Google

RPC Binder

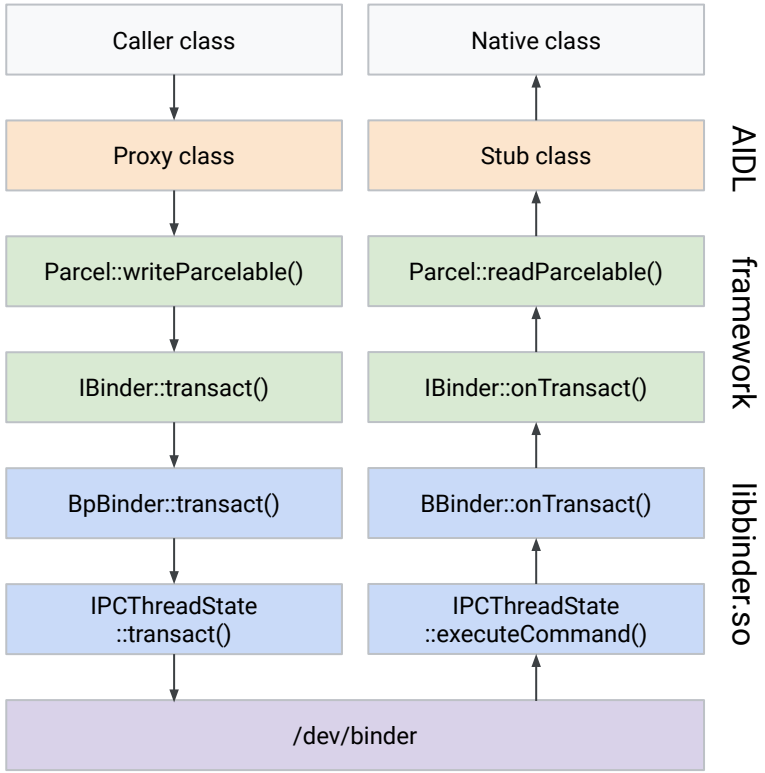
AIDL for distributed systems

David Brazdil
dbrazdil@google.com

July 2023

Binder/AIDL overview

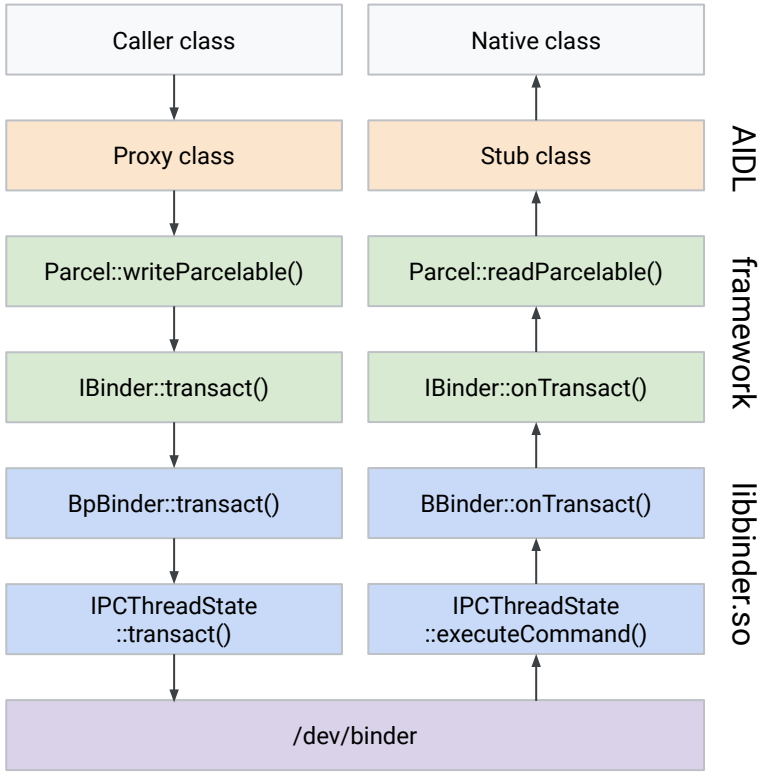
- Android relies heavily on Inter-Process Communication (IPC)
  - Binder is the primary IPC mechanism (by far)
  - Process boundary used for permission enforcement
- Binder = /dev/binder
  - Kernel driver facilitates exchange of buffers and FDs
  - Object aware - reference counting, death notification
  - Integration with scheduler to minimize latency
- Binder = libbinder.so
  - Shared library dynamically loaded by each client
  - Clients open `/dev/binder`, threads wait for events
  - Transaction wire format is not stable (by design)
- Android Interface Definition Language (AIDL)
  - Java-like syntax, serialization with Parcelable
  - `aidl` tool generates code bindings for Java/C++/Rust
- Binder = one remote object (proxy/native pair)



* heavily simplified

Why something else?

- AIDL has been hugely successful!
  - Used for interfaces on all levels of the stack, from HALs all the way up to applications
  - Treble would not have been possible without it
  - Binder is “just” an implementation detail
- We want to use AIDL in more places, but Binder requires:
  - Both clients to be running on top of the same kernel
    - What if the clients are not on the same device?
  - Both clients to share the same libbinder.so
    - What if the clients are updated independently?
- One potential solution is to use gRPC/protobuf
  - Would lead to fragmentation in interface definition, prevent reuse of existing AIDL interfaces

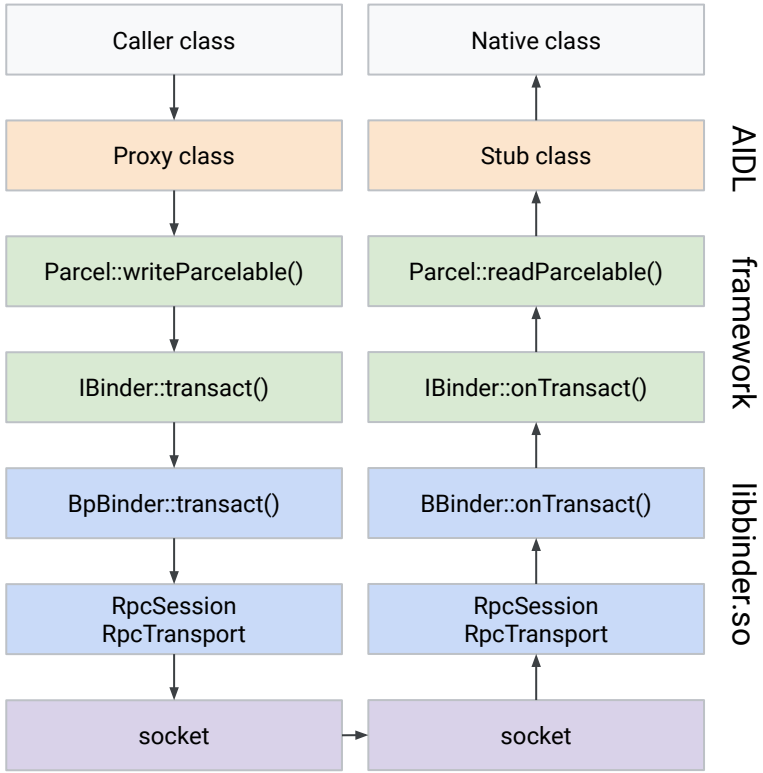


* heavily simplified

RPC Binder

- New abstraction to enable implementation of transports other than the Binder kernel driver
  - First implementation: POSIX sockets
  - Supporting AF_INET, AF_UNIX, AF_VSOCK

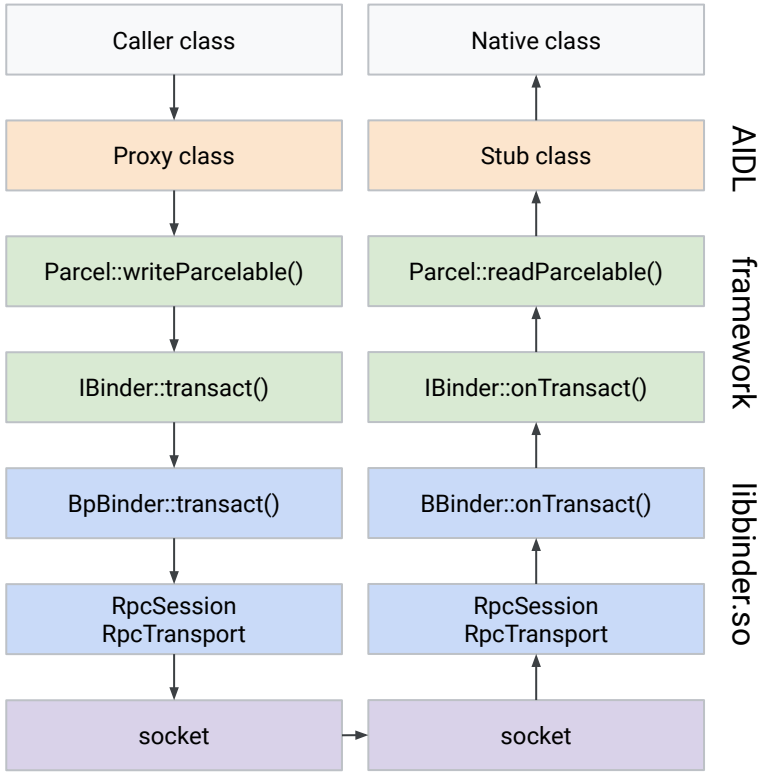
- One client starts `RpcServer` to listen for new connections, other client connects to it with `RpcSession`
- Each session dynamically spawns threads to handle `RpcConnections`, up to a configurable maximum
- `RpcSession::shutdown()` on either end interrupts all threads and closes all connections



* heavily simplified

RPC Binder

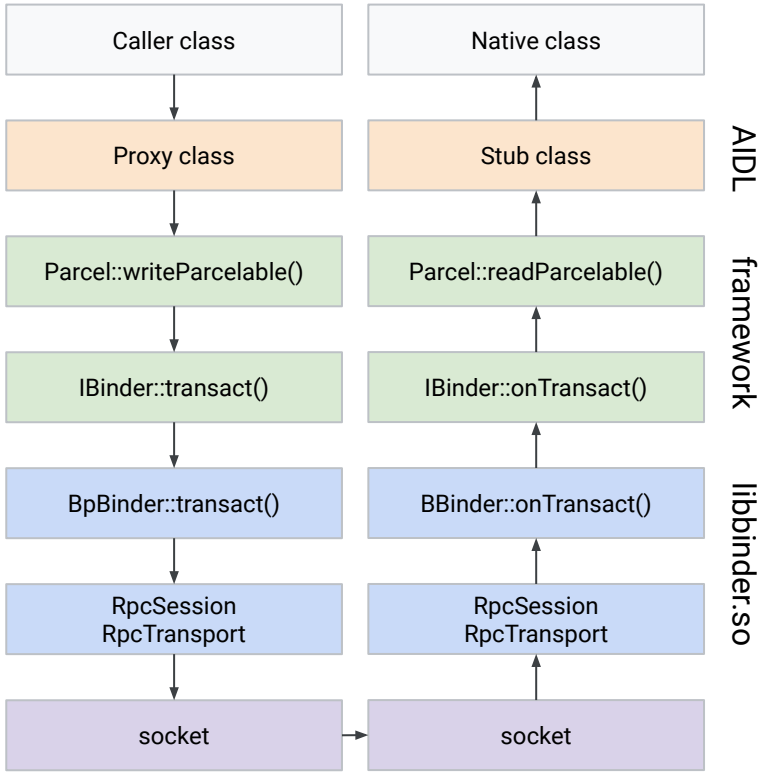
- Object management logic moved from kernel to userspace
  - Reference counting done by the object owner
  - Death notification either from owner or from HUP
- Support for TLS-based authentication/encryption
  - Authentication in either direction, configurable to specify a list of trusted certificates or CAs
  - Initialized by constructing `RpcServer`/`RpcSession` with `RpcTransportCtxFactoryTls`
- (WIP) Versioned wire format
  - No stable release yet, but protocol ready for versioning



* heavily simplified

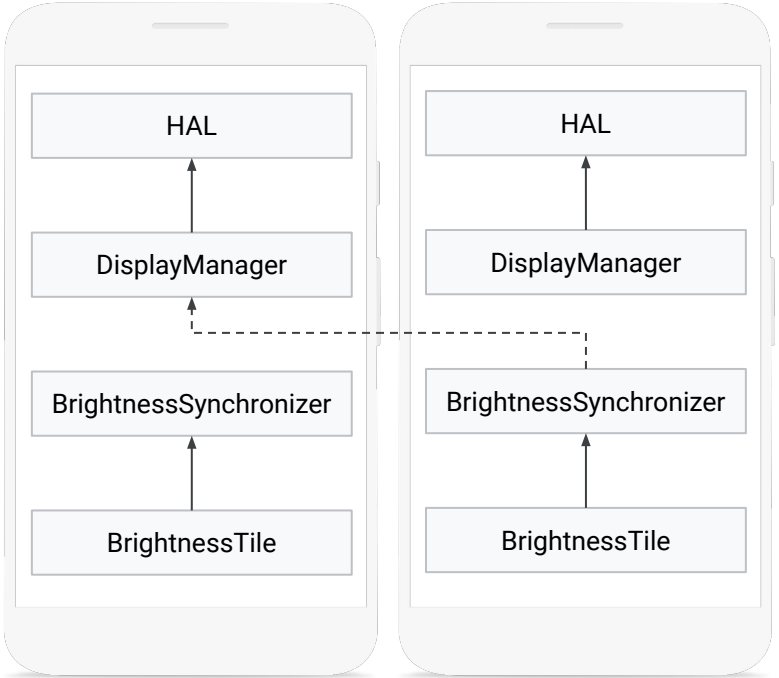
Differences from kernel Binder

- Only some address families support passing FDs
  - UDS does, inet and vsock do not
  - See authfs on a later slide
- No global object namespace
  - Each `RpcSession` creates a local object namespace shared only between the two endpoints
  - If client A passes an object handle to client B, it cannot be passed on further to client C
- No support for `getCallingPid/getCallingUid`
- No explicit scheduler hint
  - Although no reason why kernel could not optimize some address families, such as UDS
- Weak pointers non-promotable once refcount drops to zero



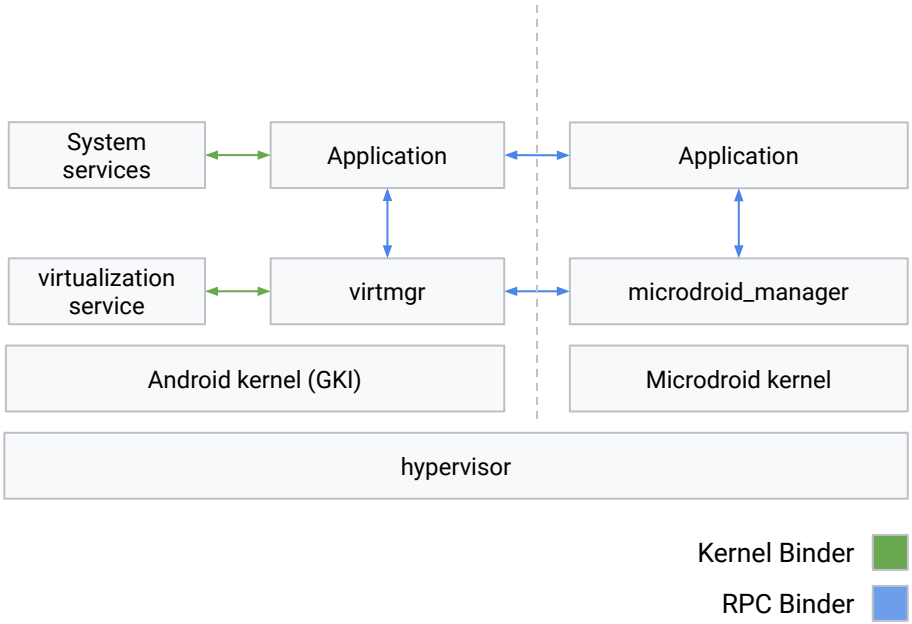
* heavily simplified

Demo



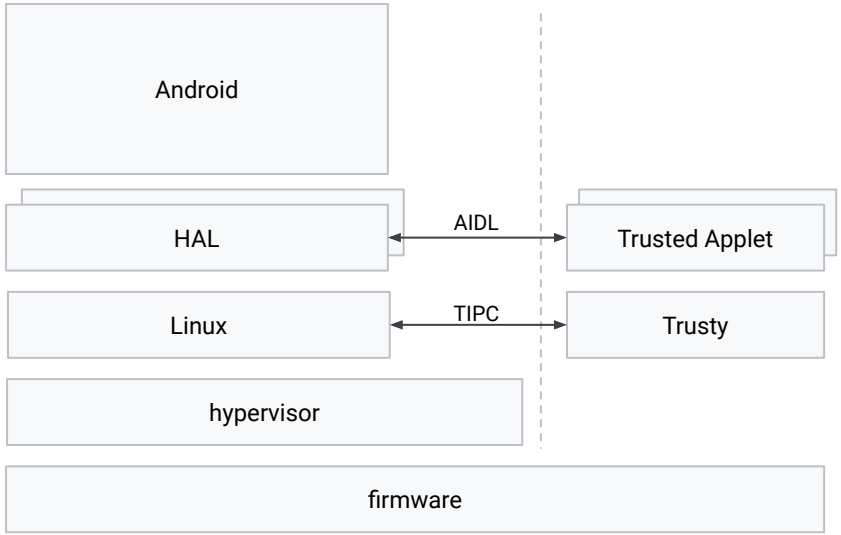
Use case: Virtualization Framework (AVF)

- AVF allows apps to offload execution of a native library into an isolated VM
  - Communication over vsock
  - Seamless-ish with AIDL over vsock
- All IPC in Microdroid is RPC Binder over UDS
  - Save space by removing kernel Binder
- App↔virtmgr is RPC Binder over UDS
  - Basically just talking over a pipe…
- authfs: FUSE-based remote file system with fs-verity
  - Client/server communication over RPC Binder
  - Currently a separate project, could be merged



Use case: Trusty

- Trusty = reference TEE OS for TrustZone (TZ)
- Uses Transparent Inter Process Communication (TIPC) for communication between Linux and TZ
  - Similar to UDS but for use across clusters
- Trusted Applets (TAs) form back-end of security HALs
  - Require a proxy to translate HAL method calls to TIPC requests and back
  - Solution: RPC Binder over TIPC
- Maybe someday: HAL implementation directly in TEE
  - Would make the services reachable from VMs

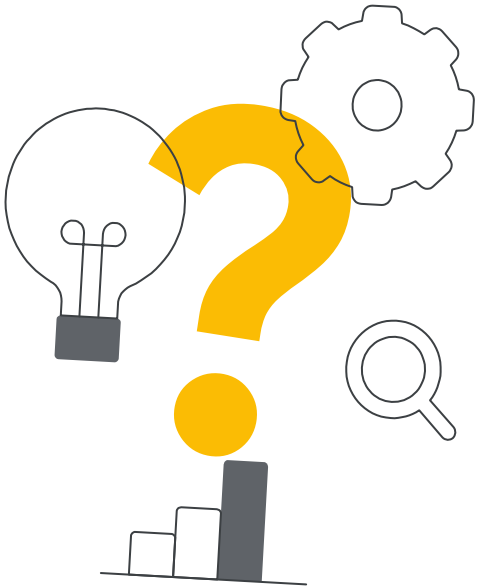


Other ideas

- RPC Binder not limited to Android
  - Can be compiled for regular Linux without bionic
  - Port to Trusty required a small amount of work (custom transport, single-threaded)

- Can be useful for testing - running tests on PC, mocking interfaces, remote debugging
- Example: `createRpcDelegateServiceManager`
  - Creates a proxy to ServiceManager running on a device over ADB

- Communication between multiple Android devices, e.g. your devices at home, …
- Communication with Android deployed on a network, e.g. in the cloud, in a vehicle, …



Thank you

Find source code in AOSP tree:

```
gomod binderRpcTest
```

Run tests in AOSP tree:

```
atest binderRpcTest
```