H2020 i**P** **O**ver **I**c**N**- the bet**T**er IP (POINT)

Design and implementation of the control plane reliability module

List of Authors:
Yannis Thomas

Extending POINT prototype: Control Plane Reliability

Reliable control messages

A known issue of Blackadder software, which is the basis of the POINT prototype, is the absence of reliability mechanisms that ensure the delivery of control messages. As a result, when it is deployed in congested network with increased rate of lost or corrupted packets, then usually communication cannot be established and data transfers fail.

A. System overview

1. Communication model

The following depicts the control plane messages supported, highlighting the distinct types of messages that must be reliable so as to guarantee the operability in congested networks.

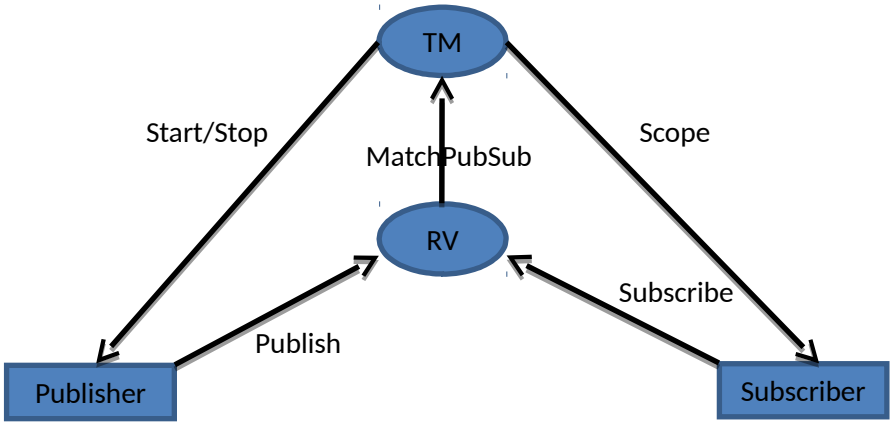

Figure 1: Control messages of service establishment

2. Control plane messages types

In follow, we present all types of inter-click control messages and their unique identifiers that require reliability:

- PUBLISH_SCOPE, PUBLISH_INFO
- SUBSCRIBE_SCOPE, SUBSCRIBE_INFO
- PUBLISH_DATA_ISUB
- START_PUBLISH, STOP_PUBLISH
- UPDATE_FID_ISUB, UPDATE_FID
- SCOPE_PUBLISHED, SCOPE_UNPUBLISHED
- UPDATE_RVFID, UPDATE_TMFID

Note that we do not include PUBLISH_DATA messages in our analysis, since those inter-click packets are meant for currying actual data, thus belonging to the data plane.

3. Structure of control plane messages over the network

We now present the structure of packets that curry control messages over the network.

Figure 2: Inter-click network packet structure

As depicted in Figure 2, five fields compose all network packets.

1. Overlay header: holds the MAC or UPD address of the overlay network
2. FID: keeps the Forwarding Identifier that encodes the unidirectional dissemination path of the overlay network
3. ID_num: keeps the number of the identifiers of the information item that is associated with the request of this message
4. IDs: stores the identifiers of the information item. In case multiple IDs are defined, the header of the packet includes multiple contiguous pairs of <ID_len, ID> fields
5. Payload: holds a message, f.i. the requests described in previous section. The first byte of Payload field also defines the **type** of the request.

The general formation of network packets is instantiated by the control messages as follows (fields like "ID_num" are omitted due to space limitations):

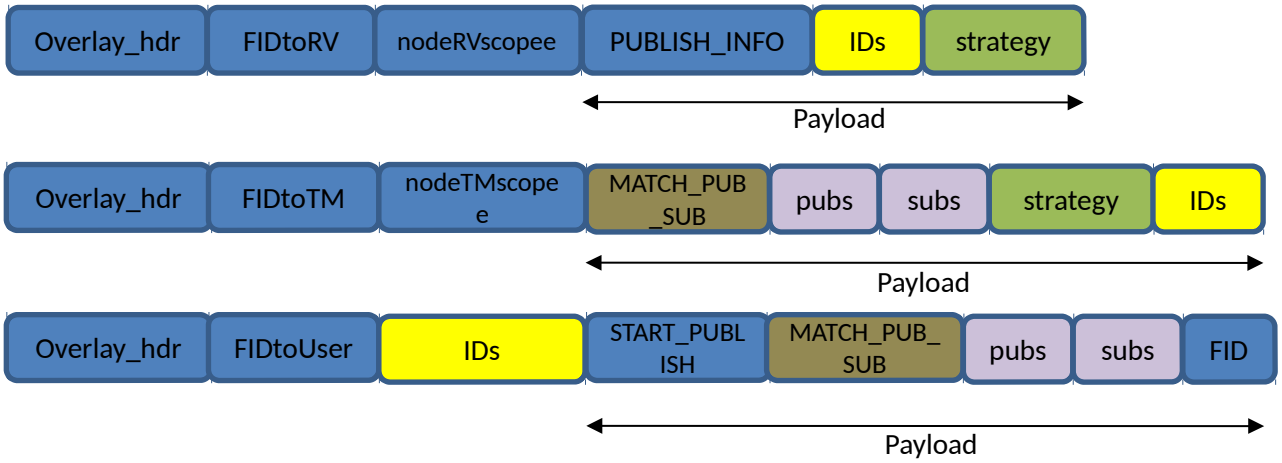

Figure 3: (a) **PUBLISH/SUBSCRIBE (User to RV) message structure, (b) MATCH_PUB_SUB (RV to TM) message structure and (c) START_PUBLISH (TM to User) message structure**

It is worth pointing out that the three messages do not share any "session" identifier in order to indicate that they compose the same session establishment process. This choice is reasonable as it preserves the loose coupling of publish-subscribe pattern. However, the individuality of messages can also complicate functions that are applied at service scope, such as reliable delivery of control messages for the entire rendezvous process (see Chained Reliability).

B. Requirements

We now list the core requirements of the reliability mechanism:

- **High performance:** The reliability mechanism must be *fast* and *scalable*. It must not require keeping state at the in-network routers and must not penalize the forwarding performance of LIPSIN. Therefore, reliability will exploit end-to-end acknowledgments, thus being transparent to forwarders and requiring state only to the sender and receiver nodes.
- **Optional**: Due to the potential overhead of the reliability mechanism, reliable control plane must be explicitly defined at node-level or at request-level. If a node is defined as *control plane reliable* then all control messages coming from local applications will be reliably delivered. Otherwise, reliability will be offered only to the requests that are marked as reliable by the applications.
- **Unchained**: The reliability mechanism must avoid the carriage of the reliability request across all four control messages of the rendezvous process (*chained reliability*)**.** Reliable delivery of control messages must be offered only when it is explicitly demanded by the individual sender of the request (i.e. node or application). For instance, a reliable PUBLISH_INFO does not trigger reliable MATCH_PUB_SUB and START_PUBLISH unless it is explicitly requested by the RV and TM respectively.
- **Informative**: The reliability mechanism must explicitly notify the publisher application about the status of a control message delivery. Special notifications must be pushed to the local application when a control message is acknowledged or when a delivery permanently fails (f.i. due to connectivity issues).
- **Transparent**: The details of the reliability mechanism must be concealed from the applications and the core system. This ensures the durability of the software.
- **Resilient to lost control messages**: The reliability mechanism must provide resilience to lost control messages via an individual retransmission mechanism. This retransmission mechanism must offer timely recovery from lost acknowledgments and avoid overburdening network load.
- **Resilience to lost or late acknowledgments**: The reliability mechanism must recover from lost or late acknowledgments. A lost or late acknowledgment can cause spurious repeated control message retransmissions that waste network resources (increase both data footprint and processing needs).
- **Unicast support:** The reliability mechanism must tackle only unicast transmissions, since all control messages are unicast.

C. Design

1. Acknowledgements

The most common and widely applied mechanisms for reliability are based on acknowledgment messages (ACK). An ACK is a control message that is sent from the receiver back to the sender in order to confirm the successful delivery of a message. An ACK may include an identifier, which indicates the acknowledged message, an integrity parameter that verifies the correct delivery of raw data, and several other feedback information depending on the application needs, such as QoS statistics.

ACKs are usually combined with timers in order to timely detect losses. In this case, the sender keeps state of any pending (unacknowledged) message along with a timer that determines if the message is yet to be delivered or it is lost. The timer is scheduled to expire a certain time after the message departure; that time is often the sum of expected propagation and processing delays of the control message and its ACK. If the timer expires before the corresponding ACK is received, then the control message is considered lost and a retransmission of the request is triggered. In order to minimize the cost of multiple retransmissions, the truncated binary exponential backoff algorithm is also frequently considered.

- **End-to-end reliability**

We assume acknowledgment mechanisms that are applied at the communication end-points, relieving the core network from additional memory and processing costs. According to this design, when the receiver (f.i. RV) gets a message, then he responds with an ACK directly to the original sender (f.i. publisher application). The ACK message is transparent to the intermediate network nodes and does not affect in any way the forwarding process.

This approach places complexity at the network edges; the end-hosts are only aware of the reliability mechanism and the core routers remain state-less. Stateless routers are quite important for our design since they enhance scalability and forwarding performance. A second gain of end-to-end reliability is increased resilience to link failures via dynamic routing. In detail, when a network link fails, then the sender can (request and) use a different FID for delivering the request to the destination.

These advantages are not met in in-network reliability where core routers acknowledge messages at per-hop basis. First, the network routers are obliged to include state for keeping the list of pending control messages. Second, dynamic routing is not supported because the on-path routers are simplified forwarding nodes without routing logic.

- **Unique Identifier**

A unique identifier that binds a control message with the respective acknowledgment is crucial to our design. This acknowledgment ID (AckID) is a random 32-bit nonce that is generated by the control message sender, it is written to the control message and it is also copied to the respective acknowledgment. Consequently, the AckID determines which control message has been successfully delivered in case of multiple pending requests. The AckID is also used for handling lost acknowledgments and repeated control messages, as discussed in next sections.

2. Packet structures

We now discuss the formation of reliable control packets and acknowledgements.

- ### Acknowledgment

The required fields that form an acknowledgment packet are:

New inter-click message type: Inter-click packets are categorized in *data packets* and *control packets* based on their request type. The first exploit message type PUBLISH_DATA, while the later use the types specified in Sect.””. Thereafter, we introduce a new inter-click message type, that is called “CTRL_ACK” and is used for distinguishing control plane acknowledgments. CTRL _ACK is used for acknowledging PUBLISH_INFO, SUBSCRIBE_INFO etc.

Acknowledgment identifier (AckID): The AckID binds control and the corresponding acknowledgment message. As discussed in Sect. “”, AckID is essential for handling multiple acknowledgments from multiple nodes.

Reserved reliability scope: This scope will only be used for sending acknowledgments. Although, it is not necessary for the reliability mechanism, it is proposed for compliance with the packet structure (control messages use predefined RV and TM scopes).

Figure 4: Packet structure of ACK message

- ### Control message

Besides the introduction of the acknowledgment packet, extensions must take place at the control messages. The extensions must be placed right after the network header (FID + IDs fields) of the inter-click packet (see Sect.””), thus inserting the reliability header before the packets payload.

Additionally, the reliability header must be inserted right before a packet is sent over the network and must be removed right after it is received from the network, therefore being transparent to core system. We, hereby, discuss the necessary extensions.

Acknowledgment identifier (AckID): The AckID binds control and the corresponding acknowledgment message. As discussed in Sect.””, AckID is essential for coupling ACKs with control packets.

Reliability Flag: Given that reliability is optional, a packet should be explicitly marked as reliable. The types of messages are currently categorized by the *type* field, which is the first byte of packet’s payload. Therefore, a new “type” must be inserted in order to indicate the reliability request.

Reverse FID: Although POINT prototype eventually will exploit bidirectional FIDs, currently FIDs are unidirectional hence packets need to curry the reverse FID as well. This field is implemented albeit it is not depicted in the related following figure.

The figure below provides the packet structure of a reliable MATCH_PUB_SUB message. The other control messages follow the same notation.

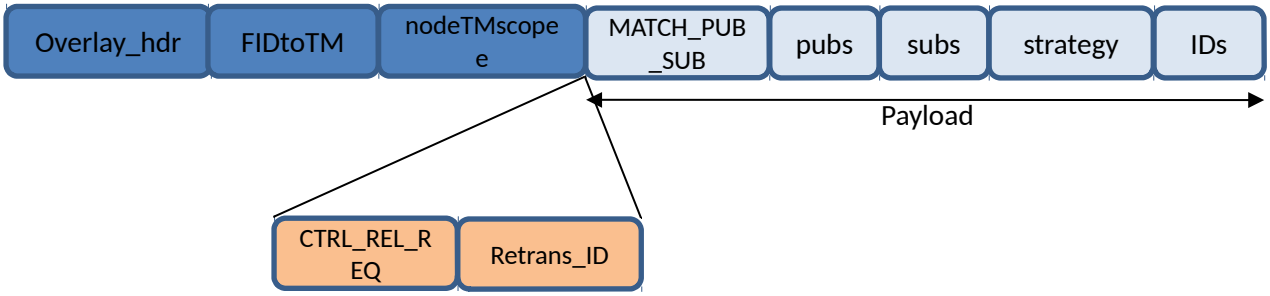

Figure 5: Reliable MATCH_PUB_SUB packet structure

3. End-host state

End-to-end acknowledgments moves computation and memory requirements needs from in-network routers to the communication end-points. Thereupon, the sender and receiver need to store and process certain state, such as pending messages and last acknowledged packets, in order to properly verify accomplished deliveries and also handle late or repeated ACKs.

Receiver: The receiver node must keep a list of the AckIDs of the last received control messages. This information is important for overcoming problems, such as lost acknowledgments. Figure below depicts the message sequence of a reliable PUBLISH_INFO delivery. In an errorless scenario, the Sender sends the request at time t0, receives the correlated acknowledgment at t3 and, finally, gets the START_PUBLISH at t7. Nevertheless, if the ACK message is lost, then the request will be retransmitted at t4 and will be received by the RV at t6. Assuming that the receiver does not maintain a list of the AckIDs of the N last received control messages, then the rendezvous functionality will be falsely triggered twice, thus wasting network resources and possibly causing inconsistencies at the applications' state.

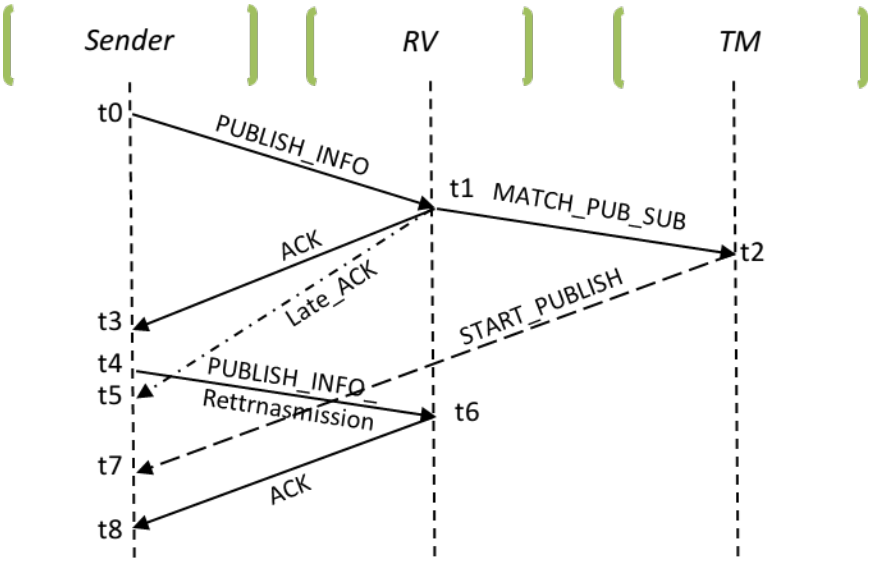

Figure 3: Sequence diagram of intra-click messages

Sender: The sender node must store unacknowledged messages along with a timer for detecting losses and sending retransmissions. Specifically, every transmitted control message is stored along with a timer that expires after a certain time period; probably the expected time to send, receive and process the reliability messages. If the timer expires without having received the associated acknowledgment, then the packet is retransmitted (with the appropriate "retransmission id") and the timer is updated. If the correlated acknowledgment is received before timer expiration, then the message is deleted.

The design is simple yet effective. Considering late ACKs, we again study the previous figure which demonstrates a relevant scenario. The late acknowledgment is received at t5 after the timer expiration (t4) that triggers the retransmission of the request. The RV's ACK to the retransmission is received at t8, but the pending control message is consumed at t5 by the late ACK. This is not a critical issue as the second ACK gets dropped for not matching to any pending AckID. Another interesting scenario arises when START_PUBLISH is received before the ACK of the PUBLISH_INFO. This may happen if the first ACK is lost and the ACK of the retransmission arrives (t8) after the START_PUBLISH (t7). This paradox is not considered an issue for either POINT prototype or the applications. First, the reliability mechanism is transparent to the rendezvous functionality, therefore the acknowledgments does not affect the state of the POINT prototype and vice versa. Secondly, the applications are not affected because the notification system is non-blocking; i.e., the publisher can process other events while waiting for ACKs or START_PUBLISH.

4. Chain of Reliability

Reliability of control plane can be optional at request and node level. Based on the described communication model of Section "", data delivery follows the successful transmission of at least four control messages (PUBLISH_INFO, SUBSCRIBE_INFO, MATCH_PUB_SUB and START_PUBLISH) that are not explicitly associated; there is not trivial method to determine which pub/sub requests trigger a MATCH_PUB_SUB and START_PUBLISH message. Thereupon, *chained reliability*,i.e. the carriage of the reliability request across all control messages, requires access to the state of the domain RV. This is a rather complicated process for proficient RV implementations, such as distributed RV network.

Consequently, reliability is selected by the sender of the message and is proposed at a sender-to-receiver level, instead of publisher-to-subscriber (following the pub/sub communication model). Specifically, the users request reliable delivery to the RV node, the RV node requests reliable delivery to the TM node and, finally, the TM node requests reliable delivery to the publisher. The reliability of these requests is independent and is explicitly defined by the message sender.

5. FIDs

In order to provide the forwarding ID of the acknowledgment route, the deployment tool of the prototype must generate four (4) FIDs per network node:

- RVFID: 'Node to RV' FID, used by the Node to make pub/sub messages
- rRVFID: 'RV to Node' FID, used by the RV to acknowledge pub/sub messages to the issuing Node
- TMFID: 'Node to TM' FID, used by the RV for sending notifications to the TM
- rTMFID: 'TM to Node' FID, used by the TM to communicate with Pubs/Subs

Note that the last two FIDs are only required by the domain's RV, hence adding them to all nodes may be considered redundant. Indeed, in most cases the rendezvous functionality is assigned to a single node, however it is possible to form a network of RV nodes that run on application layer. This scenario justifies the addition of FIDs to/from TM in the configuration of all network nodes.

D. Implementation

1. Reliability module

The reliability functionality is proposed to be implemented as an individual click element/module, called Control Reliability module (CR) that is placed between the forwarded and the proxy elements of the POINT prototype. The reliability module should not be integrated with the forwarder module, in order to preserve the simplicity and performance advantages of LIPSIN forwarding. Integration with the proxy element must also be avoided for it aimlessly increases the complexity of the system; a network-resilient request does not differentiate the operation of the core functionalities (TM, RV and FW), thus pushing this information to localProxy is redundant and penalizes the source code quality (spaghetti code). All in all, reliability is considered a consistent and autonomous logic block that can be enabled, disabled or even modified while being independent to the state or operation of the existing core library. Such a clean solution increases the durability of the system and eases its maintenance.

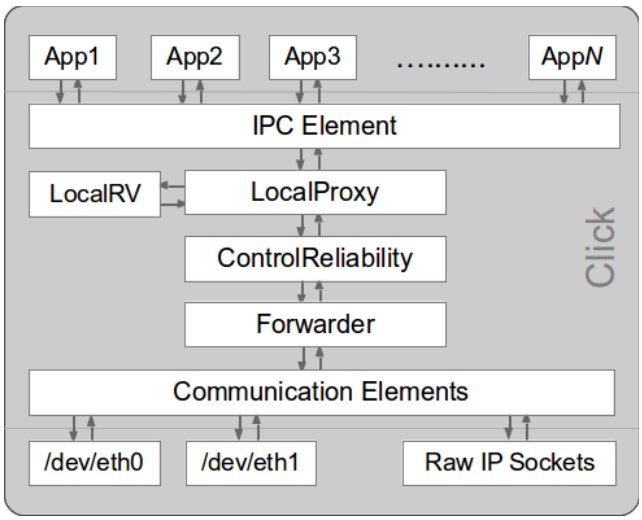

Figure 4: POINT prototype (core) modules

2. Reliability Selection

We identify two orthogonal levels of operation: *intra-click* and *inter-click*. In intra-click, the CR undertakes two tasks: inserting/removing the reliability header and notifying publisher applications about the status of the messages. In the inter-click, the CR manages the delivery of control messages, acknowledgments and, if needed, retransmissions. We further elaborate these in follow.

- **Intra-click**

Packets from the Local Proxy module

All incoming packets from local Proxy module are checked against the following rule. If the condition is true the packets are ignored, otherwise the reliability header is inserted.

```
if ((!gc->type[TM] || rID.substring(0, PURSUIT_ID_LEN)!=(gc→notificationIID.substring
(0,PURSUIT_ID_LEN)) && (packet_type==PUBLISH_DATA || forwardFID != gc->defaultRV_dl))
```

In other words, packets that are:

- not sent from the TM, or
- their Rendezvous ID is not the reserved “notificationIID”

and

- their type is not PUBLISH_DATA, or
- the FID is not the reserved “defaultRV_dl” that forwards packets to the RV

are not reliable.

Packets from the Forwarding module

All incoming packets from Forwarder module are checked against the following rule. If the condition is true the packets are ignored, otherwise the corresponding acknowledgment is emitted.

```
if (_ID.compare(gc->controlReliabilityScope) != 0)
```

In other words, packets that are:

- not using the reserved control plane reliability rendezvous identifier

are not reliable control requests hence are simply forwarded to the LocalProxy module.

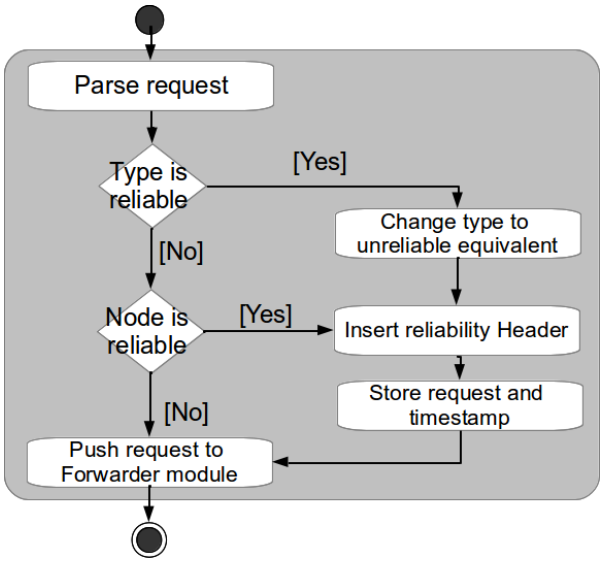

Figure 5: Activity Diagram: ControlReliability receiving request from localProxy

Pushing notifications to apps: The CR must push event notifications to the pub/sub apps upon the successful and/or failed message delivery. Following the operation rationale of the POINT prototype, we consider a reserved scope “/FFFFFFFFFFFFFFR (similarly to the TM and RV scopes) which is used by the CR for publishing reliability-related events to applications. When the localProxy receives the publications,

a new function, called "*void LocalProxy::handleCRNotification(Packet *p);*", handles the local request, creates and sends the appropriate **Blackadder event** to the corresponding application via netlink sockets. Selecting the corresponding application relays upon the rendezvous ID, the dissemination strategy and the list of local applications that are subscribed to that rendezvous ID; this information is available to either the ControlReliability or the localProxy. Finally, the transmitted event must curry the following information for identifying the correlated control message:

- the new event type called "*CONTROL_ACK*" that identifies CR notifications
- the rendezvous ID,
- the dissemination strategy that, combined with rendezvous ID, identifies the request
- the status field that describes the success of the transmission and
- a byte-buffer for metadata.

This extension is absolutely compatible with existing pub/sub applications if they can handle unknown event types, f.i. ignore them. Furthermore, assuming that the structure of the CONTROL_ACK event is implicitly known to the applications, modifying the exposed API (placed at "HOME_DIR/lib/") is not necessary besides defining the new event type in "HOME_DIR/lib/blackadder_defs.h". Consequently, the byte-buffer of a CONTROL_ACK event will have the following formation:

| number_of_ids | <IDlenght | ID> | strategy | status | *determined_by_CR* |

The *determined_by_CR* field is left open for future expansions of CR, f.i. reporting the number of retransmissions.

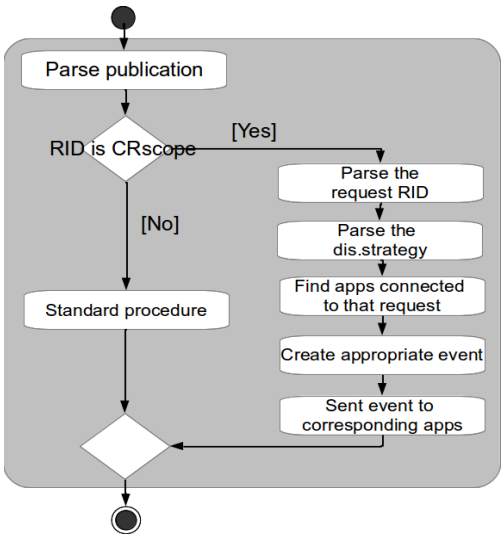

Figure 6: Activity Diagram: LocalProxy receiving publication from ControlReliability

- ### Inter-click

CR modules at the communication end-points guarantee the delivery of control messages transparently to the other core elements. The main duty of the CRs at this point is the insertion and eviction of the reliability header, as well as the retransmission of lost packets.

The procedure followed by the sender's CR is depicted in the following figure. When a request is pushed by the localProxy element, then the CR executes two parallel computations. The first stores the packet to an LRU data structure and forwards it the Forwarder element. The second deploys a timer that expires after a predefined time period, thus indicating that the control message was lost. We exploit an LRU structure because it allows handling time outs of multiple control packets with a single timer. Assuming equal delivery latencies for all control messages and retransmissions, the next control message to expire is the oldest one (LRU's tail). In case of expiration, the timer is updated to reflect the remaining time for the new-oldest packet and the expired packet is either retransmitted or dropped. In case of a retransmission, the packet is relocated to the LRU's head, otherwise it is removed from the LRU and a notification is sent to the publishing application. Each request is stored along with the (re)transmission time, in order to deduce the individual expiration times of requests.

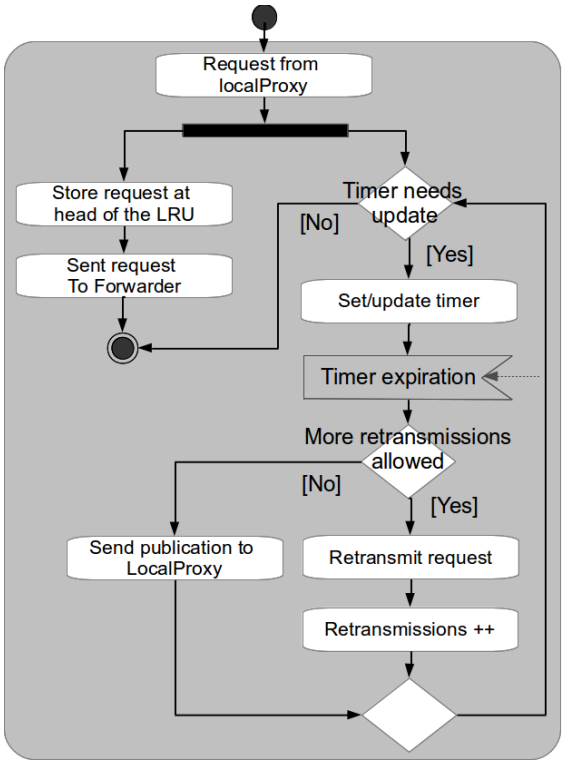

Figure 7: Activity Diagram: CR after the transmission of a request

A similar procedure takes place at the receiver's CR as demonstrated in the following figure. When a request is pushed from the network, then the CR checks if the associated AckID is stored; i.e., this implies a duplicate control message. If the AckID is found, then the CR concludes that the ACK message was lost, hence an ACK is created and retransmitted. Otherwise, the AckID is stored, the request is forwarded to the localProxy element and an ACK is sent back to the sender. In order to manage the amount of stored AckIDs the CR exploits an finite size LRU data structure that evicts the least recently used object based on memory (LRU is full) and temporal constraints (entry expired). The LRU is selected for it allows timer resets of retransmitted control packets. Each AckID is stored along with the reception time, in order to deduce the expiration time of the next-to-be-expired object.

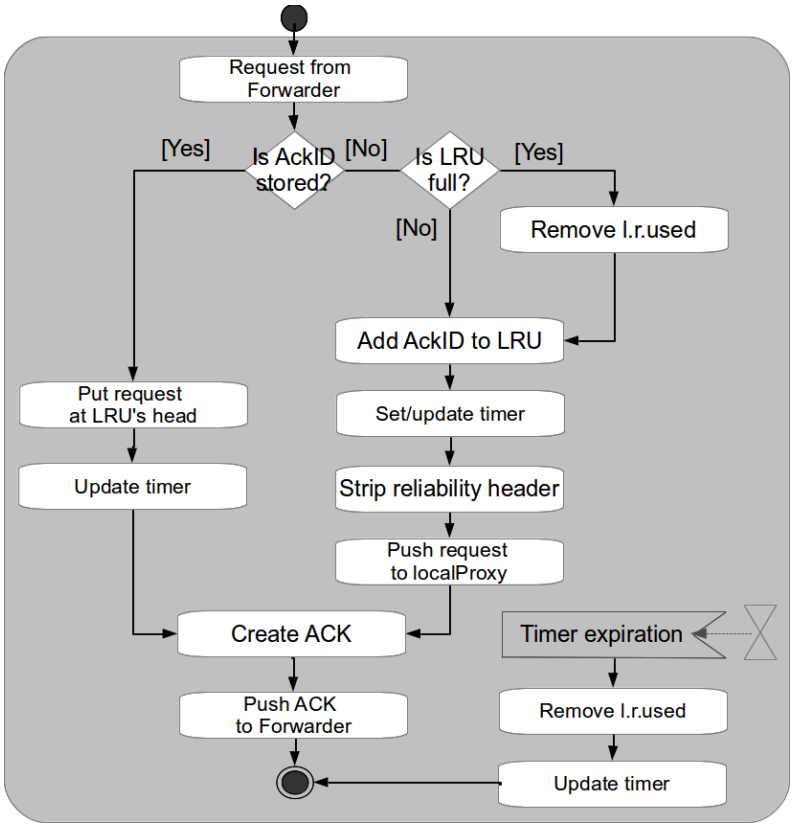

Figure 8: Activity Diagram: CR after the reception of a request