Smartphone Qualification & Linux-based Tools for CubeSat Computing Payloads

C. P. Bridges, B. Yeomans, C. Iacopino, T. E. Frame
Surrey Space Centre, University of Surrey
Guildford, Surrey, United Kingdom
Tel: +44(0)1483 689137
{c.p.bridges b.yeomans c.iacopino t.frame}@surrey.ac.uk

A. Schofield, S. Kenyon, M. N. Sweeting
Surrey Satellite Technology Ltd.
Guildford, Surrey, United Kingdom
Tel: +44(0)1483 804241
{a.schofield s.kenyon m.sweeting}@sstl.co.uk

Abstract— **Modern computers are now far in advance of satellite systems and leveraging of these technologies for space applications could lead to cheaper and more capable spacecraft. Together with NASA AMES's PhoneSat, the STRaND-1 nanosatellite team has been developing and designing new ways to include smart-phone technologies to the popular CubeSat platform whilst mitigating numerous risks. Surrey Space Centre (SSC) and Surrey Satellite Technology Ltd. (SSTL) have led in qualifying state-of-the-art COTS technologies and capabilities - contributing to numerous low-cost satellite missions. The focus of this paper is to answer if 1) modern smart-phone software is compatible for fast and low-cost development as required by CubeSats, and 2) if the components utilised are robust to the space environment. The STRaND-1 smart-phone payload software explored in this paper is united using various open-source Linux tools and generic interfaces found in terrestrial systems. A major result from our developments is that many existing software and hardware processes are more than sufficient to provide autonomous and operational payload object-to-object and file-based management solutions. The paper will provide methodologies on the software chains and tools used for the STRaND-1 smartphone computing platform, the hardware built with space qualification results (thermal, thermal vacuum, and TID radiation), and how they can be implemented in future missions.**

TABLE OF CONTENTS

1. INTRODUCTION & STRAND-1

STRaND-1 is the first in a series of Surrey Satellite Technology Ltd. (SSTL)-Surrey Space Centre (SSC) collaborative satellites designed for the purpose of technology path finding for future commercial operations. The aims of the STRaND (Surrey Training, Research and Nanosatellite Development) programme are:

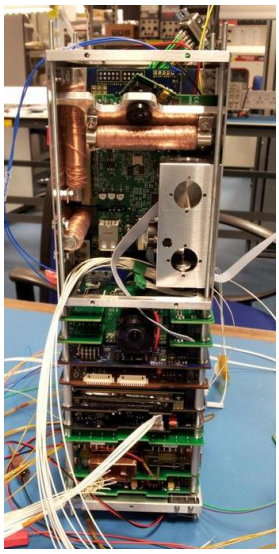
- To challenge both the current industry standard development processes and traditional Surrey approach to discover new ways of designing, manufacturing and testing space hardware,
- To demonstrate novel space technologies or the use of existing but modern terrestrial commercial-off-the-shelf (COTS) technologies in space, and
- To provide a rapid hands-on training experience for less experienced engineers and academics in designing and building new satellite technologies.

The STRaND programme builds upon a similar programme a decade ago, which resulted in the highly successful SNAP-1 nanosatellite mission, launched in 2000 [1]. It is the first time Surrey has entered the CubeSat field and differs from most CubeSats in that it will fly a modern commercial off-the-shelf (COTS) Android smartphone as a payload, along with a suite of advanced attitude and orbit technologies developed by the University of Surrey and CubeSense [2] from the University of Stellenbosch in South Africa. STRaND-1 is also different in that anyone (not just from the space engineering or space science community) will be eligible to fly their "app" in space, for free. STRaND-1 is currently being manufactured and tested by volunteers in their own free time, and will be ready for an intended launch in Feb' 2013. STRaND-1 can be seen in assembly, integration, and test (AIT) in Fig. 1. Further information on STRaND-1 and the design can be found in references [3] [4] [5] but this paper aims to focus on the Android smart-phone payload and the hardware and software required to operate within the common CubeSat, together with thermal vacuum testing and total ionizing dose (TID) radiation results. Given the volunteer-based nature of the project, this paper does not aim to answer the strengths/weaknesses of Android Linux vs RT/commerical Linux distributions, or subsequent soft or hard real-time questions – these can be answered primarily by kernel-level debugging of the Linux system under test.

The paper is divided further into 4 keys sections. Section 2 describes the operational concepts and how the payload is operated in the STRaND-1 CubeSat, Section 3 describes the data-flow and Linux tools use, Section 4 describes the Android ecosystem of apps and how to securely operate

978-1-4673-1813-6/13/$31.00 ©2013 IEEE

apps for STRaND-1, and Section 5 describes the qualification tests carried out on the smart-phones towards space flight. Section 6 concludes.



Payload Bay
Non-PC/104 subsystems
Custom Resistojet
Pulsed Plasma Thrusters (PPTs)
Magnetorquer Rods
Reaction/Mom. Wheels
Android smart-phone
Linux Single Board Computer
SGR05U GPS Receiver

Standard PC/104 Stack
CubeSense
On-board Computer (OBC)
EPS & Battery System
Pulsed Plasma Thrusters (PPTs)
Custom Modem Backend
Custom RF Frontend

Figure 1. STRaND-1 in AIT (Oct' 2012)

2. PAYLOAD OPERATIONAL CONCEPTS

As described in Section 1, the smart-phone payload operations are divided in three physical subsystems: an interface motherboard, single board computer, and the smart-phone itself. The motherboard is a custom-design and includes a number of interfaces that would be needed on CubeSat missions such as I^2C, UARTs, a magnetometer, and H-bridge drivers – all controlled via a PIC24. The single board computer is a Digi-Wi9C from Digi [9] and provides I2C interfaces to the motherboard and USB & WiFi interfaces to the smart-phone. The smart-phone is a stock Google Nexus-One [10].

In STRaND-1 the Gomspace Nanomind A712C OBC [11] is used as the primary I2C bus master operating in the classical I^2C frame, shown in Fig. 1, where the node is the 7-bit I^2C address, a read/write bit, a 1 byte channel to be read or written to, and finally the data if writing to a particular channel.

Node (7b)	R/W (1b)	Channel (1B)	Data (if write)

Figure 2. Basic I2C Frame

As the Digi-Wi9C is also an I^2C bus master, the motherboard must provide I^2C bus arbitration and a common memory area. To achieve this, the OBC sends an I^2C telecommand (TC) to the PIC24 of the motherboard which sets a GPIO pin from the motherboard to a GPIO pin on the Digi-Wi9C which indicates that it is allowed to use the I^2C bus. The Wi9C uses an interrupt service routine (ISR) from this pin and, when set, will then issue its own I^2C telemetry (TM) or telecommand requests for a default 100 ms. The allowable time the smart-phone payload has to act as I^2C master is stored in a PIC24 2B register and can be set as TC by the OBC or read as a TM point by the Wi9C.

The motherboard PIC24 must service requests for reading/writing TC/TM I^2C commands, and therefore must additionally control the memory for storing both TM readings and file-based transfers. The motherboard PIC24 uses internal 8 kB of SRAM and has a SPI connection to 8 Mb FLASH and I^2C-accessible 512 kb EEPROM. The 8 Mb FLASH is used to directly allocate and map I^2C channel addresses to FLASH memory addresses. This way, the payload data can persist between the spacecraft and smart-phone payload if the hardware is off and also be robust against single event upsets (SEUs). This architecture is shown in Fig. 3, together with some of the crucial software additions used to operate the smart-phone payload.

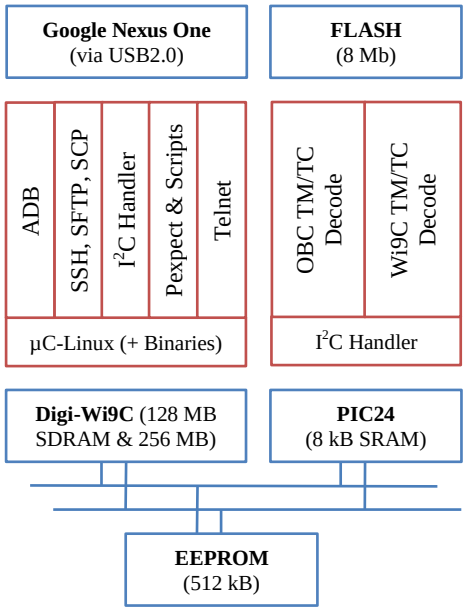


Figure 3. Smart-phone Payload Architecture

Operationally, the motherboard provides additional interfacing, arbitration, and a common memory area between the spacecraft bus and smart-phone payload.

3. DATA FLOW & LINUX-BASED TOOLS

As the Wi9C is used to communicate between the two key subsystem areas, there will be varying types of data and instructions flowing between the motherboard's PIC24, the Wi9C itself, and the smart-phone.

4.1. Data Flows & Wi9C Task Considerations

This is a summary of the different types of data flows and telecommand/telemetry flow considerations:

4.1.1. Packets between Wi9C & Smart-Phone—The packet of TM/TC data originating in the OBC will be sent to the Phone over a *telnet* [12] link with the aim to match the OBC

generation cycle. I.e. if TM/TC is generated on a 1 second cycle, it will be sent to the phone every second.

All TC/TM points are available within the I^2C packet frame. This requires parsing of packets to extract Wi9C relevant data or instructions to be sent on to the smart-phone. These could be for housekeeping, to upload/download files from the phone, start/stop apps, etc. Therefore transfers between the Wi9C & smart-phone will comprise primarily of routing and parsing of packets from bytes to/from strings. The packet format and data flow with the software framework which achieves this is as follows:

1. For flexibility, shell and Python scripting will be used to a) setup and initialise communications and b) file transfer objectives. This format will allow for scripts to be modified via upload and to preserve maximum flexibility. The Wi9C/smart-phone transfers are not time critical and thus the (small) overhead added with the scripting approach should be manageable.
2. On start up, the Wi9C will run a one-shot configuration script to start and configure relevant services, start a Master Controller Application (MCA) or 'app' on the phone and open the telnet link. This process takes several seconds to stop & restart this link and thus expected to be running all the time.
3. Because the interactive telnet link will be running, script execution can most easily be managed using the Python communication script as the master, running in a while loop. The Python script calls an I^2C read routine and the script will deliver the packet to the phone every second, and then parse the TM/TC. In the absence of TC instructions, loop timing will effectively be controlled by the OBC as the surplus time in the loop will be spent waiting for the bus free signal as previously described.

For file transfers to the smart-phone, standard directories and file names are utilised so the Wi9C knows where to put a replacement script. File transfers are managed via script in one operation, and there is no significant speed restriction.

In contrast, file transfers between the Wi9C and OBC need more attention via the motherboard PIC24 as the I^2C bus is restricted to 400 kbps transfer speed and the OBC buffer memory is also limited to 256 B packet sizes. It follows that files of any size will have to be chunked, requiring a further header; described in Fig. 4.

The header provides enough data to ensure files can be reassembled in the knowledge there are no missing packets, etc. It is assumed that all transfers will be binary as with typical Unix-based systems.

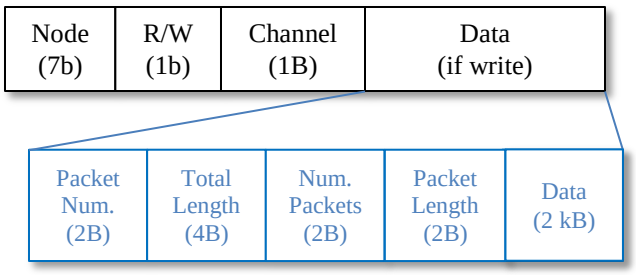


Figure 4. Wi9C File-based TM/TC Packet Header

4.1.2. Interprocess Communication—As communication over the telnet link is typically interactive, such as on Desktop PCs or laptops, the link remains open unless closed by the server, irrespective of whether the correct actions are taken by the recipient of data. Therefore an explicit instruction and response format is required to confirm receipt of TC/TM instructions, data, and acknowledgement messages. The link will be managed by a Python script incorporating the 'pexpect' module [13] which is specifically designed for this role. The Wi9C, as the client, waits for the correct response from the smart-phone server for acknowledgements. There are three potential interprocess communication channels available without physical modifications of the smart-phone:

- ABD: Communication over USB using the AOSP tool Android Debugging Bridge (ADB) [14] – a low level communications interface over the USB physical link. When implemented on Linux, no additional drivers are required. In an Android context, it is used very extensively and believed to be highly robust. ADB can be used to enable specific tasks (e.g. logging, shell access) and as a general communications link via the facility to forward TCP ports for telnet access. The ADB daemon is implemented on the phone by default, and operation of ADB over USB is persistent between phone reboots. The connection should be physically robust.
- Ethernet communications over WiFi: This solution likely to be less reliable despite the < 10 cm transmission distance. This solution will increase power consumption and requires modification to the stock Android ROM image to utilize ad-hoc networking modes.
- Ethernet communications over the USB physical link (USBNET): Although this provides an alternative to ADB with similar physical robustness, it does not appear to offer any real advantages over the ADB low level link. The phone USB interface needs to be re-configured to set up USBNET and the setting is not persistent between reboots – so to utilize this channel will require programming changes.

In view of the above issues, it was concluded that the primary communications channel should be via ADB as this can achieve both the specific housekeeping tasks and a general TCP based communications channel – however the

potential to achieve communications via Wi-Fi or USBNET were also investigated to ensure that at least one alternative would be available in the event that problems were experienced with ADB.

4.2. Software Package Requirements & Solutions

The previous discussion provides the requirements and solutions for robust Wi9C to smart-phone communications. They can be satisfied using the following tools and solutions:

1. A general purpose TCP/IP socket server implemented in the MCA on the smart-phone – the Wi9C can connect to this using a standard Linux telnet client.

The TCP socket server is custom bespoke code. But the TCP client, telnet is provided on the Wi9C as part of the *Busybox* [15] multi-call binary.

2. A second telnet or SSH server package implemented at the Linux operating system level on the phone – the Wi9C connects with a telnet or SSH client.

utelnetd [16] is a small footprint and well tested telnet server, used on Android. The *Dropbear* [17] multi-call binary is a very popular SSH server used extensively in embedded Linux systems.

3. An FTP or SFTP server implemented at the Linux operating system level on the phone to facilitate file transfers. The Wi9C can again run a standard Linux client to interact with the server(s).

A number of robust production level FTP servers were investigated, including *VSFTPD* [18] and *PureFTPD* [19]. Unfortunately, there were problems building these applications for the Android system and, although Android is based on Linux, there are many differences. For example the libraries, for licensing reasons, have been re-written from the glibc standard and often are missing important elements. E.g. Android does not implement a password based log in system, which is a particular problem with FTP comms, etc. In contrast, the dropbear application can be extended using the sftp-server application from the *openssh suite* [20], and there is existing experience of building these applications for the Android platform; for example, the dropbear native binaries are implemented in the QuickSSHd Android application [21]. The code to achieve this is open source and readily available. A particular benefit is that authentication can be achieved using key pairs rather than passwords thus mitigating lack of password support issues in the Android native code platform.

The typical user FTP client utilizes a GUI (e.g. Filezilla) which is clearly inappropriate for use in an automated scripting environment. Therefore the CLI networking package *Curl* [22] is used to automate file transfer tasks. Curl can in turn be operated from a bash or python script.

4. All of the communications will be driven by automated Python scripting on the Wi9C. This results in some additional complexities in task execution which need resolving.

The automation of telnet and operation of interactive TCP links pose particular problems for scripting applications. For technical reasons associated with how Linux handles I/O buffering, the normal approaches to script automation (e.g. using pipes) does not function properly in conjunction with telnet communications. In a desktop Linux context, the normal approach uses *expect*, a scripting language based on *tcl*. Whilst this works well, this approach will add unacceptable overhead to the Wi9C platform. There is however an alternative – the general purpose scripting language python is already implemented on the Wi9C platform. A python script *pexpect* [13] has been developed which achieves similar functionality to expect within a python context; it contains no additional native modules to those in a standard python distribution so is guaranteed to function in any normal python environment. As the Wi9C platform comes with a python implementation as standard and so this appears to be the most appropriate route.

In summary, there are many regular and powerful software packages and tools already existing which can be recompiled to operate on the Android or Wi9C platforms.

4. ANDROID CONCEPTS & APP ECOSYSTEM

The STRaND Android Application Framework is a key element in the entire software chain allowing Android applications to run on the phone and downloading their data on ground. The aim behind this framework is to be able to use any type of normal Android application giving them the capability of retrieving spacecraft telemetries and requesting the download on ground of data generated on the phone for fast development and integration into the STRaND-1 software. All code can be found on the Google Code Project called S-Android [23]. The main challenges behind this framework are the following:

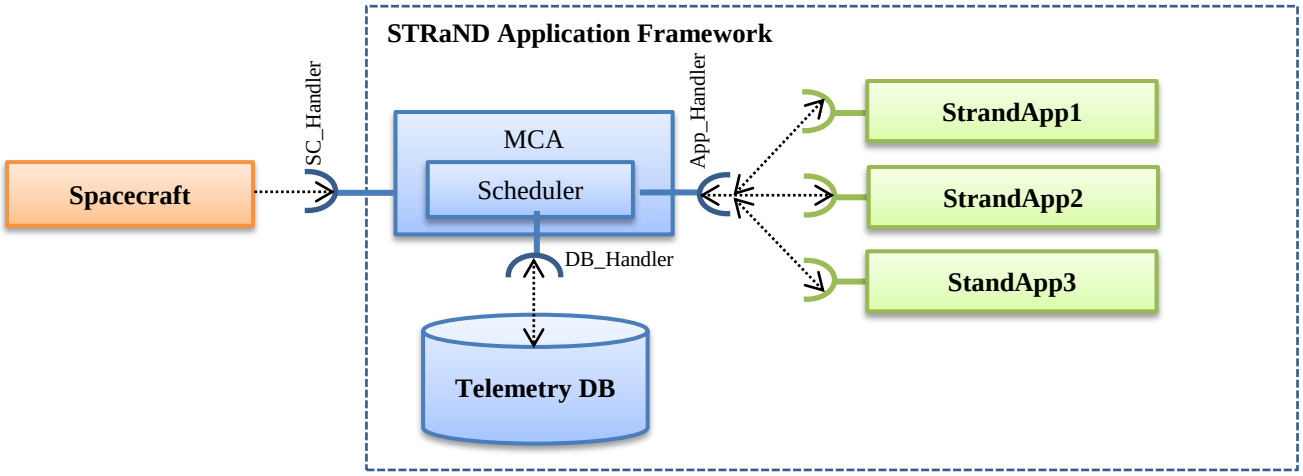


Figure 5. STRaND Application Framework

- Applications management, the application lifecycle has to be controlled centrally; a failure in one of the applications cannot block the entire framework.
- Inter-process communication system, it needs to able to synchronize all the components from the spacecraft to the apps.
- Seamless process to transform a generic Android app into a STRaND app.

Given these requirements the framework developed can be summarized in Fig. 5. In the following section, each component will be described putting in evidence how previous challenges are solved.

5.1. The Master Controller Application (MCA)

The MCA is an Android application, which runs a service component to run other applications in the background without any user interface. The MCA is the primary communication node between all the additional apps, together with the telemetry database (DB) and the spacecraft. Moreover, this component addresses the first challenge; controlling the apps' lifecycle. However, in case of failures of the MCA itself a more general design paradigm has been put in place: all the STRaND apps are bound service. A bound service is a type of Android service allowing other applications to bind to it and interact with it. The advantage of a bound service is that it runs only as long as another component is bound to it, in this case the MCA. When the MCA unbinds it the service is destroyed. This system guarantees that if the MCA terminates, all the STRaND apps terminate as well. The MCA is composed of the following elements, including:

- A *Spacecraft Handler* to communicate directly with the spacecraft OBC via the methods described in the previous Section 4. It is a separate thread that opens a socket on a specific port and keeping listening for spacecraft commands,
- A *DB Handler* to control access to the Telemetry DB and service requests to a) populate it with new telemetries coming from the spacecraft or from the phone and b) allow data to be retrieved as requested by the ground segment or by apps,
- An *Application Handler* to control app communications. Like the spacecraft handler, it is implemented a separate thread, and
- *Scheduler*, this component regulates the lifecycle of all the apps. It is a separate thread that is continuously updating the current application schedule with the new commands coming from the spacecraft and running or stopping the apps depending on the current schedule.

5.2. Telemetry Database

The Telemetry DB is a standard database implementation of and can be opened using SQL programs or converted to csv files for further interrogation.

Figure 6. Telemetry Database Example (temp = 33.7°)

The Android system supports a basic database engine called sqlite3. It provides a limited set of primitive types and reduced functionality from ANSI 92 SQL. STRaND-1 has implemented a simple telemetry database consisting of two initially identical tables comprising of fixed size records of ID, Timestamp, Telemetry Node identifier, Telemetry Channel identifier, and 8 Byte Data block.

The two tables are used for the spacecraft telemetry and the phone telemetry. They were kept separate despite the identical initial structure to facilitate greater flexibility in the event of the spacecraft telemetry being subject to change

during the project. The phone apps can only request telemetry from the database via the MCA but have access to the spacecraft or the phone telemetry. The 8 byte data block is defined as a fixed size but the usage is dependent on the telemetry channel. This may mean that only some of the 8 bytes are utilized but leads to a simpler messaging and database implementation. Functionality is available to strip standard primitive types from the 8 byte block. To facilitate backup and auditing, the database, which resides in the secured Data folder on the phone's memory, can be duplicated onto the SDCard and downloaded using the standard file download mechanism.

5.3. Android IPC

Android allows a number of ways to communicate among applications. An interface to work across different processes is required with the simplest being a Messenger that queues all requests into a single thread. In this manner, the service defines a Handler that responds to different types of Message objects. The MCA instantiating the apps with *bindService()*, retrieve a Messenger object for each of them. This allows the MCA to send message to any of them asynchronously. The messaging paradigm implemented across the entire framework is a request-driven approach going in one direction: from spacecraft to apps.

The MCA is always listening for messages coming from the spacecraft, thus communication is always started by the spacecraft. A specific list of commands allows the following functionalities:

- Editing the applications schedule,
- Retrieving URIs of files to be downloaded,
- Sending URI files to the MCA,
- Sending S/C telemetries to the Telemetry DB,
- Sending commands to the MCA or to the apps,
- Editing the MCA or apps settings, and
- Editing socket settings (timeouts).

The actual copying of files to and from the phone SD card is done through ADB. The MCA periodically sends messages to all the running apps to check if they request any action from the MCA. The apps cannot start communication with the MCA ensuring deterministic IPC and eliminating any risk of overflowing the MCA messaging queue buffer.

5.4. STRaND Application Wrapper

A key element addressed by the framework is to provide a seamless way to transform a generic Android app into a 'STRaND' app – an application able to communicate with the MCA. The architecture is shown in Fig. 7.

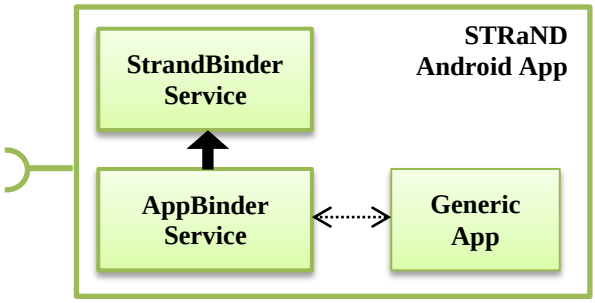


Figure 7. STRaND Application Wrapper

The first step in this process is to transform the app in a bound service. To provide binding for a service, it is required to implement the *onBind()* callback method returning an IBinder object that defines the programming interface that clients can use to interact with the service. In this case the client is going to be the MCA. The idea is therefore to wrap the generic Android app in a bound service that once created is able to instantiate the application and to terminate it once the *onDestroy()* method is called. The STRaND framework offers a wrapper described in Fig. 7 formed by an abstract class the StrandBinderService that implements the bound service. This class implements the messaging handler defined by the framework protocol and initializes the service with a specific working directory and log file. To connect this component to the actual application, the StrandBinderService need to be extended. As it is an abstract class, extending it forces the implementations of a number of methods allowing the app to communicate with the MCA. Eventually, the only new piece of software required to connect a generic android application to the framework, is this class, here called AppBinderService.

5. Space Qualification Experiments

The two primary tests which continued from our previous work has focused on thermal, thermal vacuum, and radiation tests. The smart-phone have already been tested under a vacuum and no adverse behaviours were found.

5.1. Thermal Tests

The thermal tests were carried out at SSTL to assess if the smart-phone had any erroneous operations when at sub-zero temperatures. Of two tests, two phones experienced thermal cycles similar to that in Low Earth orbit, as shown in Fig. 8. When calling the battery readings, a calibration offset is required for sub zero-temperatures:

$$If < 0, Temp = Temp - 65535 \quad (1)$$

There are likely to be other offsets and calibration tools built in to the smart-phone, but are so well integrated that we can neither call them in software or hack them in hardware. This can be shown in Fig. 9 where odd voltage readings are taken from the battery at sub-zero conditions. The battery temperature sensor did not have any further problems and was deemed suitable for flight.

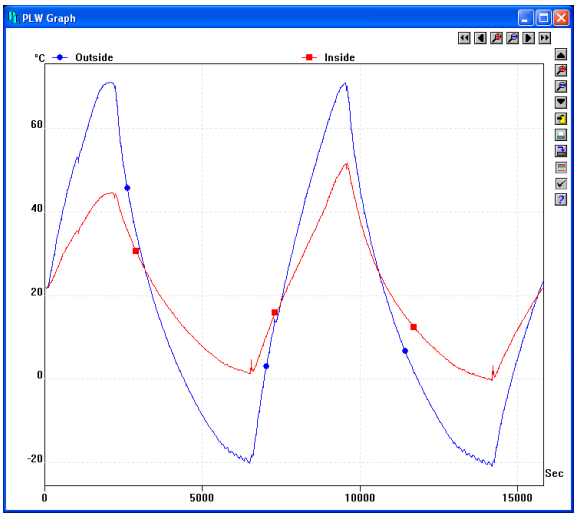


Figure 8. Thermal Chamber Results

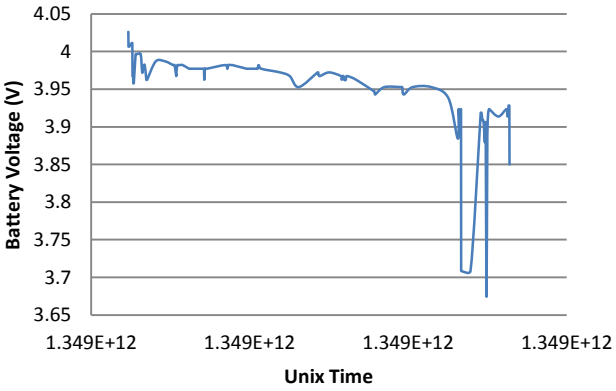


Figure 9. Battery Voltage under Thermal Cycle

5.2. Thermal Vacuum (TVAC) Tests

One of the last stages of AIT is to perform long duration thermal vacuum and vibration tests at full system level. The thermal vacuum tests performed at SSTL in Sevenoaks, UK for 3 full day prolonged tests (12 hours cold, 12 hours hot), the STRaND-1 satellite and smart-phone payload was successfully switched on and off, even with cold starts.

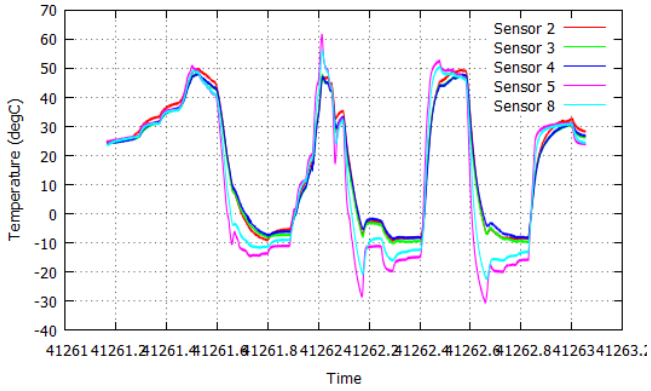


Figure 10. Thermal Vacuum Sensor Readings

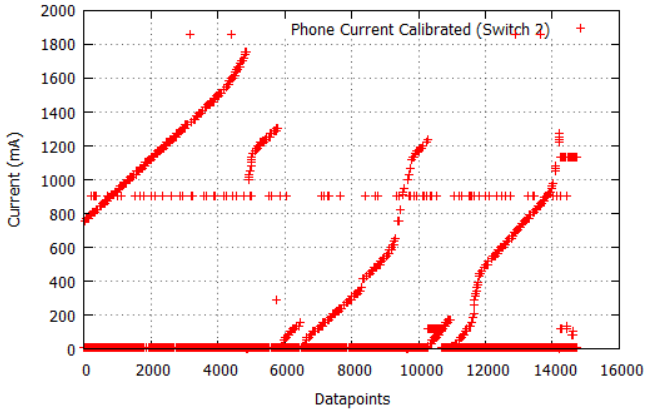


Figure 11. Smart-phone current draw under TVAC

From Fig. 10, temperature sensors 2-4 are on STRaND-1 whilst sensors 6 and 8 were on the chamber wall. From Figs. 10 and 11, the current draw from phone operations increased at lower temperatures and, more importantly, did not compromise any satellite operations.

5.3. Total Ionising Dose (TID) Tests

The ionizing radiation experiments were carried out on 3 sets of 3 Google Nexus-One smart-phones. All experiments were carried out using a Cobalt-60 gamma source: the first two at National Physical Laboratory at Teddington, UK, and the third at Synergy Health Ltd in Swindon, UK. The aim of these tests was to estimate performance and susceptibility of total ionizing dose (TID) damage to the smart-phones at system level. The first experiment was carried out to 25 krads(Si) and results were inconclusive. Both the flash memory card and Li-Ion battery operated normally immediately after the experiment and after a further 48 hours. Three (3) Nexus-One smart-phones undertook 1 krad(Si) over 8 hours on 19 October 2011. This equates to a dose rate of 3.472 mrad/s. During this experiment, the Android debugger (adb) and Logcat were in operation along with a telemetry application (available from [23]). No effects were observed on any of the devices and they continued to act as software development phones for over 6 months.

A more classical TID test, which tested the same three devices for 15.4 krads over 10 hours took place on 25 April 2012. It was the aim to test to destruction. The following sections show results from the power, battery, and sensor results from the three devices under test (DUTs). They are labeled according to the full smart-phone serial numbers. HTC1743 operates a non-stock ROM called Cyanogen whilst HTC1764 and HTC1912 operate stock Android ROMs (V 2.3.4).

Taken from Logcat, the battery charge information is logged over the experiment and shown in Fig. 11. The results show that the fast ‘poke’ charge draws large power and resulted in failures for 2 of the phones. The trickle charging of HTC 1764 appears to allow for stable operation.

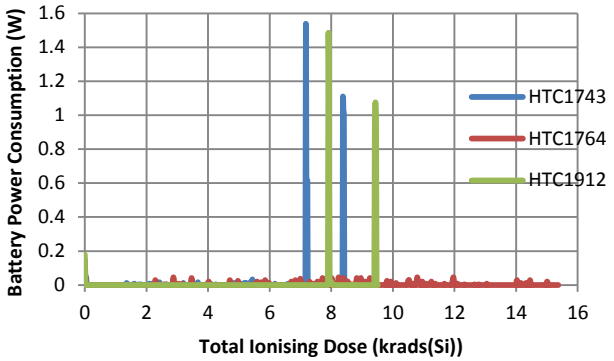


Figure 12. Battery Charge Log

As both the battery temperature was also logged at the same time, Figure 13 correlates directly with the battery power consumption logs in Figure 12. This suggests a catastrophic failure in the power subsystem circuitry leads to an elevated battery temperature reading/response correlated with the increased current flow readings from Figure 12.

Post annealing for 48 hours did not recover any of the smart-phones. The Flash memory devices and batteries however appear to be unaffected from this test.

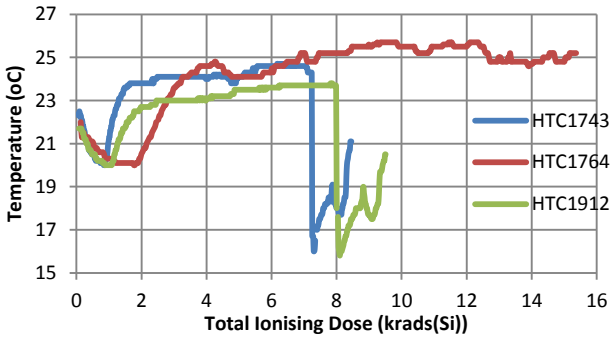


Figure 13. Temperature during TID Tests

Table 1 shows the timing, dose rates, and behaviours of DUTs from 11.00:19 to 21.01 in Swindon, UK.

Table 1. Dose Rates & Behaviours

Phone Serial	**Time & Final Dose Rate**	**Behavior**
HTC1743	16:28 = 8.470 krad + 1 krad from Test II = **9.470 krad**	- Logcat connection failure - Telnet MCA connection failure
HTC1764	17:12 = 9.548 krad + 1 krad from Test II = **10.548 krad**	- Logcat connection failure - Telnet MCA connection failure
HTC1912	21:01 = 15.4 krad + 1 krad from Test II = **16.4 krad**	- No failure mode observed throughout the test till 21:03. - After pressing the power button, powering the screen, Logcat & Telnet MCA connection failures were observed

Preliminary Software Results: During the experiment, the MCA requests telemetry (tlm) every 5 seconds. Scripts were utilized every 15 minutes to collect telemetry files from the Nexus-Ones individually using FTP over USB 2.0. Figure 14 shows the time responses between logged MCA TM requests (which can be either 0 or 5 s). Large response times are noted from the two stock Android phones when FTP calls are made, but provided MCA TM responses at 0 or 5 s (standard deviation of 2.803 x 10^{-5}). The non-stock Android phone, however, replied to MCA TM requests erratically but FTP downloads were much timelier.

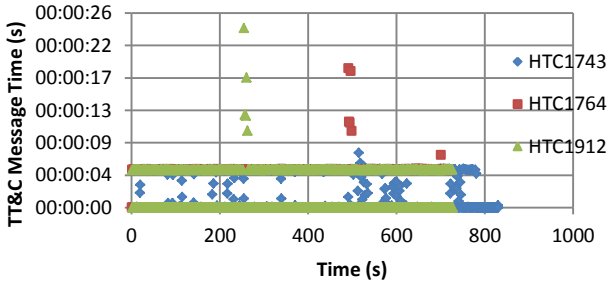


Figure 14. MCA Telemetry Request Timing Log

Figure 15 shows the memory as shown from Logcat's GC_* logs. The total memory for the Android ecosystem, MCA & Telemetry services and GUI frontend did not exceed 6 MB.

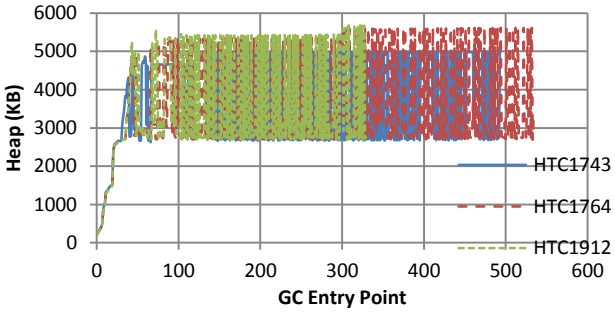


Figure 15. Memory Usage of MCA, Telemetry, and GUI Services for STRaND-1

Note here that the non-stock Android system operates in less memory and appears to be more stable too. Further profiling is required to confirm our method/function costs and on the Facebook apps which will run on the smart-phone [24].

6. SUMMARY

This paper aimed to give the reader a thorough knowledge of the STRaND-1 smart-phone payload and how it operates within the CubeSat nanosatellite. The extensive software toolchains are proven to be reliable and portable to both the µC-Linux on the Wi9C and the Android system alike and can act as a guide for those that want to continue the work further. The STRaND-1 ecosystem for reliably controlling typical Android 'apps' is presented and has been found to provide a new object-to-object IPC framework towards utilizing a standard Android system for flight – available online in S-Android, a Google Code project. The thermal, thermal vacuum and radiation tests have uncovered that the

high levels of integration in these commercial devices are resilient to thermal conditions and to typical levels of dose in comparison to other COTS devices. Further single-event tests using charge particles in the 50-100 MeV energy band would provide better knowledge of how these devices can be used and replace older or obsolete processors currently in use.

ACKNOWLEDGEMENTS

The main author would like to thanks all the STRaND-1 team for their time and contributions towards the paper – without their support, none of the work could have been achieved. The authors also wish to additionally thank NPL & Synergy Health in the UK for the use of their TID Testing facilities.

REFERENCES

[1] Underwood, C., Richardson, G., Savignol, J (2001) "SNAP-1: A low cost modular COTS-based nanosatellite - Design, construction, launch and early operations phase" *Fifteenth AIAA/USU Conference on Small Satellites, Logan, Utah*

[2] CubeSense Manual, Website, http://www.isispace.nl/brochures/CubeSense%20Brochure_CSS.pdf (last accessed 28.10.2012)

[3] Bridges, C.P., Kenyon, S., Underwood, C., Sweeting M.N. (2011) "STRaND: Surrey Training Research and Nanosatellite Demonstrator" *1st IAA Conference on University Satellite Mission and CubeSat Workshop January 24-29, 2011 Rome, Italy*

[4] Kenyon S, Bridges CP, Liddle D, Dyer R, Parsons J, Feltham D, Taylor R, Mellor D, Schofield A, Linehan R. (2011)'STRaND-1: Use of a $500 Smartphone as the Central Avionics of a Nanosatellite'. International Astronautical Federation*Proceedings of the 2nd International Astronautical Congress 2011, (IAC '11)*, Cape Town, South Africa: 62nd International Astronautical Congress 2011, (IAC '11)

[5] IAC11 NASA Boshuizen CR, Marshall W, Bridges CP, Kenyon S, Klupar PD. (2011) 'Learning to Follow: Embracing Commercial Technologies and Open Source for Space Missions'. International Astronautical Federation *Proceedings of the 62nd International Astronautical Congress 2011, (IAC '11)*, Cape Town, South Africa: 62nd International Astronautical Congress 2011, (IAC '11) (IAC-11-D4.2.5)

[6] Digi International Inc., ConnectCore™ 9C/Wi-9C Hardware Reference, Document, http://ftp1.digi.com/support/documentation/90000789_D.pdf (last accessed 06.09.2011)

[7] Google Inc., Nexus One – Google Phone Gallery, Website, http://www.google.com/phone/detail/nexus-one (last accessed 06.09.2011)

[8] Gomspace, NanoMind A702/A712 Datasheet, GS-DS-NM702-2.1

[9] telnet Website, http://www.telnet.org/ (last accessed 28.10.2012)

[10] Pexpect - a Pure Python Expect-like module, Website, http://www.noah.org/wiki/pexpect (last accessed 28.10.2012)

[11] Android Debug Bridge, Website, http://developer.android.com/tools/help/adb.html (last accessed 28.10.2012)

[12] Busybox, Website, http://www.busybox.net/ (last accessed 28.10.2012)

[13] Utelnetd Sourceforge Source Website, Website, http://sourceforge.net/projects/utelnetd/ (last accessed 28.10.2012)

[14] Dropbear SSH server and client, Website, http://matt.ucc.asn.au/dropbear/dropbear.html (last accessed 28.10.2012)

[15] VSFTPD, Website, https://security.appspot.com/vsftpd.html (last accessed 28.10.2012)

[16] PureFTPD, Website, http://www.pureftpd.org/project/pure-ftpd (last accessed 28.10.2012)

[17] OpenSSH Suite, Website, http://www.openssh.org/ (last accessed 28.10.2012)

[18] QuickSSHd Google Play App Download, Website, https://play.google.com/store/apps/details?id=com.teslacoilsw.quicksshd&hl=en (last accessed 28.10.2012)

[19] Curl, Website, http://curl.haxx.se/ (last accessed 28.10.2012)

[20] S-Android, Space Android: Android for Space Subsystems, http://code.google.com/p/s-android (last accessed 28.10.2012)

[21] STRaND-1 Facebook Page, Website, https://www.facebook.com/nanosats (last accessed 28.10.2012)

BIOGRAPHIES

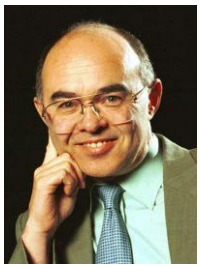
Dr Chris Bridges *received his BEng in Electronic Engineering from the University of Greenwich in 2005 and his PhD in Agent Computing for Distributed Satellite Systems in 2009 from Surrey Space Centre, University of Surrey. He is currently leads the On-Board Data Handling (OBDH) Group and is the SSC Lead for the STRaND-1 Programme with Shaun Kenyon (SSTL). He researches real-time embedded systems, agent computing, Java processing, multi-core processing in FPGAs, and astrodynamics software and hardware computing methods towards real spaceflight payloads together with colleagues in SSC & SSTL.*

Brian Yeomans received his Masters in Space Technology and Planetary Exploration from the University of Surrey in 2009 and is currently studying for a PhD at the Surrey Space Centre. He is researching the modeling of leg / terrain interaction for walking planetary exploration vehicles under work co-funded by the European Space Agency. He has extensive experience of the design and implementation of Embedded Linux systems, particularly for mechatronics and robotics applications.

Claudio Iacopino received a M.S. in software engineering in 2007 from the Universita' Politecnica delle Marche, Ancona. Since then he has been working for the European Space Agency in the Earth Observation and Operations field, designing advanced technologies to be included in the ground segment and in the Operations infrastructure. He is currently a PhD student at the Surrey Space Centre, University of Surrey. He is researching on automatic mission planning & scheduling systems for distributed missions, using natural inspired approaches. The project is co-funded by the Surrey Satellite Technology Ltd (SSTL) and by the European Space Operation Centre (ESOC).

Andrew Schofield received a BSc in Computer Science from Liverpool University in 1987 and an MA in Computer Animation and Visualisation from Bournemouth University in 1992. He has been working with computers throughout his career in education, health, entertainment and Space, with a particular focus on ICT in developing countries. He has been with Surrey Satellite Technology Ltd since 2006 in the role of Groundsegments Team Leader.

Shaun Kenyon is a Mission Concepts Engineer at SSTL specialising in feasibility analysis for future nanosatellite and microsatellite concepts, and enabling technologies. He graduated with an MEng in Aerospace Engineering from the University of Southampton in 2006, and has since worked on various projects including Space Situational Awareness, ORS, and Galileo.



Professor Sir Martin N. Sweeting, B.Sc.Hons., PhD (Surrey), FRS, FREng., FIEE, FRAeS, FBIS, SMIEEE, SMAIAA, MBIM, MIAA has pioneered the concept of advanced microsatellites utilizing modern commercial-offthe-shelf (COTS) devices for 'affordable access to space.' After completing BSc & PhD degrees at the University of Surrey, in 1985 he formed a spin-off University company (SSTL - Surrey Satellite Technology Ltd) which has designed, built, launched and operates in orbit a total of 26 nano, micro, and minisatellites - making SSTL the world's leading microsatellite company. As Chief Executive of SSTL, he has been responsible for the leadership and management of the Company which by 2006 has grown to 210 commercial staff and achieved a total export sales of over £110M. Sir Martin is also Director of the Surrey Space Centre, leading a team of 60 faculty and doctoral researchers investigating advanced small satellite concepts and techniques. Sir Martin was knighted by HM The Queen in the 2002 British New Year Honours for services to the small satellite industry.