UDP Transport and Router Architecture

1. INTRODUCTION

Introducing the UDP Transport Enhancements and Router Optimizations architecture

1.1. VERSION

Version	Description	Author	Date
0.01	Capture High-Level Architecture	Craig Dowell	February 28, 2014

2. ARCHITECTURE

2.1. PRELIMINARIES

There is an existing architecture. It looks like this

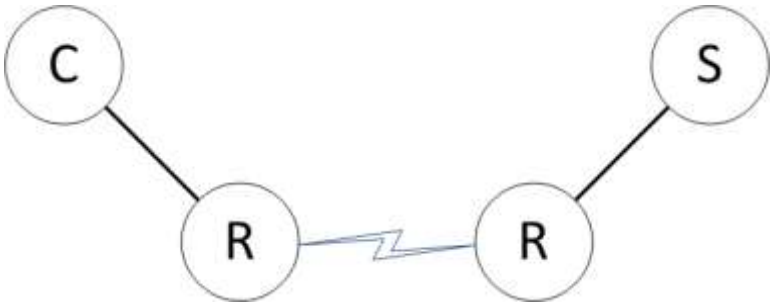

Figure 1: Basic AllJoyn Architecture

When you use Bundled Routing Nodes it looks like this

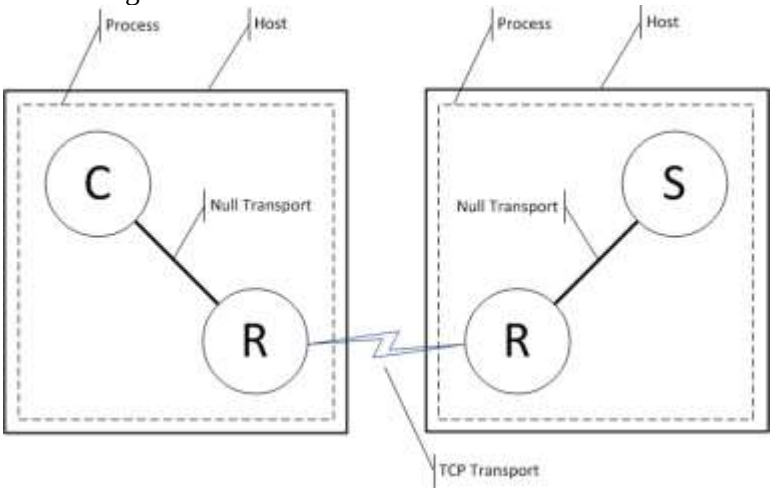

Figure 2: Bundled Routing Node Case

When you have a Bundled Client and Routing Node and a preinstalled Routing Node running on Linux with a service in another process it looks like this

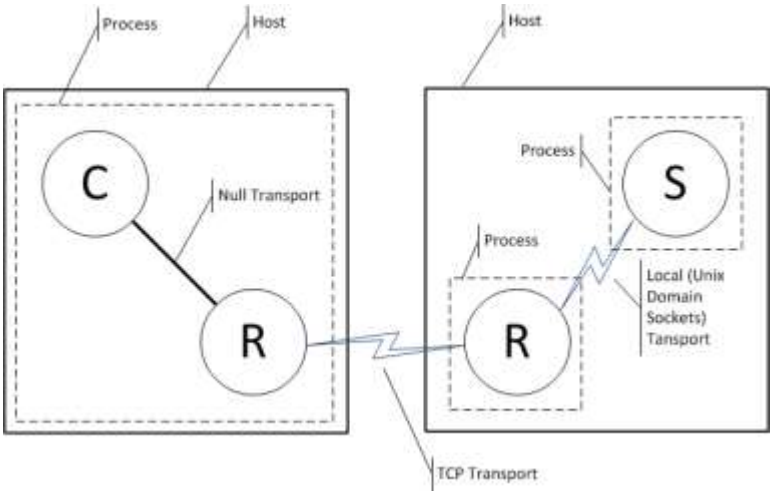

Figure 3: Linux Pre-installed Routing Node Case

When you have a bundled Routing Node client and a preinstalled Routing Node running on Windows with a service in another process it looks like this

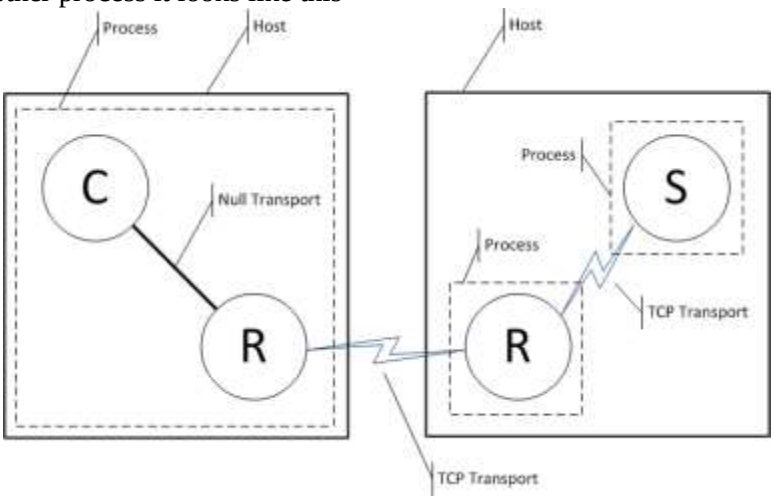

Figure 4: Windows Pre-Installed Routing Node Case

If you throw Thin Library into the mix it looks like this

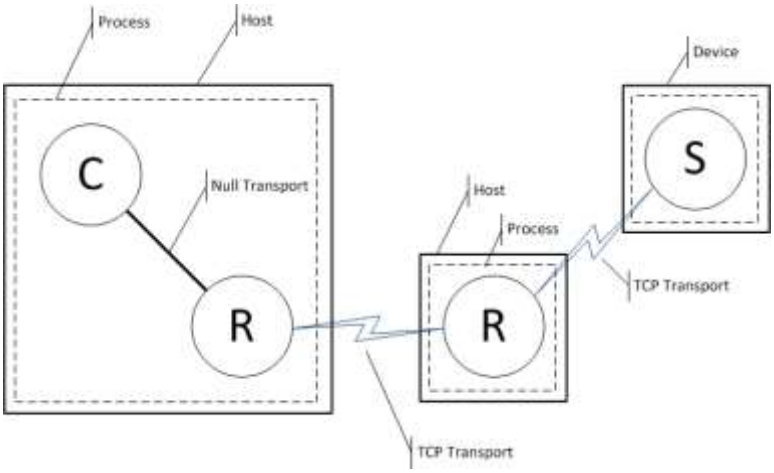

Figure 5: Thin Library to Routing Node Case

We can't go around arbitrarily breaking any of this stuff.

2.2. WHAT TO DO

The most fundamental requirement is to allow substitution of UDP for TCP when appropriate. The corollary is that some of our transports are perfectly fine when viewed in context. For example, the Null Transport is our most efficient transport since it is basically a method call transport. We should not design a system that would force us to substitute out that transport. Also, the Unix Domain Socket-based Local Transport is a perfectly fine transport for inter-process communication in Linux. There is no reason to force a change that would dictate a traversal of a networking stack unnecessarily. What we really want to do is to be able to switch out UDP for TCP in the Routing Node to Routing Node (bus-to-bus) case and in the Thin Library to Routing Node connection case and possibly to the Windows-based Standard Library to Preinstalled Routing Node (local) case.

The really illustrative case is the case where we have a bundled Routing Node on one side and a Thin Library and associated Routing Node on the other.

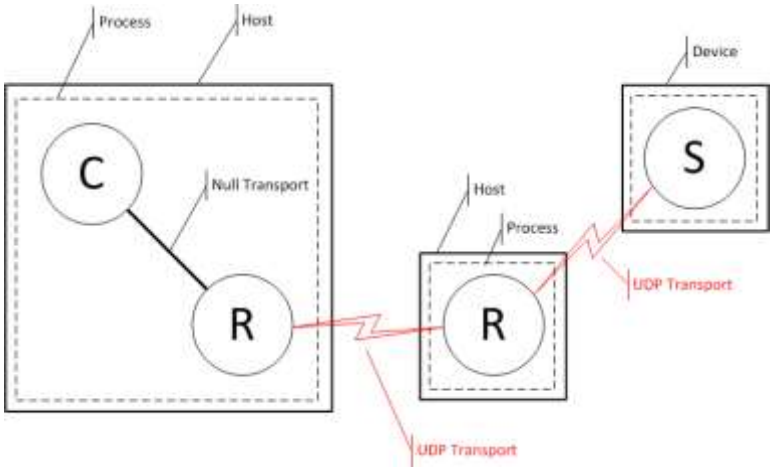

Figure 6: Replacing the TCP Transports

Sounds simple, eh?

3. APPLYING THE END-TO-END PRINCIPLE

The end-to-end principle is a classic design principle in computer networking. The end-to-end principle states that application-specific functions should reside in the end hosts of a network rather than in intermediary nodes. In AllJoyn terminology, it is easier and more tractable to obtain reliability by introducing mechanisms at the bus attachments rather than in the routing nodes. Several implications of applying this principle follow immediately if one accepts the argument.

- Because it is easier and more tractable to add a reliability layer at the endpoints of the communication link (in the client labeled "C" and the service labeled "S" in Figure 6), the reliability layer does not belong in a Routing Node, it should be associated with the LocalEndpoint of the source and destination BusAttachments.

- Since Routing Nodes have a LocalEndpoint corresponding to an AllJoynObj, Routing Nodes supporting UDP do need a reliability layer, just not where one might expect.

- If we do not want to arbitrarily replace Null Transports and Local Transports, a new kind of message will need to flow through the system – Message Fragments. This corresponds to a new messageType in the AJ message header.

- Since existing ClientRouter and DaemonRouter modules only route complete messages, we need to provide an equivalent FragmentRouter that deals with correctly routing message fragments. Routing using a fragment router is basically just determining if a fragment is destined for the local endpoint and if not mapping a unique ID to an IP address and sending it. Simple.

- The UDP Transport in a Routing Node is just an empty shell to hold the single UDP endpoint resource and to add routes to the fragment routing table.

- Since the reliability layer is pushed to the endpoint, careful attention must be paid to keep that layer as simple as possible and to be very configurable with respect to resource consumption. This is most important on Thin Library devices which will have limited capacity for addiontal instructions and data.

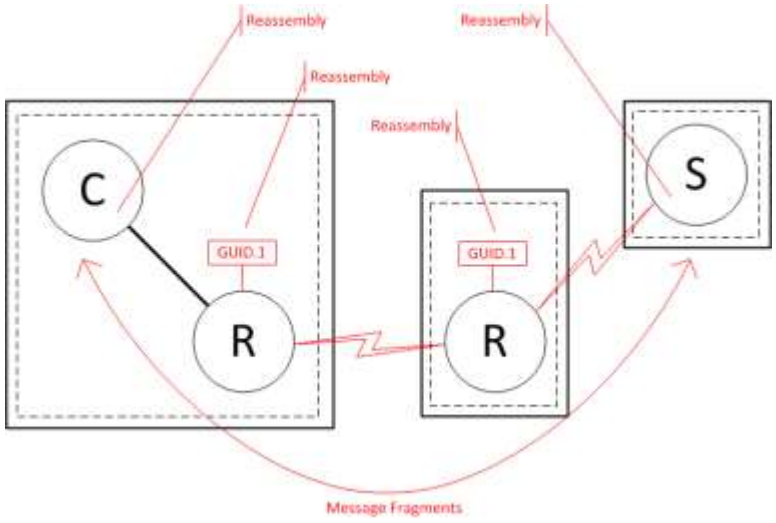

Figure 7: Routing Message Fragments

Figure 7 shows how this might work.

If the Thin Library on the right hand side needs to communicate with the AllJoynObj on its routing node, it will map local messages to message fragments and pass them onto its Thin ClientRouter which will send the message fragments to the Routing Node as UDP packets. The Routing Node will receive the UDP packets in its UDP Transport. The UDP Transport will call the Fragment Router directly, which will identify the destination as the local endpoint. The incoming message fragment will be queued directly into the RX Queue of the LocalEndpoint. The receive handler of the local endpoint will wake up and pull the message fragment. Since the message is a fragment, it will pass this to its reassembler. When a complete message is assembled, the reassemble will hand it back to the local endpoint which will process the inbound message as usual.

If the Thin Library Service on the right hand side needs to communicate with the Standard Library Client on the left side, it will map local messages to message fragments using its UDP "transport" and pass them onto its Thin ClientRouter. The client router knows nothing about the destination unique name (and has no space to store a routing table anyway) just as in the default gateway case for IP packets, it sends the resulting message fragments on to its associated Routing Node. The message fragments will be received by the Routing Node UDP "transport" but will ***not*** be reassembled. Instead, the message fragments will be routed as if it were an IP router doing the work. The routing table in the Routing Node will say indicate that to in order to get message fragments to the unique ID of the destination, the message fragments need to be ***forwarded*** to the remote Routing Node. The Fragment Router on the local Routing Node will simply receive a UDP packet, consult the fragment routing table and forward the UDP packet. Note that if there is room on the source to hold a fragment routing table, it would be possible to send the packet directly to the destination Routing Node. When the destination Routing Node receives the message fragment, it consults its routing table and determines that the destination is associated with the local host and so forwards the message fragment to the appropriate endpoint. In this case, the endpoint is a Null Transport endpoint. The message fragment is received by the destination and since it is a message fragment is passed to a reassembler. When a complete message has been received by the reassembler, the newly complete message is passed on to the Client as usual.

4. THINNER, LIGHTER, FASTER

Notice that the Thin Library will include a reassembler. One of the benefits of applying the end-to-end principle is that the Thin Library can have an important voice in the amount of resources it consumes in the process of reassembly. If the Thin Library has only enough buffer space for one UDP receive buffer, then it can present a window size of one and therefore the code can be simplified tremendously since there is no out-of-order problem to solve with a window size of one. On the sending side, one buffer is needed to hold a message fragment until it is ACKed, and on the receiving side one buffer is required to receive data until the correct sequence is received.

The real challenge will be to introduce a protocol that allows for ultra-lightweight and ultra-simple implementations on the Thin Library side, but which also allows for high-performance implementations on the Standard Client side.

5. DBUS, CONNECTIONS, LINK LOST

We have an interesting opportunity to rethink the bus-to-bus connection mechanisms. Currently, our Transports perform a DBus-style connect whenever a bus-to-bus connection is made for any reason. This involves a SASL exchange and a BusHello exchange which exchanges all of the unique names known in the system. This results in floods of NameOwnerChanged signals and possibly very large amounts of work which is ultimately typically not necessary at all.

Some local presence and absence detection is required for upper level protocols to work. For example, the Thin Library relies on NameOwnerChanged to allocate and free security-related resources. There seems to be no reason that we cannot ignore the connection and ExchangeNames process and construct the appropriate times to generate local-only versions of the signals when the UDP Transport is chosen.

If the UDP Transport always happens to be chosen as the bus-to-bus transport, this has the possibility of solving some other architectural problems.

Basically, a FoundAdvertisedName would drive a JoinSesssion request as in the current AllJoyn API. Instead of resulting in a new TCP connection being formed in the TCP Transport (which causes SASL, BusHello and ExchangeNames); a HeyHowdy (a very informal BusHello) message could be sent by the system. This message would establish routes and cause a NameOwnerChanged (Null, unique name) to be generated at the ultimate destination. A LeaveSession would similarly be used to clear routing tables and generate the corresponding NameOwnerChanged (unique name, NULL) at the other side.

6. MULTICAST

Although explicitly not a requirement for the current project, using UDP admits the possibility of actual and not simulated point-to-multipoint addressing. Reliable multicast protocols do exist which would make the performance of large-scale delivery of signals or no-return method calls much faster.