* retorna.
36. El *WriterProxy* recibe el submensaje HEARTBEAT desde el *MessageDecoder.*
37. El *WriterProxy* recibe el submensaje DATA desde el *MessageDecoder.*
38. El *WriterProxy* deserializa los datos mediante la operación *deserialize\_data.*
39. La operación *deserialize\_data* retorna.
40. El *WriterProxy* genera un nuevo *CacheChange* con la operación *new\_change*  en el *Reader* RTPS.
41. La operación *new\_change* retorna.
42. El *Stateful Reader* añade el cambio al *HistoryCache* del *Reader* RTPS por medio de la operación *add\_change.*
43. El *HistoryCache* del *Reader* RTPS notifica el cambio al Suscriptor DDS por medio de la operación *notify\_change.*
44. La operación  *notify\_change* retorna.
45. La operación *add\_change* retorna.
46. El Usuario llama a la operación *take* para obtener los datos desde el *DataReader* DDS.
47. El *DataReader* DDS solicita los cambios por medio de la operación *get\_change*.
48. La operación *get\_change* retorna.
49. La operación *take* retorna. Los datos recibidos son entregados al usuario.
50. El *WriterProxy* envía la confirmación de los datos mediante un submensaje ACKNACK e indica el destinatario mediante el submensaje INFO\_REPLY. No se utiliza el submensaje INFO\_DESTINATION ya que el publicador es sin estado.
51. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
52. La operación *encoded­\_message retorna.*
53. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
54. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
55. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
56. La operación *doDecode* retorna.
57. El *ReaderLocator* recibe los submensajes INFO\_REPLY y ACKNACK desde el *MessageEncoder.* El submensaje INFO\_DESTINATION contiene el destino el cual ha confirmado el cambio.
58. El *ReaderLocator* deserializa el submensaje por medio de la operación *deserialize\_data.*
59. La operación  *deserialize\_data* retorna.
60. El *HistoryCache* del *Writer* RTPS solicita los cambios no confirmados al *ReaderLocator* mediante la operación *requested\_changes.*
61. La operación *requested\_changes* retorna.
62. El *ReaderLocator* envía al *MessageEncoder* el submensaje GAP. Este submensaje en este caso no solicitará ningún número de secuencia.
63. El *MessageEncoder* encapsula el mensaje y lo envía mediante la operación *encoded\_message­* al *UDPTransmitter*.
64. La operación *encoded­\_message retorna.*
65. El *UDPTransmitter* envía el mensaje *UDP\_Message* hacia la red de Datos.
66. El *UDPReceiver* recibe el mensaje *UDP\_Message* desde la red de Datos.
67. El *UDPReceiver* desencapsula el mensaje mediante la operación *doDecode*  en el *MessageDecoder*
68. La operación *doDecode* retorna.
69. El *WriterProxy* recibe el submensaje GAP desde el *MessageDecoder.*
70. El *WriterProxy* llama a la operación *deserialize\_data* al *Deserializer*
71. La operación *deserialize\_data* retorna.
72. Una vez obtenido los cambios el *DataReader* DDS elimina los cambios mediante la operación *remove\_change*.
73. La operación *remove\_change* retorna.

# Protocolo Descubrimiento

## Resumen de tráfico de Descubrimiento

**Fase de Descubrimiento de Participantes.**

1. Participante 1 ha sido creado.
2. El participante 1 se anuncia enviando mensajes SPDP.
3. El Participante 2 es creado.
4. El participante 2 se anuncia enviando mensajes SPDP.
5. El participante 1 recibe los paquetes SPDP y en este caso añade al participante 2 a la base de datos.
6. El participante 1 se anuncia enviando mensajes SPDP.
7. El participante 2 recibe los paquetes SPDP y en este caso añade al participante 1 a la base de datos.
8. El participante 2 se anuncia enviando mensajes SPDP.
9. El participante 1 crea un *DataWriter.*
10. El participante 1 envía su publicación por medio de mensajes SEDP.
11. El participante 2 recibe el mensaje SEDP y añade al *DataWriter* remoto a su base de datos.
12. El participante 1 continúa enviando su publicación mediante mensajes SEDP.
13. El participante 1 destruye su *DataWriter.*
14. El participante 1 informa que el *DataWriter* ha sido eliminado enviando mensajes SEDP.
15. El participante 1 es destruido.
16. El participante 1 informa queha sido eliminado enviando mensajes SPDP.
17. El participante 2 recibe el mensaje SPDP y remueve al participante 1 de la base de datos.
18. El participante 2 es destruido.
19. El participante 2 informa queha sido eliminado enviando mensajes SPDP.