ALLSEEN ALLIANCE

AllJoyn over MQTT

Sheshambika Venkateshwaran
Staff Engineer,

Qualcomm Connected Experiences, Inc.

Agenda

1. Rationale
2. MQTT basics
3. Overview
4. Implementation
5. Test results

Rationale

AllJoyn over MQTT

- Sets out to answer this question:
  - *“Can an MQTT broker can replace an AllJoyn routing node?”*
- Approach
  - Modify thin client code to communicate directly via MQTT messages
  - MQTT broker to take care of message routing
- Requirements
  - Transparent to application code – existing apps should still work
  - Must support About-based discovery
  - Must support sessions (including multi-point sessions)
  - Must support session-less signals

Why run AllJoyn over MQTT

- MQTT designed for high scalability
  - Benchmarks at 1000x connections supported by AllJoyn RN
  - Some implementations claim > 200K simultaneous connections
  - 1,000,000 messages/second
- Multiple broker options
  - Hosted
  - SaaS solution
  - Open source
- Dozens of client libraries
  - Many open source
  - Mosquitto, RSMB, WebSphere MQ, HiveMQ, etc.

MQTT basics

MQTT basics

- MQTT clients communicate with an MQTT broker
  - Broker handles all communication between clients
  - Brokers can communicate with other brokers
- Protocol is message based publish/subscribe
  - MQTT does not define payload format – payload is a binary blob
  - Messages published to broker are tagged with a *topic*
  - A topic is slash-delimited hierarchical string
  - Messages can be made persistent by setting the “retain” flag
- A client can subscribe to a topic with an exact match or a wildcard
  - Client sends a subscription request to the broker
  - Broker from then on forwards all messages that match the subscription

MQTT basics continued

- MQTT clients specify a unique client-id when connecting to broker
  - Clients can optionally disconnect while maintaining existing subscriptions
- Clients can define a *will* (c.f. last-will-and-testament)
  - A *will* is a message published by the broker on behalf of the client if a client disconnects unexpectedly
  - A client can only specify a *will* once at the time it connects.
- A published message is canceled by publishing NULL to same topic

AllJoyn over MQTT Overview

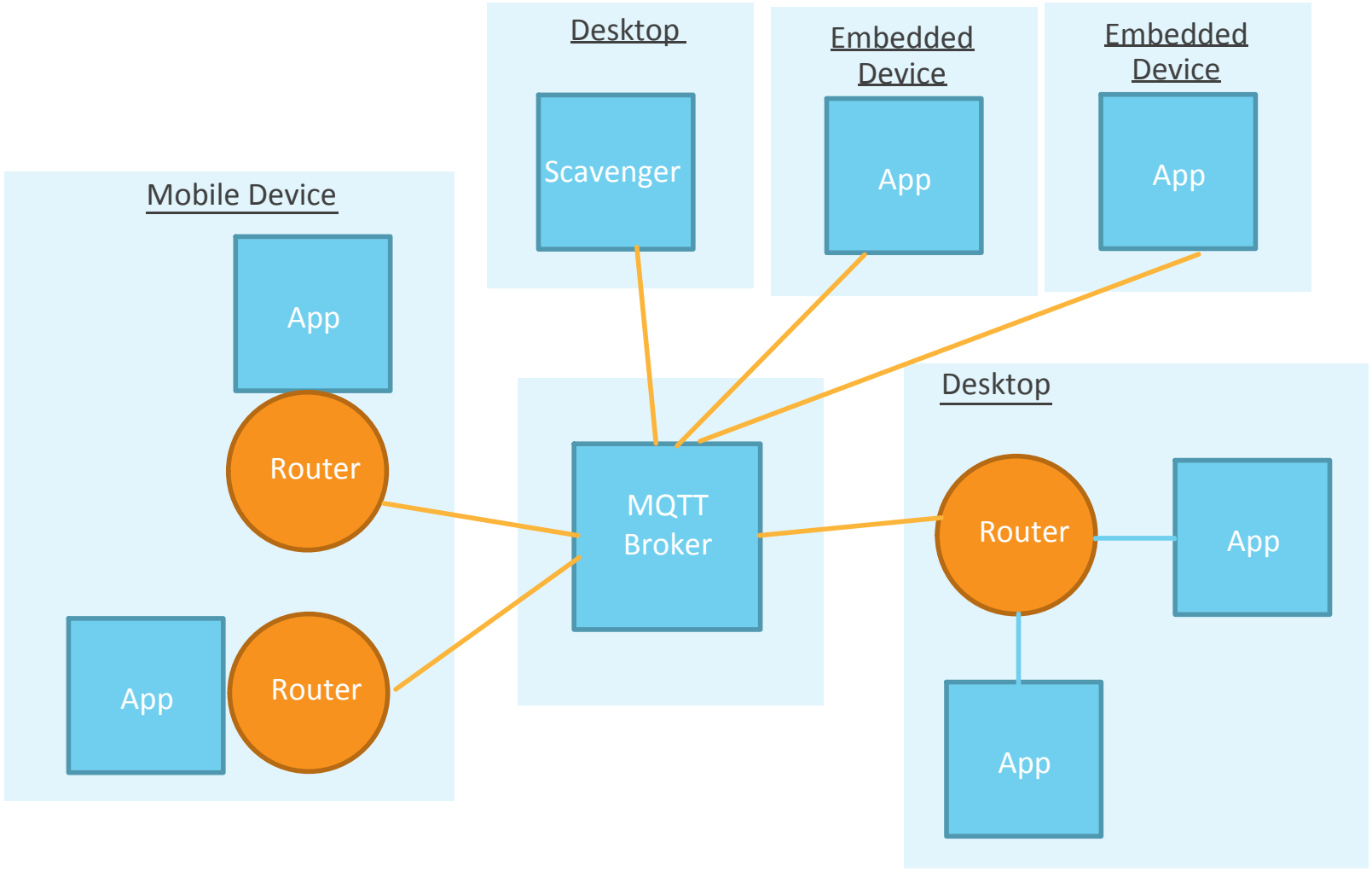
General Approach

- AllJoyn Thin clients connect directly to the MQTT broker
  - Code shim sends AllJoyn messages by publishing to topics
  - Code shim receives AllJoyn messages by subscribing to topics
- AllJoyn Standard clients connect to an AllJoyn routing node which in turn connects to the broker
  - Similar shim code layer as for thin clients
- Scavenger connects to the MQTT broker
  - A special application responsible for session-less signal cleanup
  - Can run on the same machine as broker or different machine

What code has been contributed

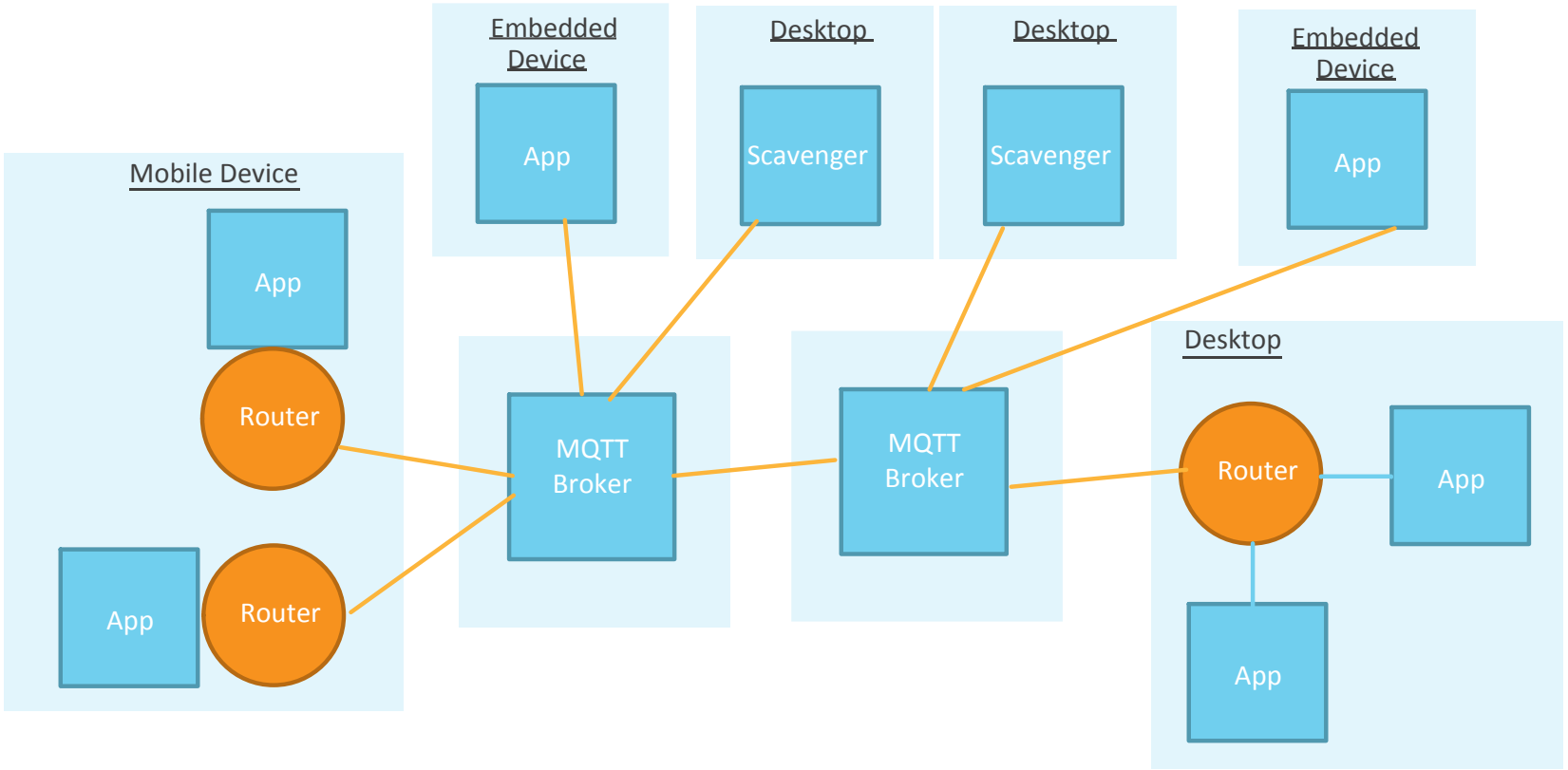
- Thin client and router experimental implementation contributed
- Builds on Linux, Windows and Darwin
- Links to Mosquitto open source (BSD license) client library
- Transparent to application code – existing applications still work
- Maintains API compatibility
  - Existing sample applications still work
  - AllJoyn.js still works
- Connection to broker via configured URL
  - Unlike Alljoyn RN which is discoverable via MDNS
- No client to broker authentication in current code
  - AllJoyn peer-to-peer security still applies

Topology – single broker



Topology – multiple brokers

- Can configure MQTT brokers to forward messages sent over certain topics to other brokers



Supported features

- Supports direct messaging
  - All nodes subscribe to their unique name topic with a pre-defined format to which messages can be published by other nodes
- Supports session-less signals
  - Session-less emitters publish session-less signals to a session-less topic with a predefined format with the retain flag set to true
  - Session-less receivers subscribe to a wildcarded session-less signal topic
  - Broker retains session-less signals and forwards them as applicable
- Supports about-based discovery
  - Well known name based discovery is not supported
  - Uses session-less signal delivery mechanism for the org.alljoyn.About interface and Announce signal

Supported features contd.

- Supports sessions
  - Session joiner discovers the session host using About discovery
  - Session joiner sends JoinSession/AttachSession message to the session host's topic with a predefined format
  - The session host thin client library/router sends an AcceptSession to the app
  - Depending on the app response, the thin client library/router sends a successful/unsuccessful response back to the session joiner using the consumer topic
  - Both ends subscribe to a predefined session topic and to each other's presence
  - Supports session cast messages

AllJoyn over MQTT Implementation

Topic strings

- Presence: <scope_guid>/presence/<guid>/<num>
  - scope_guid: This is used to partition the huge space of clients. Clients in the same scope can communicate with each other
  - If uniquename is :X.1, guid is :X and num is 1
  - num = 0 for a thin leaf node, 1 for a routing node, 2+ for a standard client leaf node
- Direct destination: <scope_guid>/<guid>/<num>
- Sessioncast: <scope_guid>/<session_id>/<host_guid>/<host_num>
  - host_guid and host_num are part of the session host's unique name
- Sessionless Topic: <scope_guid>/<prod_guid>/<prod_num>/<intf>/<member>

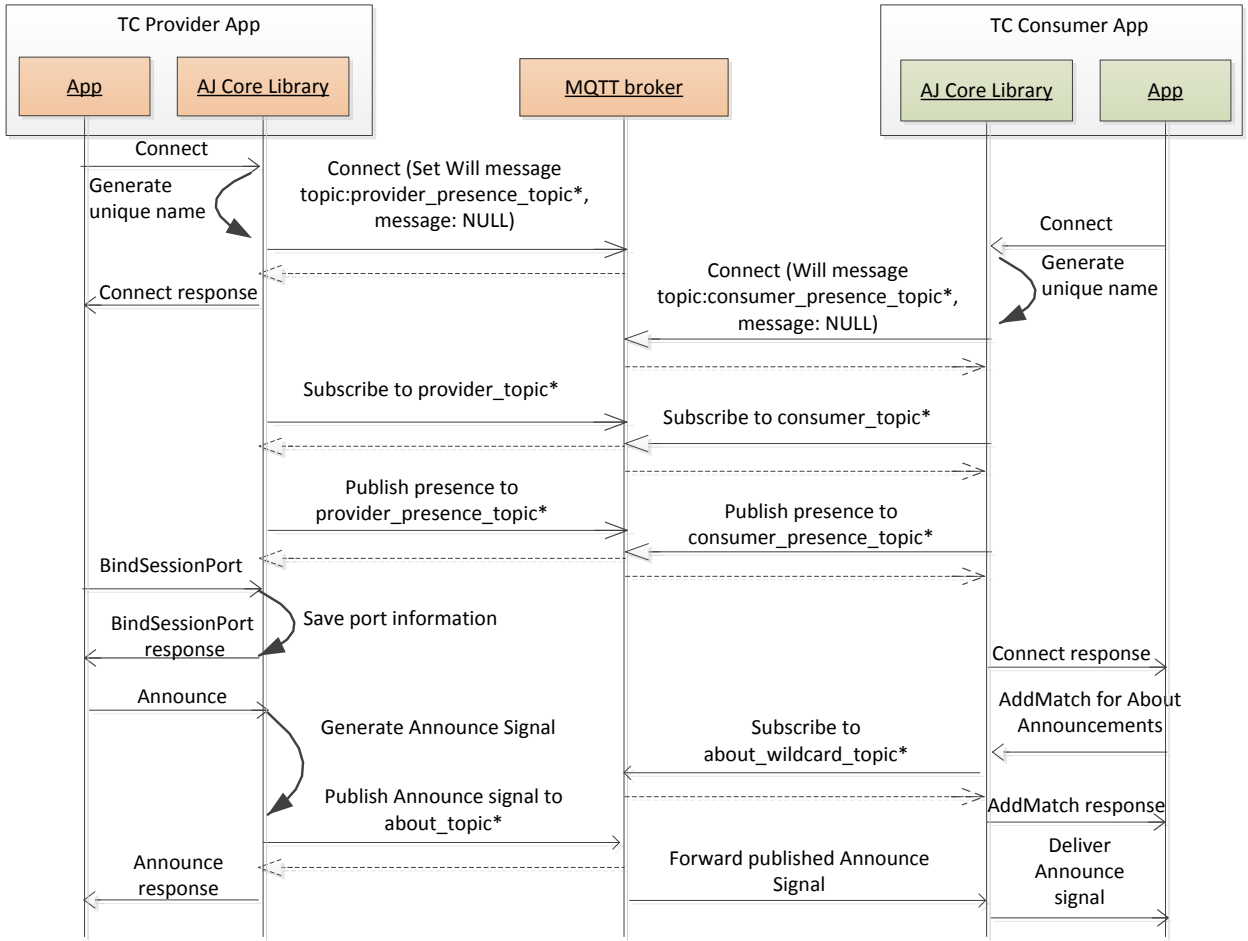
Last will

- Presence message for the routing/thin leaf node with a NULL payload
- Used by other thin clients/routing nodes to detect that a node has left the network

Sessionless signals

- Sessionless Topic: <scope_guid>/<prod_guid>/<prod_num>/<intf>/<member>
  - prod_guid and prod_num are part of the sessionless emitter's unique name
  - intf: Interface of the sessionless signal
  - member: Member of the sessionless signal
- Retain flag is set to true
- Sessionless emitter publishes a message to the sessionless topic
- Sessionless receiver subscribes to the sessionless topic
- About discovery uses the sessionless signal with interface org.alljoyn.About and member Announce

About discovery – Thin client

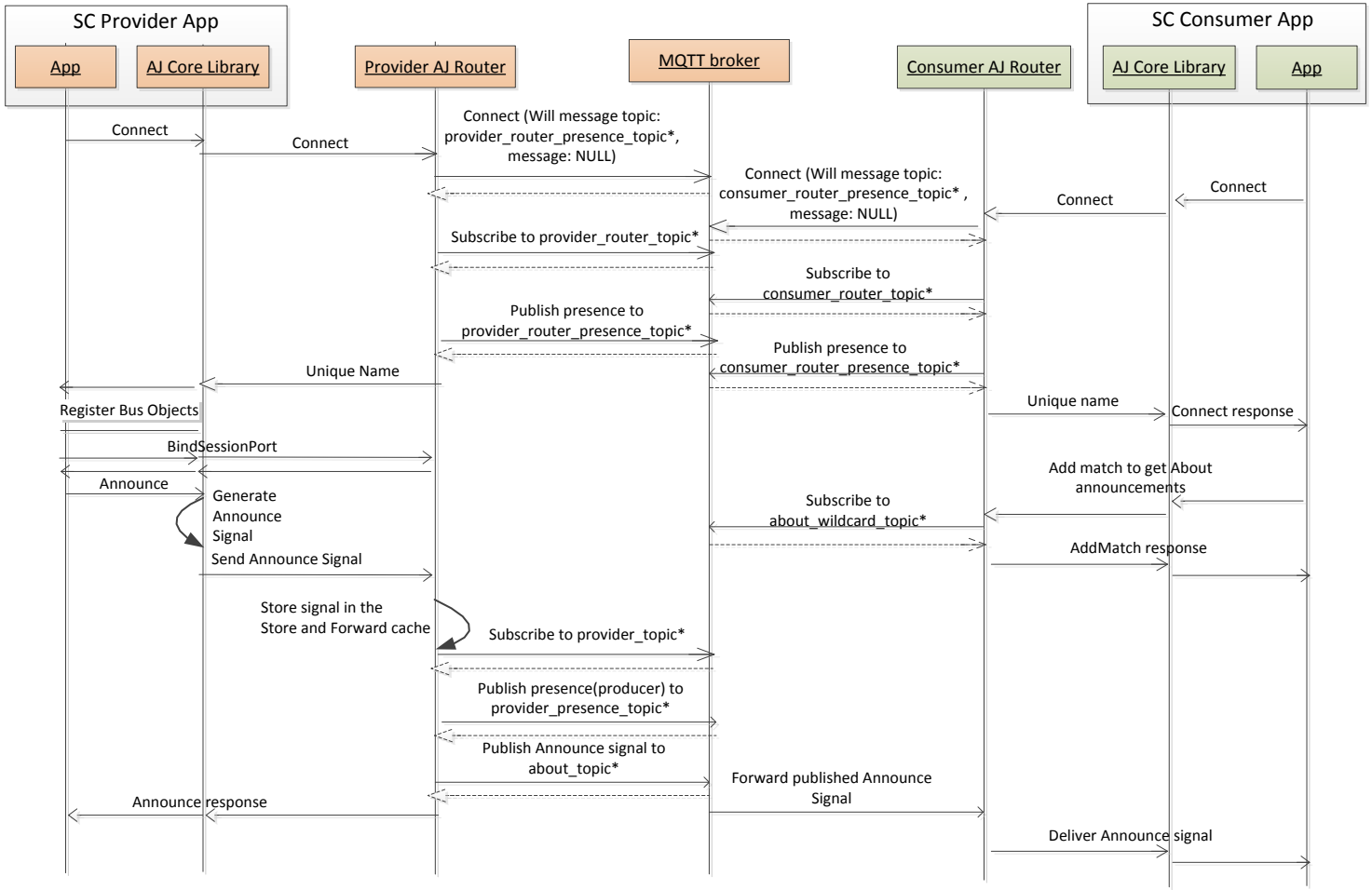


provider_presence_topic: scope_guid/presence/provider_guid/0
consumer_presence_topic: scope_guid/presence/consumer_guid/0
provider_topic: scope_guid/provider_guid/0
consumer_topic: scope_guid/consumer_guid/0

about_topic: scope_guid/guid/provider_identifier/org.alljoyn.About/Announce
about_wildcard_topic: scope_guid/+/+/org.alljoyn.About/Announce
Provider unique name: provider_guid.0
Consumer unique name: consumer_guid.0
Note: The 0 denotes that this is a thin leaf node

----------> Connect Ack/Subscribe Ack/PublishAck

About discovery – Standard client



*provider_router_presence_topic: scope_guid/presence/provider_guid/1
consumer_router_presence_topic: scope_guid/presence/consumer_guid/1
provider_presence_topic: scope_guid/presence/provider_guid/2
provider_router_topic: scope_guid/provider_guid/1
provider_topic: scope_guid/provider_guid/2

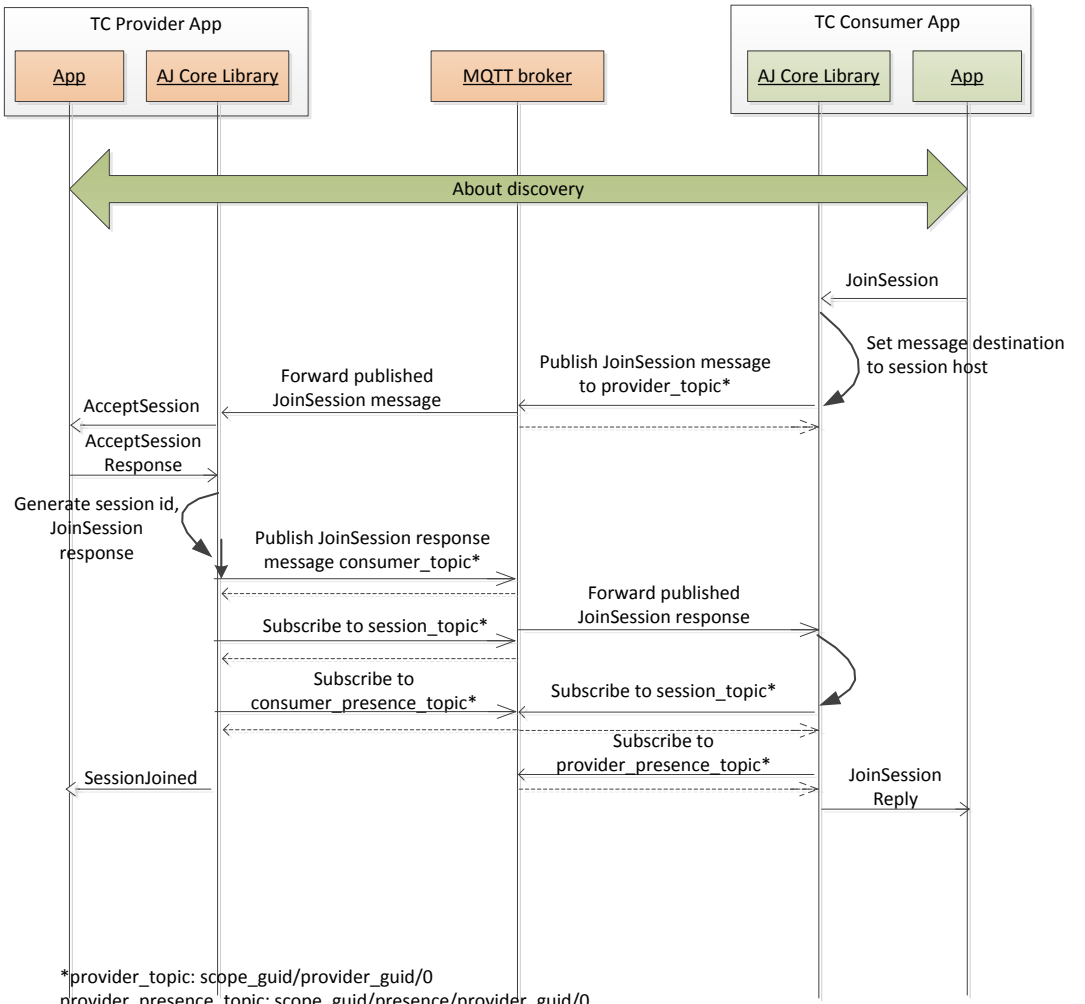
consumer_router_topic: scope_guid/consumer_guid/1
consumer_topic: scope_guid/consumer_guid/2
about_topic: scope_guid/guid/provider_identifier/org.alljoyn.About/Announce
about_wildcard_topic: scope_guid/+/+/org.alljoyn.About/Announce
Provider unique name: provider_guid.2
Consumer unique name: consumer_guid.2

----------> Connect Ack/Subscribe Ack/PublishAck

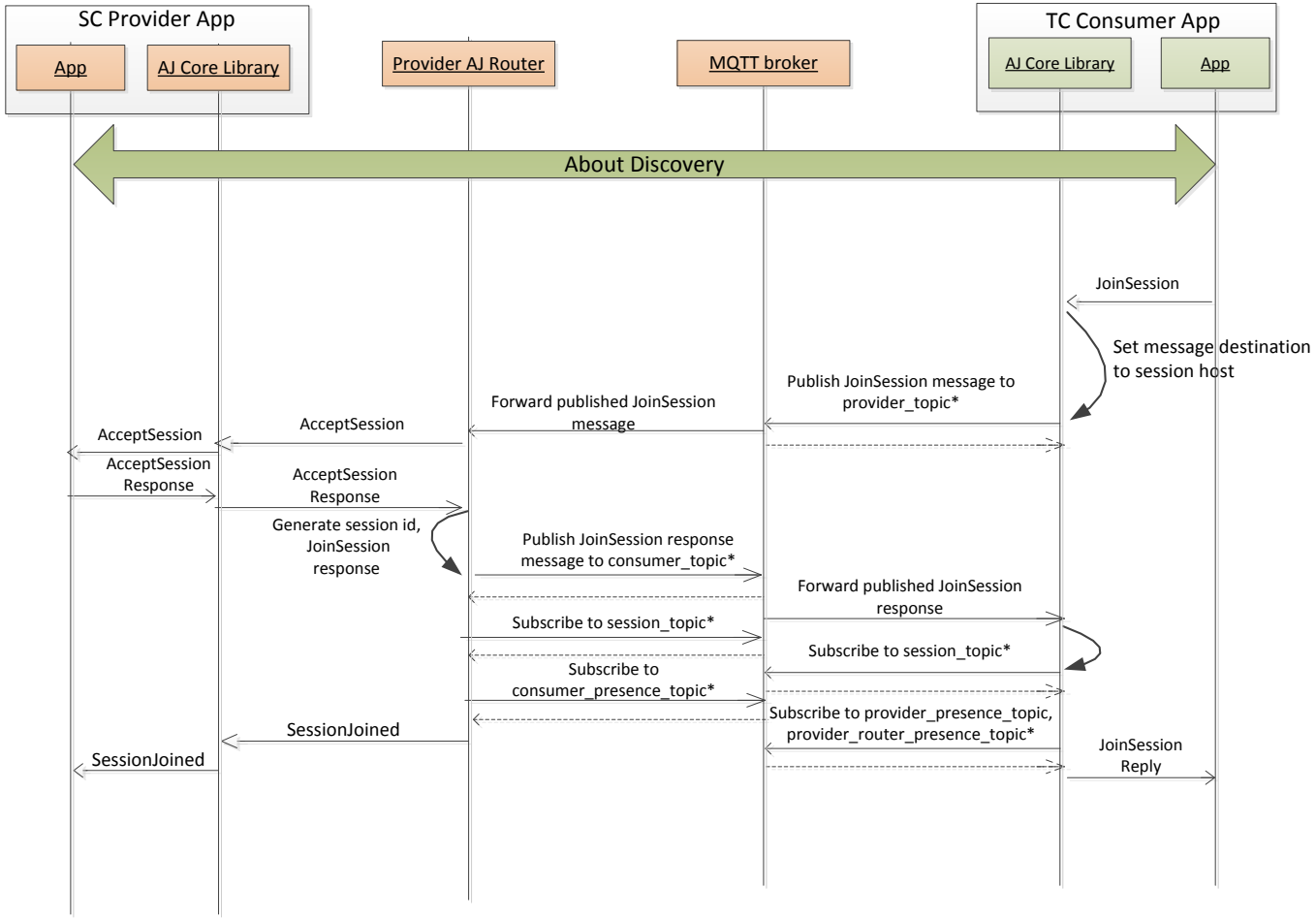
Point to point Session Establishment

- The consumer thin client/consumer router publishes a JoinSession or AttachSession message directly to the provider thin client/provider router
  - Standard client consumer router to standard client producer router: AttachSession message
  - Otherwise, JoinSession message
- The provider app receives an AcceptSession call
- Depending on the AcceptSession reply, the provider thin client/provider router publishes a JoinSession or AttachSession reply to the client thin client/provider router
- Both sides subscribe to the sessioncast topic and session based messages are exchanged using the same: <scope_guid>/<session_id>/<host_guid>/<host_num>

Session: TC client to TC service



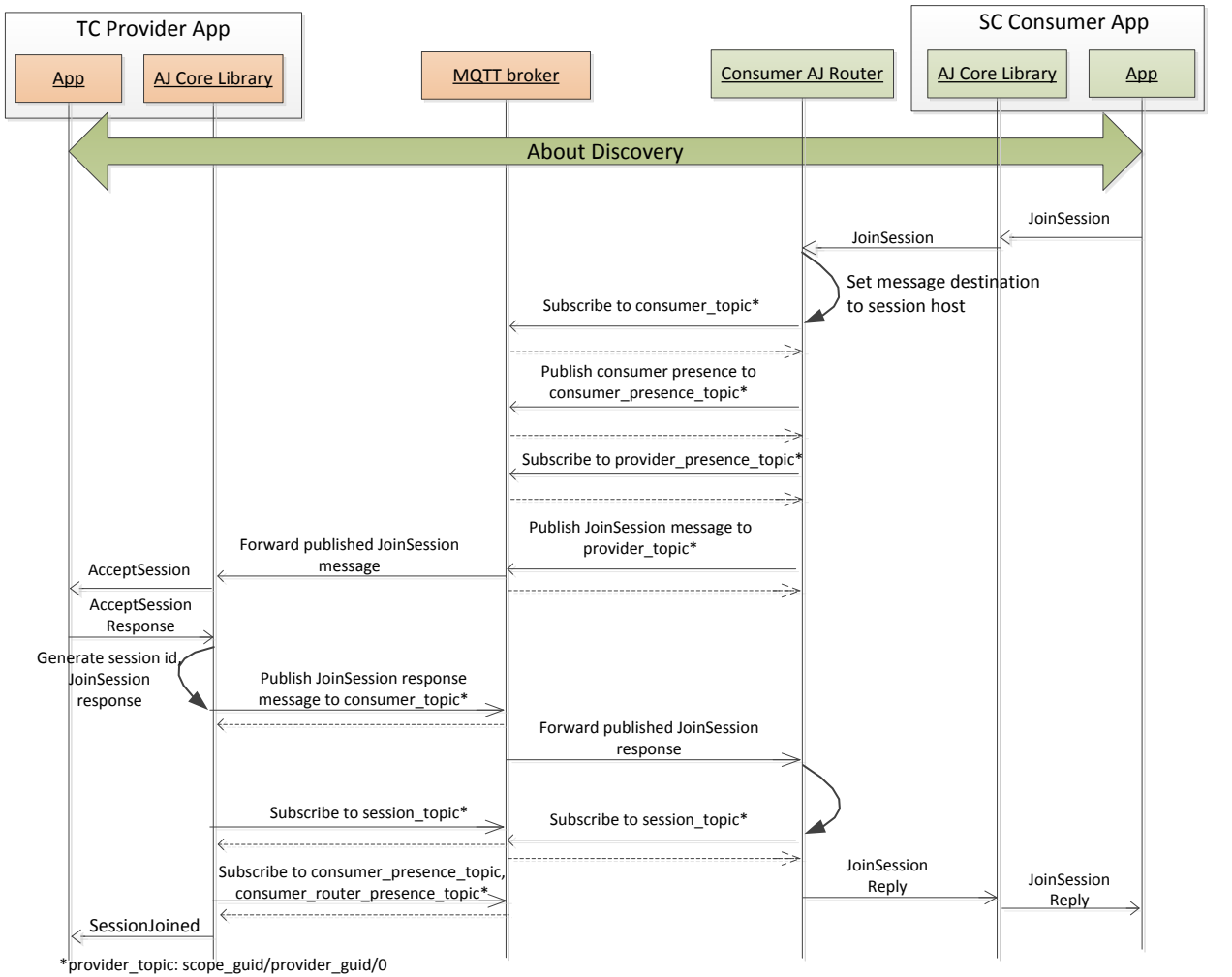
Session: TC client to SC service



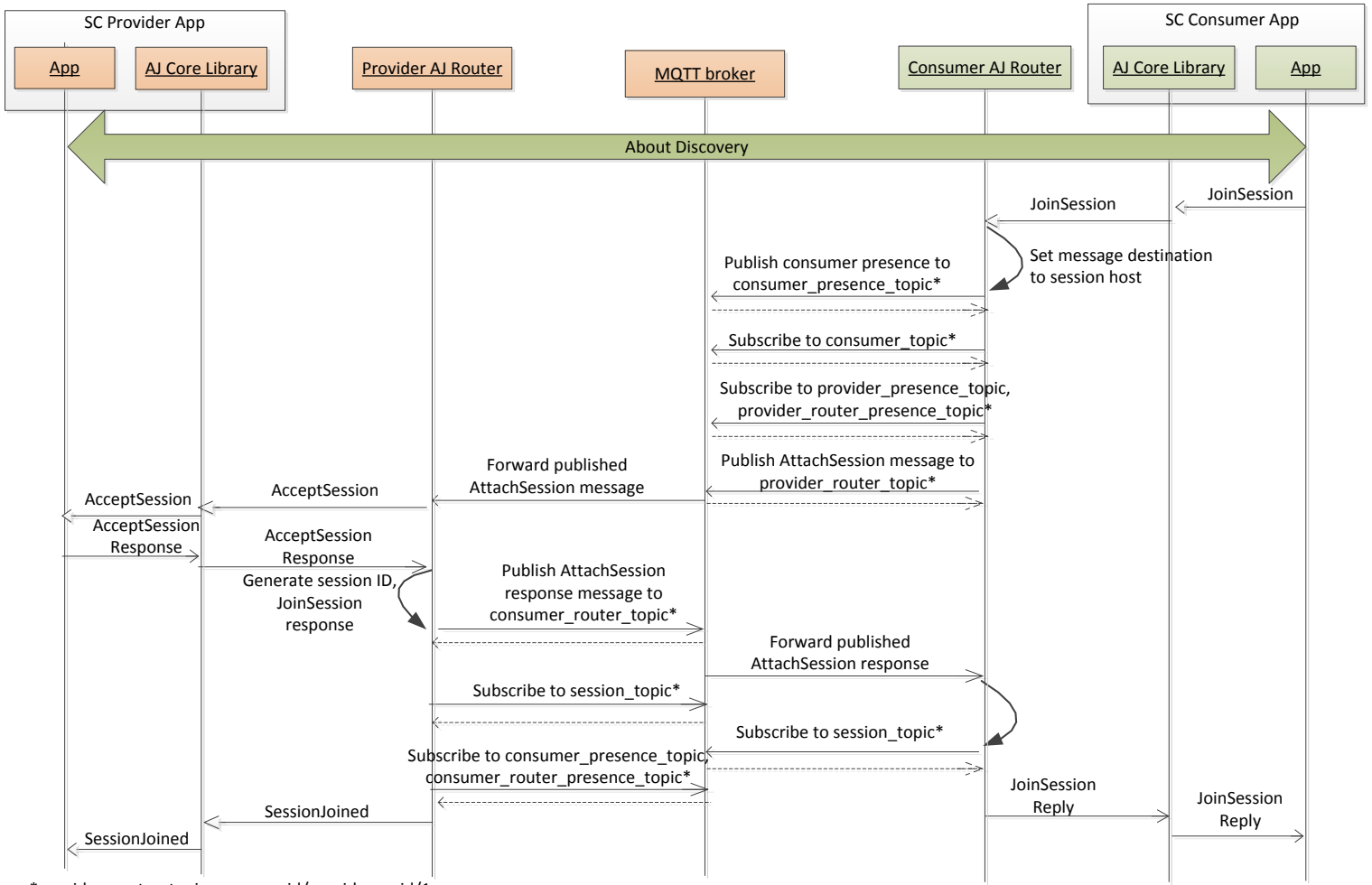
*provider_topic: scope_guid/provider_guid/2
provider_presence_topic: scope_guid/presence/provider_guid/2
provider_router_presence_topic: scope_guid/presence/provider_guid/1
consumer_topic: scope_guid/consumer_guid/0
consumer_presence_topic: scope_guid/presence/consumer_guid/0
session_topic: scope_guid/SessionId/producer_guid/2
where the producer's unique name is producer_guid.2 and consumer's unique name is consumer_guid.0

----------> Connect Ack/Subscribe Ack/PublishAck

Session: SC client to TC service



Session: SC client to SC service



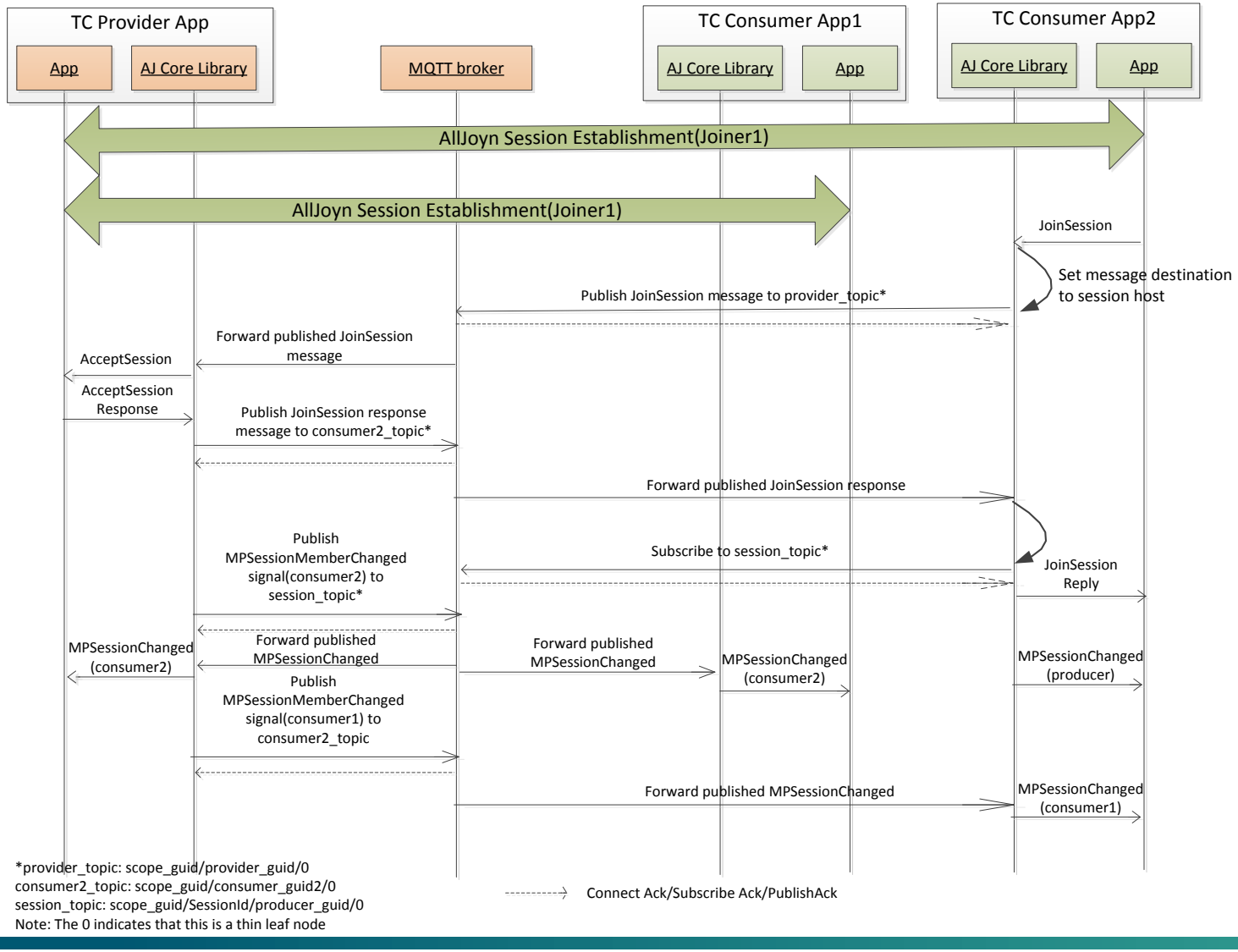
*provider_router_topic: scope_guid/provider_guid/1
consumer_router_topic: scope_guid/consumer_guid/1
consumer_topic: scope_guid/consumer_guid/2
consumer_presence_topic: scope_guid/presence/consumer_guid/2
session_topic: scope_guid/SessionId/producer_guid/2
where the producer's unique name is producer_guid.2 and consumer's unique name is consumer_guid.2

----------> Connect Ack/Subscribe Ack/PublishAck

Multi-point Session Establishment

- Uses a similar procedure to point-to-point session establishment. The provider app receives an AcceptSession call
- In addition, the provider thin client/provider router sends out an MPSessionChanged signal for the new joiner to the sessioncast topic
- As a part of catch-up, the provider thin client/provider router sends out a series of MPSessionChanged signals with the current members of the session to the new joiner

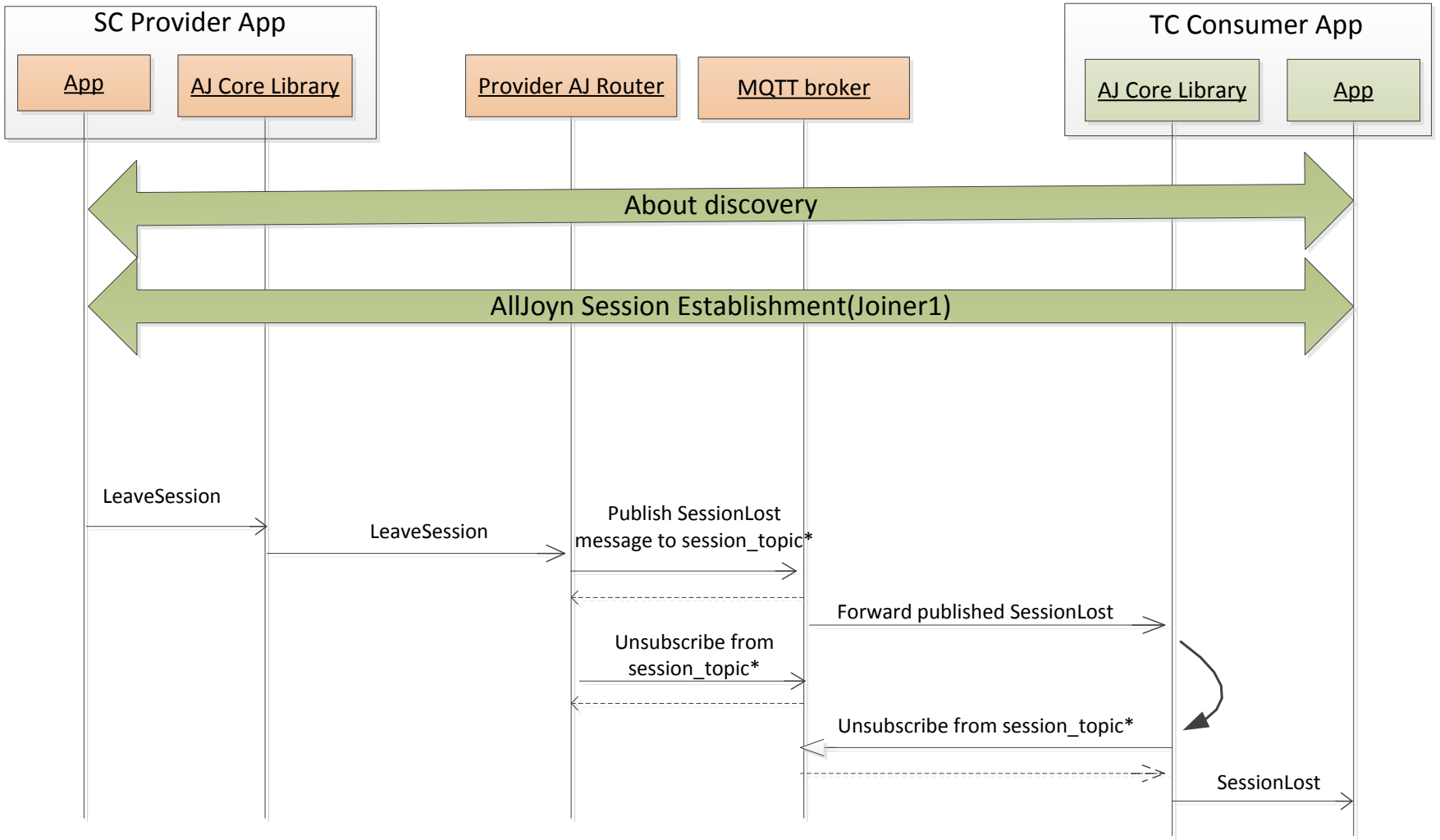
Multipoint session establishment



Leaving a Point to point session

- The consumer/provider leaving the session sends out a SessionLost signal to the session topic
- The other end receives the signal and cleans up the allocated session and forwards the signal to the thin/standard client app

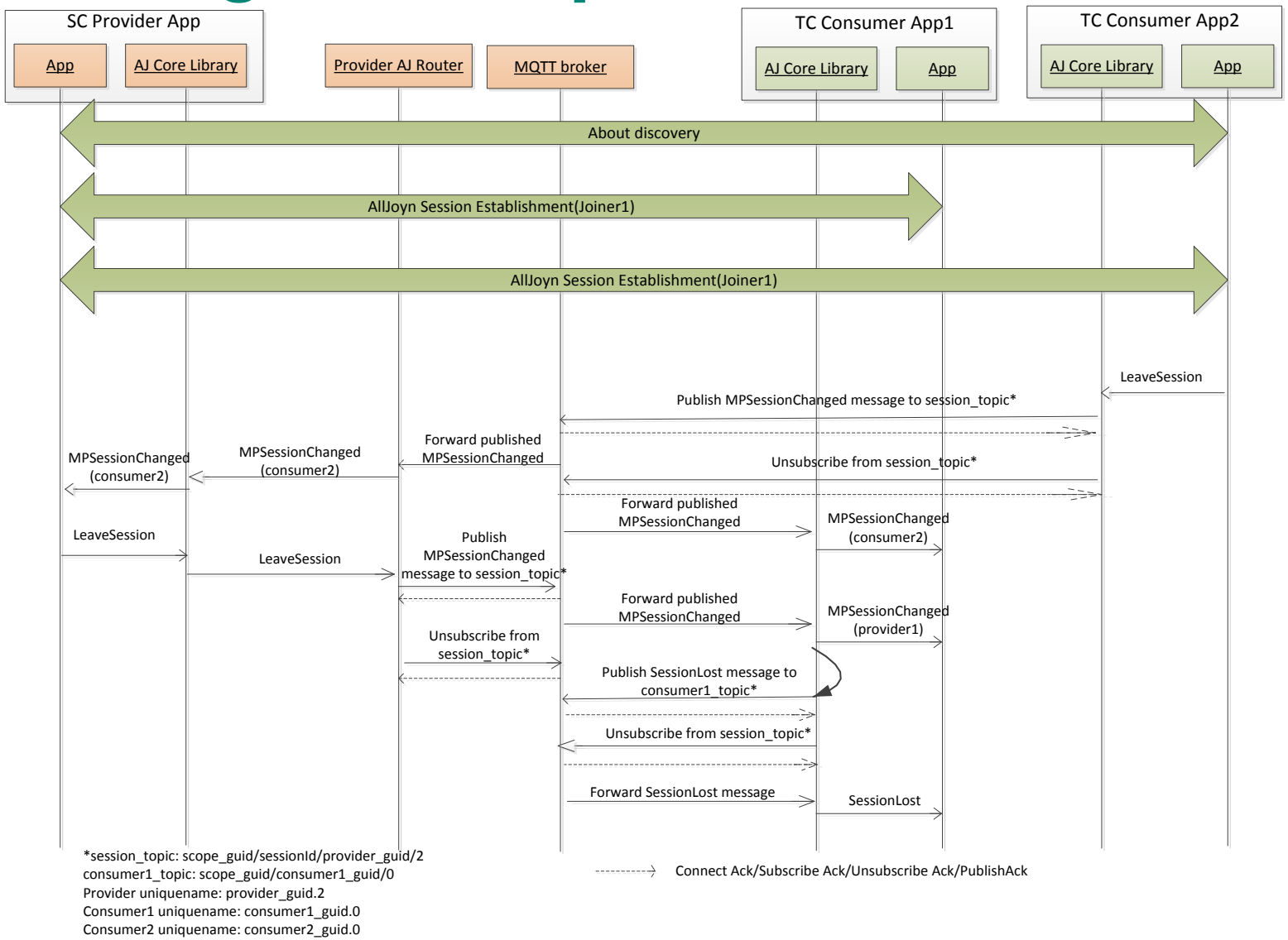
Leaving a point-to-point session



Leaving a multi-point session

- The consumer/provider leaving the session sends out a MPSessionChanged signal to the session topic
- The other session members receive the signal, clean up the allocated session and forwards the signal to the thin/standard client app
- If this was the last member in the session, the thin client library/router will publish a SessionLost signal to the broker destined for itself
- When the SessionLost signal arrives back, it is forwarded to the thin/standard client app

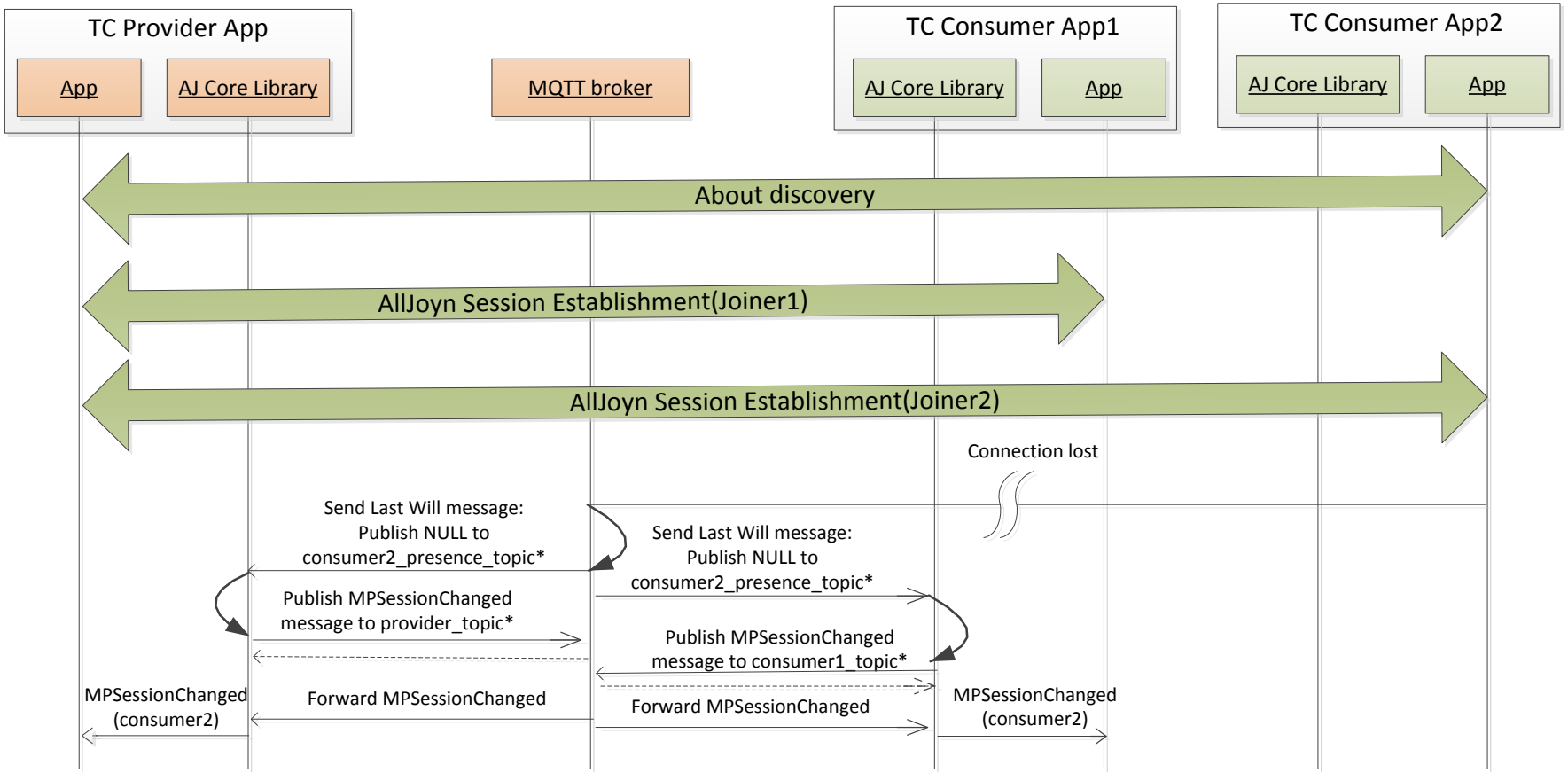
Leaving a multi-point session



Presence loss detection

- When a thin client/router leaves the network or does not send timely ping messages, the broker forwards the last will - a presence signal with a NULL payload to all its connected applications
- When this is received by the other thin clients/routers, they clean up all allocated sessions to the node that has left the network and send MPSessionChanged/SessionLost signals as appropriate

Presence loss detection



*session_topic: scope_guid/sessionId/provider_guid/0
provider_topic: scope_guid/provider_guid/0
consumer1_topic: scope_guid/consumer1_guid/0
consumer2_presence_topic: scope_gui/presence/consumer2_guid/0
Provider uniquename: provider_guid.0
Consumer1 uniquename: consumer1_guid.0
Consumer2 uniquename: consumer2_guid.0

----------> Connect Ack/Subscribe Ack/Unsubscribe Ack/PublishAck

Scavenger - introduction

- The scavenger is responsible for the following tasks:
  - Purging session-less signals that have timed out
  - Purging session-less signals emitted by the node when the node leaves the network
  - Purging session-less signals emitted by the standard clients connected to a router when the router leaves the network
  - Purging Presence signals for standard client nodes connected to a particular router, when the router leaves the network
- The scavenger subscribes to the presence topic for the scope that it is responsible for

Scavenger – Session-less signal cleanup

- When the scavenger receives a presence signal with a “true” payload for a particular thin client/standard client/router, it subscribes to the session-less signals from the same
- It tracks its topic string and timeout(if any) of session-less signals that it receives
- If the signal timeout expires, it publishes a NULL message to the particular topic, thus deleting the message
- If it receives the last will message for a thin client/standard client, it publishes a NULL message to all the session-less topics sent by that node
- If it receives the last will message for a router, it publishes a NULL message to all the session-less topics sent by all leaf nodes connected to the router

Scavenger – Standard client presence cleanup

- It tracks all standard clients that it receives a presence signal with “true” payload for, and keeps track of their router
- If the router leaves the network i.e. it receives the last will message for the router, it publishes a NULL message to the presence topics of all the standard clients that were connected to that router

AllJoyn over MQTT Test results

Test results

The following experiments have been performed successfully so far.

- Stress application joining sessions and sending signals/method calls continuously
- Session establishment: Multipoint session with 1000 nodes in session
- About announcement fetch: 100% signal reception for 1 consumer, 2000 producers sharing a single MQTT broker
- About announcement fetch: 3000 thin applications and 3000 standard client applications using two brokers, emitting and subscribing to About announcements with 100% reception

Thank you

Follow us on

For more information on AllSeen Alliance, visit us at: allseenalliance.org & allseenalliance.org/news/blogs