## 1. IMPLEMENTATION

Moving from simulation to implementation on physical devices can reveal unforeseen challenges to the design, protocols, or experimental conditions around which a system is developed. We therefore see the real implementation, in addition to the simulation, as a necessary step to demonstrating our system's usefulness and effectiveness; for that reason we have implemented our system both on the NS3 simulation platform and on real Android mobile devices.

The Android platform represented 80% of the mobile device market in 2014 and is expected to maintain three quarters of the global market share during the following four years cite International Data Corporation http://www.idc.com/getdoc.jsp?containerId=prUS24857114. We feel that targeting Android gives the most realistic view of what capabilities are available, now or in the future, for novel mobile systems.

The core system functionality is implemented as a standalone C++ module that can be built as an external library or into an application. The core library contains all of the platform-agnostic functionality and accesses platform resources via abstract interfaces.

Cross-platform support is fundamental to our software design: we can support NS3, Android, POSIX, and WinNT with no modifications to the core module. A thin platform-specific wrapper provides access to the environment, as seen in Figure 1.

NS3 and Android differ in important ways; for one, the NS3 simulator uses only a single thread of execution for numerous application instances while Android offers many threads of execution for a single application instance. To be maximally flexible to different test environments or deployments, the core system logic is implemented in terms of the following abstractions:

**Clock** gives access to the current time and date. In NS3 this reflects the simulation time, but in all other implementations it reflects the system clock.

**Async** provides a mechanism for the core module to create tasks that may be run in the background, in the future, or periodically. Task scheduling and parallelism, if any, is defined by the implementation. The core functionality is thus scalable across many threads of execution as it operates in terms of decoupled tasks.

**Cache** is how the core module stores and loads data, in lieu of files or databases. Caches of varying persistence are provided that may be implemented in volatile memory, on a hard disk, or in flash memory on a mobile device. The caching policies may vary given the storage medium and expected persistence of cached data.

**Links** represent logical network connections and are abstracted as message queues. The implementation defines the network device and access mode of a particular link. The link layer aggregates one or more links into the logical network view held by the core logic.

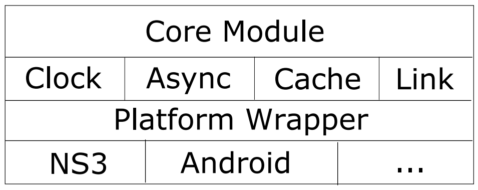

**Figure 1:** **The implementation architecture abstracts platform resources away from the core functionality for platform independence.**

## 2. NS3 SIMULATION

Developing against the NS3 simulation environment involves writing "scripts" that construct a network topology and insert into it network devices bearing applications. The scripts in our implementation are standalone C++ applications that link to a static library containing the core functionality, a wrapper module that implements the abstract facilities using NS3, and a set of shared libraries containing the NS3 platform (Fig. 3).

The simulation begins when each node schedules some event in the future. NS3 processes the events for the current time, potentially generating more events, before incrementing the time. The nodes are unaware that they are in a simulation and behave identically to if they were physical devices whose system clock reflected the simulation time.

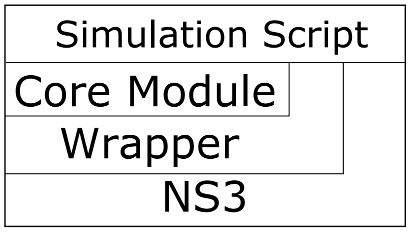

**Figure 2:** **NS3 simulation scripts are standalone applications that link to the core module, the NS3 platform implementations, and NS3 itself.**

## 3. ANDROID IMPLEMENTATION

Android applications are bytecode programs executed on the Dalvik virtual machine (VM), which compiles them to architecture-appropriate machine code at run time. The Android Native Development Kit (NDK) enables C and C++ code to be compiled to target specific processor architectures, packaged as shared libraries, and deployed with Android applications. These libraries can be loaded by the Dalvik VM and integrated into Android applications using a thin layer of glue to translate between the VM and native code.

Using cross compilers provided with the NDK we are able to compile the core module to target x86 and ARM Android devices. An additional wrapper module implements the core abstractions in terms of the Android platform, e.g. by automatically configuring WiFi interfaces.

A small Java-language application serves two roles: first, it loads the native modules in a long-running background service so that the core functionality can operate while the user is multitasking; second, it provides the user interface for creating and viewing content.

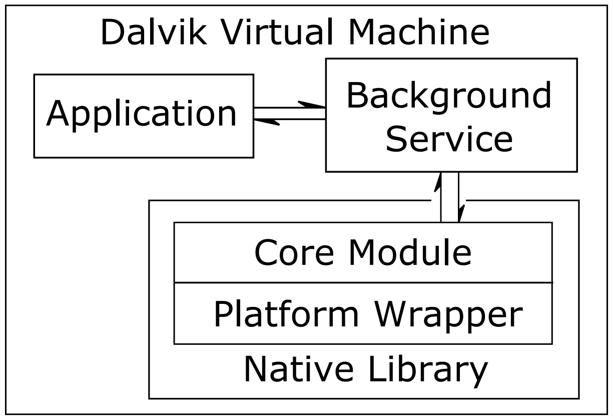

**Figure 3:** **The Android application communicates with a constantly-running background service that hosts the native code.**

We deployed the application across five ASUS Nexus 7 tablets with ARMv7 processors and one LG Nexus 5 smartphone with an ARMv7 processor. All devices were running the CyanogenMod build of the Android operating system, as this project contains non-standard 802.11 firmware to support IBSS mode for ad-hoc networking.