BlueHoc: Bluetooth Ad-Hoc Network Android Distributed Computing

G. Hinojos, C. Tade, S. Park, D. Shires, and D. Bruno
Computational Sciences Division
U. S. Army Research Laboratory
Aberdeen Proving Ground, MD, USA

Abstract—*Mobile devices are ubiquitous in everyday life and are becoming valuable devices for today's Soldiers as part of a larger battlefield network. Due to the open nature of the development platform, Android was recently selected to be a supported operating system within this evolving and maturing technology delivery paradigm. The Army's networks must operate in often hostile environments and are mobile and* ad hoc *in nature; thus often rendering communication links tenuous at best. Common, however, on handheld device are low-power network capabilities such as WiFi and Bluetooth. This work analyzes the use of Bluetooth as a low-power network protocol for coupling handhelds operating in a deployed setting. By aggregating the capabilities of distributed handhelds through Bluetooth, task and data parallelism can be achieved, thus providing potentially faster solutions and reduced battery drain. This paper discusses the performance of a preliminary scalable boss-worker paradigm known as "BlueHoc" in the context of a simplified test case with proposed extensions that will provide greater capabilities to Soldiers operating at the tactical edge.*

Keywords: Mobile ad hoc networks, Bluetooth, distributed computing, Android

1. Introduction

Computer networks are common in modern society and span a wide range of wired (DSL, Ethernet) and wireless (3G, 4G, WiFi) services. Military missions, particularly those of the Army, do not have the luxury of a fixed infrastructure with high capacity and low latency. Working in hostile environments is common, and mobile *ad hoc* networks (MANETs) will form the backbone of the deployed forces. These networks can be hierarchical and complex with widely varying data rates at all levels.

Typically the last hop to an edge node, such as a handheld device, is the most costly of all. Cloud-based services in well-covered network areas have extended capabilities to streaming data rather than just guaranteed access to data. Rapid processing of requests coming from handhelds, such as that offered by Apple's Siri service, is handled by offloading from handhelds to high capacity servers. Planning is underway in Army MANET delivery to provide the required bandwidth to the deployed network, but processing and data delivery at deployed edge nodes will remain a critical need.

Of interest then is how to exploit capabilities inherent in these devices should access to existing networks fail. Within each device is preloaded applications and data. When paired with other devices in close proximity, what new capabilities can be afforded by the higher capacity if these devices are coupled and aggregated? Synchronizing the computing power in a way that limits overall battery drain is very important to missions conducted on the battlefield.

This project discusses BlueHoc, a system that enables distributed computation across mobile devices communicating wirelessly via Bluetooth. Mobile devices have become highly utilized in recent years; due to the high demand, advances in mobile technology occur every day. Mobile devices are perhaps the most pervasive computing devices available today. In the final quarter of 2010, smarthphone sales surpassed global PC sales for the first time [1]. Mobile cellular subscriptions worldwide are estimated to be around 6 billion as of 2011 [2]. The vast number of available mobile devices presents computational resources that can be utilized to solve a diversity of parallelizable computations. Mobile device resources can be combined and leveraged to create a distributed infrastructure that is able to perform parallel computing for both mobile Soldiers and stationary operators in a Tactical Operations Center (TOC).

By aggregating computing power, important questions, such as "what computing capacity can I achieve from many Army connected devices?" and "what new capabilities can they bring to the Army operational domain?", can be answered. These questions are nontrivial and extremely valuable for the Army as it is not always feasible to build a single node High Performance Computing (HPC) system in the field or ensure its connectivity at all times. Furthermore, traditional cloud-based services will not always be available at the tactical edge where Soldiers operate. High throughput networks will not be available to off-load computing requests, and methods to overcome this limitation are only beginning to be explored [3]. This project attempts to bring HPC closer to the Soldiers and make it possible to build a HPC system from resources that are available and underutilized.

In the following, Section 2 summarizes related work in distributed computing using Bluetooth. Section 3 describes the operational and testing environments that were selected for this architecture and implementation. Preliminary results of the performance of the system are given in Section 4. Finally, conclusions and possible future extensions are briefly discussed in Section 5.

2. Related Work

The use of mobile phones to form small clusters of shared resources has not been well researched. Proposed architectures, built on the Bluetooth standard, would enable small mobile computers such as those found in robotics to communicate. The DynaMP architecture achieves scalability to larger networks through the formation of scatternets; larger networks formed by dedicating one node per subnet to communicate with another link node in another subnet [4]. The BlueHoc design roughly mirrors the architecture described in DynaMP and attempts to test the design in an actual hardware and software system. Issues with the communication protocols suggested in DynaMP are identified and differing methods are employed in BlueHoc to perform actual communication between nodes.

Gartrell et al. describe another architecture for Bluetooth device communication known as BlueHydra [5]. BlueHydra proposes methods for remote method invocation and uses the Marge framework for remote device discovery. The architecture is evaluated in terms of the Java Wireless Toolkit emulation framework and hence does not use an actual hardware and software device pairing. BlueHoc leverages built-in Bluetooth chat clients to implement the network and handle device communication under an Android operating system. This allows for testing of performance on physical ARM-based processors that would be commonly found in handhelds. Rather than evaluate the system in emulation, data rate tests of the Bluetooth network were performed using PandaBoard platforms.

3. Operational and Testing Environment

Bluetooth technology provides for dynamically linked mobile devices that can exploit the potential of wireless networks used in parallel computing. Bluetooth provides a low power transmission mechanism that is commonly embedded in most mobile devices nowadays. The widespread nature of Bluetooth makes it an ideal technology for building *ad hoc* networks to create a multiprocessor distributed infrastructure from mobile devices. Bluetooth offers a great means of wireless communication for mobile devices: it offers low power consumption, low cost, robustness, and *ad hoc* networking protocol capabilities [6]. Bluetooth v4.0 was the most recent version of the standard as this research was being conducted and it significantly reduces energy consumption over prior versions [7].

The target platform for this project is Android. Android is an open source project and is the most popular mobile platform in the market; 68.1% of mobile devices shipped during the second quarter of 2012 use Android as their operating system [8]. The prevalent and sophisticated nature of Android allows for the creation of countless applications with endless possibilities. Additionally, the Army is moving towards utilizing Android as their main operating system for mobile devices [9].

3.1 Android Parallel Computing Support

In the world of HPC, message passing (and the Message Passing Interface [MPI]) is a widely used and tested paradigm for parallelism. Following the Single Program-Multiple Data (SPMD) paradigm, it can be useful for both task and data parallelism. It has also been shown to be effective in distributed memory systems [10]. MPI is a library of routines for portable message passing programs in parallel systems and thus the project's initial investigation evaluated MPI support for Android. Due to the fact that Android deviates from the Linux kernel, Android does not fully support common Linux applications and libraries. Consequently, MPI was not successfully ported to Android. As a result, an alternate approach to MPI was developed using the radio frequency communication (RFCOMM) protocol embedded in Bluetooth and fully supported by Android.

The difficulty of porting Linux applications to Android lies in the two significant differences between the operating systems. First, the Android operating system does not utilize the standard Linux kernel. For example, Google chose to branch off from the GNU kernel to create their own kernel which gave them the flexibility to make changes that they felt were necessary to increase efficiency on a low power device. The Android kernel replaces the GNU libc with Bionic, a lightweight libc library developed by Google to target low power devices. The first of many differences between the two libraries is that Bionic does not support the full C/C++ standard. It does not handle, throw, or pass C++ exceptions. Since Android's primary programming language is Java, Google made the decision that all exceptions would be handled at the Java run-time level. Additionally, Bionic does not have the C++ Standard Template Library (STL). While missing a few C++ libraries may not inhibit the porting of many Linux libraries, additional differences between Android and Linux certainly increases the difficulty.

The second difference between the Linux and Android operating system is the degree to which they have implemented additional libraries. When Google has a need for certain functionality in the Android operating system that another Linux library already provides, they choose, like most programmers, to utilize the tried and tested Linux library. That being said, Google forks their own version, just like their kernel, and may only choose to support a couple of functions in that library while leaving the others undefined or unimplemented. In particular, the libpthread library utilized by the Dalvik JVM has been stripped of a few functions required by many of the libraries. Most of the pthread library and functionality are still there, but it is missing functions like pthread_cancel(); Google decided not to support pthread_cancel() because doing this would involve making the C library significantly larger for very little benefit [11]. While many may argue that pthread_cancel() may be required in certain scenarios and cannot or should not be replaced by other pthread calls, ultimately, Google has the choice of whether or not certain functionality is included in its forked libraries. As such, the developer usually has to build all required dependencies themselves should they wish to port a Linux library to Android.

3.2 Bluetooth Technology Networks

The context of field operations assumes a zero network infrastructure where dependence shifts to *ad hoc* networks. An example MANET at the tactical edge could be a collection of wireless mobile devices that can configure to form a network without any preexisting infrastructure. MANETs are robust, dynamic networks that can be rapidly deployed and reconfigured, making them ideal for military applications. Since they are extremely important parameters, the Bluetooth standard is adopted to address the challenges related to power consumption and battery life. Bluetooth operates within the 2.4 GHz ISM band and hops over 79 channels (2 through 80) at a rate of 1600 hops per second using the Time Division Duplex (TDD) scheme [12].

The BlueHoc system architecture is boss-worker, where the boss can connect with a maximum of seven workers in a piconet. A piconet is an *ad hoc* network connecting wireless devices using the Bluetooth protocol. Because piconets have a 3-bit address space notation, the maximum number of devices is limited to $2^3 = 8$, or eight devices composed of one boss and seven workers. To expand the physical size of the piconet network, two or more piconets can share a common Bluetooth device acting as a bridge between piconets to form a larger network known as scatternet. A scatternet is formed in an *ad hoc* fashion when two or more independent piconets overlap where a member of one piconet, either a boss or a worker, elects to participate in a scatternet. In a scatternet, a Bluetooth device can participate as a worker in several piconets, but can only be a boss in one piconet [13]. Figure 1 depicts an example configuration of a scatternet consisting of three piconets.

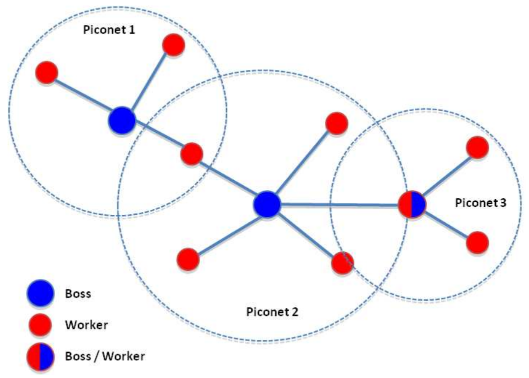

Fig. 1: A scatternet configuration composed of three piconets.

The current architecture of BlueHoc is static, meaning that the workers are required to join the network and remain connected throughout the work interval. For this preliminary implementation, the boss waits for the workers to connect to the network. After all the workers have completed the join process, the boss is able to issue job requests. Job requests are distributed among the workers as tasks by the boss and the computed results are sent back by the workers to the boss for the final result calculation. More elaborate protocols for scatter-gather-broadcast could also be substituted into this basic communication configuration. Figure 2 illustrates the data flow of job requests and computed results of the BlueHoc system architecture.

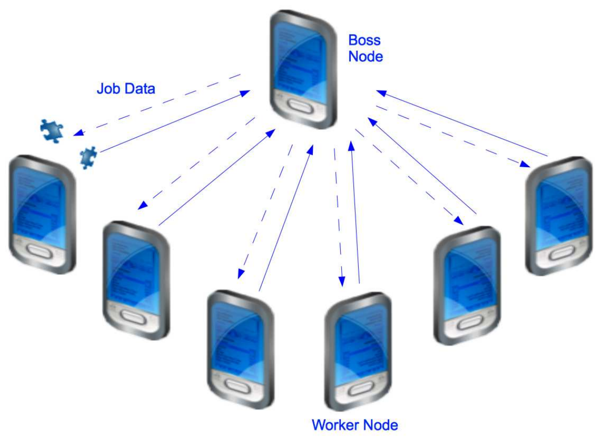

Fig. 2: BlueHoc architecture data exchange.

3.3 Network Performance

Latency and throughput tests are executed within an Android application developed for the project. BlueHoc is built upon an existing Bluetooth chat client provided as an example for Bluetooth connectivity by the Android SDK. The application delivers a very basic user interface (UI) that provides the user with a text entry box and button to send streams of text between boss and worker devices. There is an option menu that allows for device connection and enabling device discovery as well as a browser to select from files to send. Device names are added and removed from a "connected devices" list as each device enters/exits the network. A series of performance tests were conducted to determine the overall speed of transmission throughput and latency of the network. The tests were performed between two PandaBoards in close proximity running BlueHoc.

The ping utility (l2ping for Bluetooth devices) was not functional for the Ice Cream Sandwich Android OS build for PandaBoard. Therefore, the latency tests were conducted programmatically. The latency was determined by transmitting a small stream of data (44 bytes) and recording the round trip time (RTT). The clock times were taken from a single device to avoid synchronization between device clocks. The latency was found to be about 37.8 ms taken from an average of 500 recorded RTTs. As seen in Figure 3 the latency tends to stay between the 30 ms to 50 ms range. Values were recorded periodically throughout a 30 minute time window.

Next, the throughput tests were conducted for this Bluetooth network setup. An increasing range of file sizes was transferred via data stream buffers over open Bluetooth sockets between two PandaBoards. The process was repeated

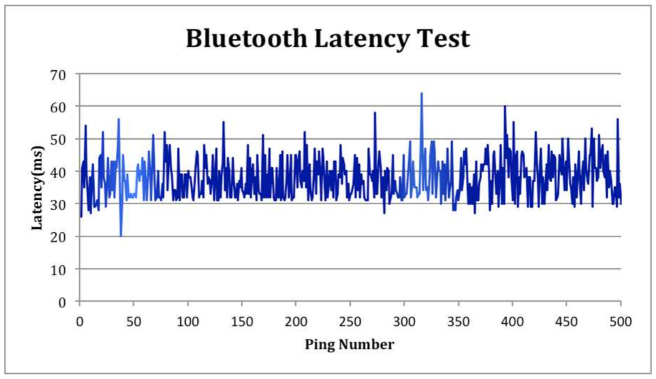

Fig. 3: PandaBoard Bluetooth latency test results.

for multiple iterations to develop an average transfer time. The ratio of file sizes to transmission time was recorded and plotted. As seen in Figure 4, the throughput is very low for less-than-one kilobyte of data. This speed steadily increases until the file size exceeds five kilobytes at which point transfer rate levels off at around 0.9 mbps. Considering the bandwidth of the Bluetooth module on the PandaBoard ES is rated at 2.1 mbps, the results suggest an achieved throughput of roughly half the theoretical data rate. Performance reduction can be justified by the application and network overhead (e.g. broadcast traffic, packet collisions, routing protocols, OS jitter).

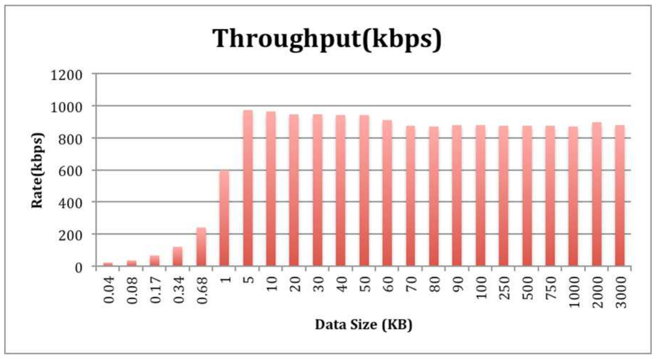

Fig. 4: PandaBoard Bluetooth throughput test results.

4. Application and Performance

The Monte Carlo method for π estimation served as a experimental application for distributed computing with Android devices. Leveraging Bluetooth wireless technology to establish a low power *ad hoc* network, multiple mobile systems can collaborate in performing a collective computation. The method used to estimate π followed the implementation of the popular random darts method [14]. This method allows for an approximation of π to be calculated by throwing darts randomly at a hypothetical dart board. Imagine a unit circle circumscribed by a square. By randomly throwing darts, hits inside the circle and square will be proportional to the respective area of each part, which can be written as

$$\frac{ndc}{nds} = \frac{ac}{as} = \frac{\pi r^2}{4r^2}, \tag{1}$$

where ndc is the number of darts in the circle, nds is the number of darts in the square, ac is the area of the circle, as is the area of the square, and r is the radius of the circle.

After substituting the number of darts in the square with the total number of throws, solving for π leads to the following equation:

$$\pi = 4 \times \frac{ndc}{nt}, \tag{2}$$

where nt is the number of dart throws. For this Monte Carlo method, the value of π becomes more precise as the iteration count increases.

A simple block scheduling algorithm handled the workload distribution across the multiple devices. The total number of iterations is evenly divided by the number of available devices for computation. For the cases where the number of devices fails to evenly divide the number of iterations, the ceiling value of the division is issued to workers. Each device is then initialized to compute their assigned number of iterations for the problem. At this beginning stage of Android distributed computing evaluation, the scheduling technique ignores differences in performance characteristics of a heterogeneous network of mobile devices. For example, given twenty million iterations and five worker devices, each device would compute four million iterations individually. In the current implementation, the designated boss node does not perform any dart throws, but gathers the results from the connected nodes and performs the final calculation from collected data.

Table 1: Specifications of Android devices.

Android Device	Processor	Android OS version
PandaBoard ES	Cortex-A9 1.2 GHz	Ice Cream Sandwich
Nexus 7	Tegra 3 1.3 GHz	Jelly Bean
Samsung Galaxy SII	Cortex-A9 1.2 GHz	Ice Cream Sandwich
Asus Transformer	Tegra 3 1.2 GHz	Jelly Bean
Motorola Xoom	Tegra 2 1 GHZ	Honeycomb

The π application was analyzed on five different Android platforms. Details regarding hardware specification and operating system setup are organized in Table 1. Non-distributed, base performance measurements of a single device for different Android devices are summarized in Figure 5. Regarding the unexpected result of the Samsung Galaxy SII, running background user applications adversely affected its execution time (being an actively utilized smartphone). Consequently, compared to the other Android devices under examination, the Galaxy SII had a multitude of user applications installed and loaded, taking a noticeable toll on the algorithm's performance.

The experimental test setup analyzed both homogeneous and heterogeneous Bluetooth device networks. For this exercise with block scheduling, the results obtained for uniform device networks outperformed the mixed Android

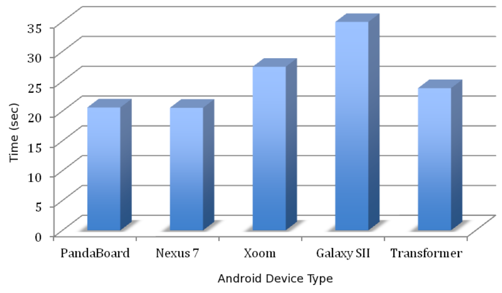

Fig. 5: Single device execution times for various Android platforms.

device network since the workload distribution was optimal. Recorded execution times for PandaBoard networks are presented in Table 2. To test a non-uniform Android devices network, a heterogeneous network was formed by using PandaBoard, Nexus 7, Samsung Galaxy, and Asus Transformer. This simulates a scenario where Soldiers have different types of mobile devices with different characteristics at their disposal. Table 3 provides the execution times for a Bluetooth network setup composed of different Android devices as the number of iterations is increased to 10^8. A graphical representation of the performance measurements achieved is presented in Figure 6.

For the π estimation algorithm, parallel computing via work distribution across multiple Android devices unequivocally reduces overall time to solution. As expected, a network with a homogeneous makeup of devices shows superior scalability as the overall time to result is bounded by the slowest device (and hence inefficiencies for processors that are spinning idle). Regardless of iterations or network type, leveraging four Android devices for this computationally intensive task decreased execution time to less than one third of its original time in the worst case. The execution time reduction exhibits the potential advantage of distributed computing with Bluetooth networked Android devices.

Table 2: Execution times for homogeneous networks consisting of PandaBoards.

	PandaBoards (sec)			
Iterations (millions)	1	2	3	4
10	8.11	4.15	3.09	2.12
25	20.21	10.16	7.39	5.20
50	40.08	20.14	15.18	10.33
75	65.00	30.52	22.59	15.47
100	84.34	41.74	33.00	20.83

5. Conclusion and Future Work

The capacity and capabilities of handheld devices continue to improve with processing power and the creativity of application developers. One of the biggest advances of these devices is how they allow for geospatial awareness; the user's location can bring a wealth of information and be an important filter to the vast number of queries these devices process daily. By subscribing to the larger cloud, these handhelds also become important sensors in the field.

Table 3: Execution times for heterogeneous networks consisting of a mixture of Android devices.

Iterations (millions)	PandaBoard (sec)	PandaBoard Nexus 7 (sec)	PandaBoard Nexus 7 Asus Trans (sec)	PandaBoard Nexus 7 Asus Trans Galaxy SII (sec)
10	8.11	4.75	3.19	2.49
25	20.21	11.16	7.49	6.22
50	40.08	22.14	17.28	12.06
75	65.00	33.52	24.39	18.30
100	84.34	43.74	34.05	24.09

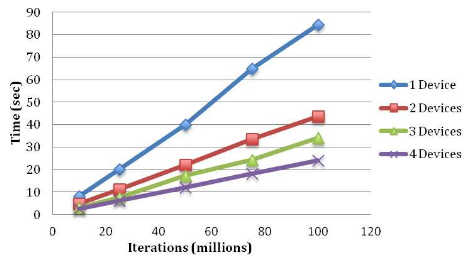

Fig. 6: Mixed Android devices networks execution times.

From reporting weather, restaurant reviews, or traffic speeds, important and often temporal data can be broadcast to a wider user community.

However, making data and processing available when the network connectivity of the cloud is not guaranteed is only beginning to be investigated. By pooling the resources of deployed mobile devices, one can envision scenarios where data can be preloaded and distributed amongst devices where the internal storage of one device would be insufficient. Additionally, these devices can be brokered and shared, thus providing either a faster time to solution, or a shared workload to conserve battery life, or some combination of the two. All of this can be accomplished using common communication protocols, such as Bluetooth on the Android-based devices described in this paper. This framework provides an important capability for a small group of friendly forces geospatially co-located, and has been evidenced in the small test study described in this paper.

As with other past research being conducted on mobile networks using Bluetooth, BlueHoc is at its infancy. Further advances are being planned, such as improvements in scheduling to allow for device drop-out and drop-in along with better load balancing. Discovery protocols will need to incorporate host processor types and expected performance, possibly coupled with scheduling approaches such as guided self scheduling, to achieve optimal workload distribution.

6. Acknowledgments

This work was supported in part by the Army High Performance Computing Research Center (AHPCRC) cooperative agreement.

References

[1] "Industry first: Smartphones pass PCs in sales," http://tech.fortune.cnn.com/2011/02/07/idc-smartphone-shipment-numbers-passed-pc-in-q4-2010/, 2011.

[2] "Measuring the information society," http://www.itu.int/dms_pub/itu-d/opb/ind/D-IND-ICTOI-2012-SUM-PDF-E.pdf, 2012, international Telecommunication Union.

[3] D. Shires, B. Henz, S. Park, and J. Clarke, "Cloudlet seeding: Spatial deployment for high performance tactical clouds." Parallel and Distributed Processing Techniques and Applications, 2012.

[4] R. Shepherd, J. Story, and S. Mansoor, "Parallel computation in mobile systems using bluetooth scatternets and java," in *Parallel and Distributed Computing and Networks*, 2004, pp. 159–164.

[5] M. Gartrell, J. Kelly, and S. Razgulin, "Bluehydra: Distributed computing on mobile bluetooth-enabled devices," University of Colorado at Boulder, Tech. Rep., 2008.

[6] "About the bluetooth sig: Overview," http://www.bluetooth.org/About/bluetooth_sig.htm.

[7] "Specification of the bluetooth system," Bluetooth Special Interest Group, Tech. Rep., June 2010, covered Core Package version: 4.0.

[8] D. Reisinger, "Android smartphone share quadruples iOS in Q2," http://news.cnet.com/8301-1035_3-57488926-94/android-smartphone-share-quadruples-ios-in-q2/, August 2012.

[9] E. Montalbano, "Army selects android for mobile battlefield network," http://www.informationweek.com/government/mobile/army-selects-android-for-mobile-battlefi/229402123, April 2011.

[10] P. S. Pacheco, *Parallel programming with MPI*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[11] "Bionic C library overview," http://github.com/android/platform_bionic/blob/master/libc/docs/OVERVIEW.TXT, 2009.

[12] T. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[13] "Scatternet - part 1," Ericsson, Tech. Rep., June 2004, baseband vs. Host Stack Implementation.

[14] "Monte carlo methods," in *Statistics Applied to Clinical Trials*, T. Cleophas, A. Zwinderman, T. Cleophas, and E. Cleophas, Eds. Springer Netherlands, 2009, pp. 479–486. [Online]. Available: http://dx.doi.org/10.1007/978-1-4020-9523-8_41