ALLSEEN ALLIANCE

AllJoyn® Thin Library Target Porting

Peter Krystad
Engineer, Staff/Manager
Qualcomm Connected Experiences, Inc.

Mathew Martineau
Engineer, Staff/Manager
Qualcomm Connected Experiences, Inc.

October 19, 2015

Agenda

1. Overview
2. Networking and Wi-Fi
3. Persistent Storage
4. System Resources and Utilities
5. Security
6. Build System

Overview

- Thin Library
  - History and motivation
  - Memory usage for simple application with security: ~125K code, 4K heap, 2K stack
  - Configurable for big- or little-endian
- Platform features required:
  - Memory allocation
  - Persistent storage
  - IP networking
  - Security (optional)
  - APIs discussed correspond to AllJoyn® v15.09

Application Integration

- The AllJoyn Thin Core Library is single-threaded, and blocks waiting for messages to be received
- C Code compatibility
  - Uses C89 language features, plus '//' comments
- Naming conventions
  - APIs use the 'AJ_' prefix
  - Files are named with an 'aj_' prefix

Networking and Wi-Fi

Networking Overview

- Does not provide general-purpose network access
- Routing Node Discovery
  - Leverages mDNS
  - Legacy support not required if not necessary to connect to pre-14.12 Routing Nodes
- Routing Node Connection
  - TCP
    - Legacy AllJoyn
  - UDP/ARDP
    - Product-ready in 15.04, use is prioritized over TCP
- Wi-Fi
  - Only required for applications that use Onboarding

Networking

- Networking APIs to be provided by porting layer are in inc/aj_net.h
- Discovery
  - AJ_Net_MCastUp(AJ_MCastSocket *)
    - Configure sockets and AJ_IOBuffers for discovery
    - Provide send and recv routines for the IO buffers
    - Thin library will format discovery packets in IO buffers and call IO buffer send
    - One call to AJ_Net_Sendto() routine sends datagrams to mDNS port via IPv4 broadcast and multicast and IPv6 multicast, on all interfaces
    - Thin library calls IO buffer recv to get responses
    - AJ_Net_Recvfrom() blocks on IPv4 socket bound to mDNS port for unicast discovery responses
    - Thin library consumes discovery response packets
    - Quirk: mDNS packets modified with IP address and port on the way out
    - Additional sockets required for legacy discovery
  - AJ_Net_MCastDown()

Networking

- Routing Node Connection
  - AJ_Net_Connect(AJ_BusAttachment*, AJ_Service*)
    - Configure socket and AJ_IOBuffers in the AJ_BusAttachment
    - AJ_Service as chosen by Thin library from discovery results
    - Provide send and recv routines for the IO buffers
    - Call AJ_ARDP_UDP_Connect() to initialize ARDP connection
    - If UDP/ARDP not configured or if UDP connection fails use TCP
  - AJ_Net_Disconnect()

Wi-Fi

- Optional support for Onboarding service
- APIs to be provided by porting layer are in inc/aj_wifi_ctrl.h
  - AJ_WiFiScan()
  - AJ_ConnectWiFi() / AJ_DisconnectWiFi()
    - Connect to network, returns after IP is up/down
  - AJ_EnableSoftAP()
    - Begin operating as AP, returns after one STA connected
  - AJ_ResetWiFi()

Persistent Storage

Persistent Storage Overview

- Need persistent storage for GUID, security credentials, network parameters, etc.
- Default implementation for Linux/Win32/Mac is a single file in the filesystem, but other targets may use a platform-specific implementation.
- Platform implementation must provide RAM-based image of persistent storage block.

Persistent Storage

- Porting layer supplies internal APIs defined in aj_target_nvram.h
  - uint8* AJ_NVRAM_BASE_ADDRESS
    - This symbol must be defined and initialized to point to an array of RAM of size AJ_NVRAM_SIZE that contains persisted storage contents. Core will only read from this array.
  - AJ_NVRAM_Init()
    - Initialize (load) the persisted storage into RAM array
  - _AJ_NV_Read() / _AJ_NV_Write()
    - Provide write-through to persistent storage
  - _AJ_NVRAM_Clear()
    - Clear contents of persistent storage to all FF's
  - _AJ_CompactNVStorage
    - Compress
- Alternatively implement APIs in inc/aj_nvram.h and leave out src/aj_nvram.c

System Resources and Utilities

System Resources

- AJ_Malloc() / AJ_Free() / AJ_Realloc
  - Platform-independent wrappers for malloc()/free()/realloc()
  - For constrained devices a small (4K?) pool dedicated to Thin Library needs is adequate
- AJ_InitTimer() / AJ_GetElapsedTime()
  - Millisecond accuracy timing

Utilities

- AJ_DecodeTime()
  - Use strptime() or format as “%y%m%d%H%M%SZ“
- AJ_Printf()
  - Varadic printf used for logging, use printf
- AJ_ByteSwapXX()
- AJ_MemZeroSecure()
  - Secure clear memory, ensure that this is not optimized out

Security

Security Overview

- AllJoyn uses cryptographic algorithms for certificates and securing connections
- Using platform crypto APIs is ***optional***: AJTCL open source code implements all required algorithms. However, these default implementations may duplicate platform functionality and do not use crypto hardware.
- Pick and choose which security modules to port
- Greg Zaverucha will cover “Cryptography in AllJoyn” at 10:30AM tomorrow.

AES Algorithms

- AllJoyn uses 128-bit AES-CCM (counter mode with CBC-MAC)
- Biggest runtime impact: All encrypted messages use AES-CCM
- Provides authentication for the full message (headers + payload) and encryption for the payload
  - Must support large Additional Authenticated Data
- Quirk: Compatibility with v14.12 and older requires a 5-byte nonce. Append zeroes to the nonce to reach the minimum length.
- Default implementation code size (x86): 4.5kB
  - aj_crypto_aes.c, aj_sw_crypto.c
- See aj_crypto.h
  - AJ_Encrypt_CCM(), AJ_Decrypt_CCM(), AJ_AES_Enable(), AJ_AES_Disable()

Random Number Generation

- A cryptographic pseudorandom number generator (CPRNG) is required for key generation
- The default implementation is CTR DRBG using algorithms from NIST SP 800-90A
  - Builds on the default AES code
  - Platform entropy is always required for seeding
- Platform APIs for random numbers have the benefit of more entropy sources
  - If you rely entirely on random numbers from the platform, make sure they are of cryptographic quality
- DRBG code size: 1.1kB
  - aj_crypto_drbg.c, aj_target_crypto.c

SHA256

- The SHA256 hash and HMAC algorithms are used during authentication. The SHA256 hash is also used with ECC certificates.
- Hash APIs are listed in aj_crypto_sha2.h
  - AJ_SHA256_Init()
  - AJ_SHA256_Update() – Provide data to hash
  - AJ_SHA256_GetDigest() – Get current digest value but allow further updates
  - AJ_SHA256_Final() – Free resources and optionally calculate digest
- HMAC is used within AJ_Crypto_PRF_SHA256() but not otherwise exposed
- SHA256 code size (x86): 6.9kB
  - aj_crypto_sha2.c, sha2.c

Elliptic Curve Cryptography (ECC)

- ECC is used in membership and identity certificates
- AllJoyn uses the NIST P256 curve with ECDSA and ECDH algorithms
- Default ECC implementation is constant-time
- Pre-15.04 AllJoyn versions use a nonstandard shared secret that will not have direct support in most ECC implementations
  - Current AllJoyn version supports this for backward compatibility
  - Optional if secure connections to pre-15.04 devices are not needed
  - See AJ_GenerateShareSecretOld()
- API is documented in aj_crypto_ecc.h
- ECC code size (x86): 25.3kB
  - aj_crypto_ecc.c, aj_crypto_ec_p256.c, and aj_crypto_field_p256.c

Tips

- Size network buffers and NVRAM appropriately for secure devices
  - Network buffers must be large enough to hold certificate chains during the claim process
  - NVRAM is used to store certificate information
- Test programs are included in the open source code
  - aestest
  - ctrdrbg
  - ecctest
  - shatest
- Make sure buffers with sensitive data are cleared using AJ_MemZeroSecure()

Build System

Build System Integration

- Windows/Linux/Mac builds are implemented in SCons
- GNU Make and other systems work as well
- Considerations when porting
  - Set up the include paths to point to platform system headers and AllJoyn headers
  - Define necessary symbols
    - AJ_NVRAM_SIZE (if different from default in aj_nvram.h)
    - AJ_NUM_REPLY_CONTEXTS (if different from default in aj_config.h)
    - AJ_TCP and AJ_ARDP (to enable network transports)

Closing

Implement required platform functions

AllJoyn requires networking, storage, memory allocation, and a few utility functions.

Create platform security layer

If platform functions are available, if the code space and runtime benefits may be compelling.

Configure your toolchain

AllJoyn Thin Library should be compatible with most toolchains

Contact the Core Working Group

For questions, feedback, or contributions:

allseen-core@lists.allseenalliance.org

https://ask.allseenalliance.org

Thank you

Follow us on

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