Authors:
Andrey Krokhin (Affinegy)
Way Vadhanasin (Microsoft)
Daniel Mihai (Microsoft)
Marcello Lioy (Qualcomm)

Version: 3
Revised: July 20, 2016

ASACORE-2946 Notes and Proposals

Most of the changes required to implement ASACORE-2946 and related tasks need to be done to the RemoteEndpoint class. The tasks are discussed in detail below, but due to code interdependence, they might have to be combined or further split into subtasks.

Memory Usage

Currently, there is no way to get the total memory used by all nodes connected to a single Routing Node. Each instance of RemoteEndpoint (which can be either a Routing Node or a Leaf Node) has its own message queue (txQueue) and threads queue (txWaitQueue). A static class variable, such as totalMsgCount, can be added to the RemoteEndpoint class, which should be incremented for every message received. Memory usage is comprised of two parts: stack memory, which can be calculated by totalMsgCount * sizeof(Message), and heap memory, which is consumed by pointers to other objects inside the Message class. In particular, `msgBuf, refMsgArgs`, and `handles` are dynamically allocated (there may be others). One possible way to track dynamic memory usage is to add it in constructors/destructors of these objects, similar to

Constructor

```
…
if (s_trackMemoryUsage) {
 QCC_VERIFY( AddAndFetch(&s_totalMemoryUsed, sizeof(*this) ) > 0 );
    }
}
```

Destructor

```
…
if (s_trackMemoryUsage) {
 QCC_VERIFY( SubtractAndFetch(&s_totalMemoryUsed, sizeof(*this) ) > 0
);
    }
}
```

(the above refers to total (global) memory, hence the “s_” prefix for static. For tracking individual memory usage of each node, we have to add memory outside of the constructor.) The objective would be to enforce memory limits (which could be infinite for no limit). Each RemoteEndpoint object would be able to return its memory usage, and take action (such as

disconnecting from the Router Node) if the limit is exceeded. Aleternatively, memory usage could be used for informational purposes. Marcello's proposal is to use a global “ranking” of Leaf Nodes by memory, and optionally drop the ones that consume the most.

Refactoring PushMessageRouter() and PushMessageLeaf()

The RemoteEnpoint class processes messages from both leaf nodes and routing nodes. The code in PushMessageRouter() and PushMessageLeaf() is largely duplicate, and can be refactored into a common function.

Message queues

Currently, there is inconsistent treatment of messages in routing nodes and leaf nodes. PushMessageLeaf() checks if MAX_TX_QUEUE_SIZE, hardcoded as 1, is exceeded, before deciding to add the message to txQueue. PushMessageRouter() checks whether the message is a “control message” and if it is, whether maxControlMessages is exceeded, before deciding to add the message to txQueue. If the message is not a “control message”, it checks whether MAX_DATA_MESSAGES, also hardcoded as 1, is exceeded, before deciding to add the message to txQueue.

The justification for treating control messages differently from data messages is, according to Marcello, to ensure that control messages continue to be delivered even in heavy traffic. However, the current implementation limits the control messages through queue size, while no such restriction exists for data messages.
TBD:
- Investigate message queue behavior for multicast messages with slow readers (node that dequeue messages very slowly)
- Investigate the feasibility of treating control and data messages in the same way
- Investigate the possibility of adding another limit for data messages

The mechanism for removing messages from queue should also be reworked. Currently, messages are only removed if (1) they have been successfully delivered, (2) their TTL has expired, in PushLeafNode() and PushRouterNode():

```
/* Remove a queue entry whose TTLs is expired.
 * Only threads that are the head of the txWaitqueue will purge this deque
 * and enqueue new messages to the txQueue.
 * This is to ensure that the original order of calling of PushMessage
 * is preserved.
 */
    uint32_t maxWait = Event::WAIT_FOREVER;
    if (internal->txWaitQueue.back() == thread) {
        deque<Message>::iterator it = internal->txQueue.begin();
        while (it != internal->txQueue.end()) {
            uint32_t expMs;
            if ((*it)->IsExpired(&expMs)) {
                internal->txQueue.erase(it);
                break;
```

```
        } else {
            ++it;
            if (maxWait == Event::WAIT_FOREVER) {
                maxWait = expMs;
            } else {
                maxWait = (std::min)(maxWait, expMs);
            }
        }
    }
```

Proposed change: When a new message gets added to the queue past a certain queue size, or when RN's global memory usage becomes close to the limit, we should disconnect the endpoint, and destroy its queue. We could also implement a mechanism for deleting messages from queue (and logging an error) if delivery was attempted but failed after a certain time.

Timeouts

There are three types of cases in which an endpoint is closed:

- During a WriteCallback(), if encountering a exceeding a "thread stopping" or "bus stopping" condition.
- During ReadCallback(), if maxIdleProbes is exceeded. maxIdleProbes is defined as a constant with value of 1, in TCPTransport.cc, so this is the condition most frequently reached with multiple LNs connected to a RN.
- During PushMessageRouter(), if maxControlMessages is exceeded. That limit is defined in RemoteEndpoint::Start() as `maxControlMessages = sendTimeout * MAX_CONTROL_MSGS_PER_SECOND.`

Proposed Changes:
We should consider increasing the maxIdleProbes limit, so that other limits become meaningful.
Initializing maxControlMessages (to 30) in RemoteEndpoint constructor, then setting it to a different value in Start() is confusing and unnecessary. Furthermore, the comment for sendTimeout class variable says "Send timeout for this endpoint i.e. time after which the Routing node must disconnect the remote node if the remote node has not read a message from the link in the situation that the send buffer on this end and receive buffer on the remote end are full". However, this limit appears not to be directly enforced (there is no disconnect happening due to a sendTimeout). Rather, sendTimeout is used indirectly in computing maxControlMessages. We propose to either change the wording in the comment, or make maxControlMessages independent of sendTimeout.

In theory, maxControlMessages might become redundant if memory limits are implemented (see "Memory"). However, they need to preserved for now, due to the issue of control messages vs all messages (see "Message Queues"), and the optionality of memory limits (see "Memory").

Sender vs Receiver

One of the disconnect conditions described happens when maxControlMessages is exceeded in a particular branch of PushMessageRouter():

```
    if (IsControlMessage(msg)) {
        if (internal->numControlMessages < internal->maxControlMessages) {
            internal->txQueue.push_front(msg);
            internal->numControlMessages++;
            if (wasEmpty) {
                internal-
>bus.GetInternal().GetIODispatch().EnableWriteCallbackNow(internal->stream);
            }
        } else {
            QCC_LogError(ER_BUS_ENDPOINT_CLOSING, ("Endpoint Tx failed (%s)",
GetUniqueName().c_str()));
            internal->stream->Abort();
            internal->bus.GetInternal().GetIODispatch().StopStream(internal-
>stream);
            SetState(Internal::EXIT_WAIT);
            Invalidate();
            status = ER_BUS_ENDPOINT_CLOSING;
        }
```

Here, the routing node (receiver) gets disconnected, even though the leaf node (sender) was likely responsible for message overflow. In PushMessageLeaf(), there is no similar disconnect mechanism.

We need to keep track of “noisy apps” that generate excessive message volume, and disconnect them. Different metrics or limits could be applied here: the absolute number of messages in txQueue, whether a node uses more memory than other nodes (as explained in “Memory Usage”), or the frequency of messages (this can be implemented as a function that goes through each message in txQueue, calculates its timestamp difference with the next message, and computes the average).

There also needs to be a mechanism for making the Routing Node disconnect a Leaf Node that violates one of the limits (memory, queue size, message frequency, etc.). If a new message arrives to the Routing Node, we could do one of:

- If that Message arrived from Leaf Node, the Routing Node should be able to insert this Message into a new list attached to Leaf Node, or

- The Message object could own a reference to the Leaf Node

Currently, we do not have a way to force a disconnect of a leaf node from the routing node.

TBD: How to preserve the existing solution for the slow reader (RN) problem, while adding a solution to the aggressive sender (LNs) problem.