

Using Remotely Executing Software via a Mobile Device

Vipul Delwadia

Stuart Marshall

Ian Welch

School of Mathematics, Statistics and Computer Science
Victoria University of Wellington,
PO Box 600, Wellington 6140,
Email: (vipul | stuart | ian)mcs.vuw.ac.nz

Abstract

There are scenarios in mobile computing that may benefit from separating presentation from computation. Traditionally this separation can be achieved via tools such as VNC. However such factors as network latency and additional communication overhead can slow down the presentation of a remotely executing mobile application below acceptable performance levels, especially for domains like gaming where responses may need to appear to be instantaneous. We present RemoteMe, an architecture and Java-based prototype for mobile-client / server communication that only requires a very thin mobile client. We hypothesise that RemoteMe will support faster response times to user input than existing software solutions such as VNC. This paper presents a preliminary analysis of our first prototype, and experimentally compares it to an open-source mobile-based VNC system.

Keywords: Mobile Devices, Remote Gaming, Human Computer Interaction

1 Introduction

Mobile devices are now extremely commonplace in developed countries. Increasingly, mobile devices are being used for tasks other than human-to-human communication, and support tasks previously associated with more powerful desktop computers or servers. We propose a method for provisioning mobile devices with applications that are accessed through the mobile device while in fact executing remotely on a server connected over wifi.

While this is — on the surface — similar to existing client-server architectures, our method would not require an application-specific client on the mobile device. This lack of an application-specific client is reminiscent of how web-based applications can be accessed solely through a standard web browser. However, our method would also provide a greater ability for individual mobile devices to tailor the user interface to those devices' specific input/output mechanisms. This is useful because different mobile devices have different screen sizes, different keypads, and different button mappings / locations for controls in a given mobile application user interface.

In this paper we present our prototype and architecture for a system that will support a very-thin-client approach to remote software access via mo-

bile devices. Our subsequent analysis and experimentation with our prototype and related works will be framed by the particular requirement that the user should get “instantaneous” response to their actions. We define “instantaneous response” to be within 100ms (Nielsen 1993). The preliminary experiments we present at the end of this paper attempt to measure whether our prototype and other existing related tools can support this response time for various values of network connectivity and distance.

2 Motivation

There are a number of scenarios where we see such a method proving useful.

Firstly, while mobile device technology has advanced significantly in the past few years, they still have a relatively low computational power. This means that there are some tasks that may be beyond the ability of the mobile device to perform by itself, even though the device has sufficient screen real estate and input mechanisms to handle the necessary input and output. One possible solution to this is to divide interaction and computation between the mobile “client” and a more powerful server. An example of this is a user playing a computer game that requires significant and computationally expensive AI.

Secondly, mobile devices by their very nature open up opportunities in providing location-specific services (Hinze & Buchanan 2006). In this scenario, users could walk into a particular area (such as a museum) and access applications that the museum supplies to provide more information or interactive features. The user would not need to already have or install a client specific to the museum, and could interact with the applications as if they were native applications on their mobile.

Thirdly, there are situations where the owner of an application wishes to make the application available to users, but does not want to distribute (and potentially lose control of) the software directly. An example of this could be a user walking into a game store and evaluating the latest mobile game to hit the market by playing the full version on their device, without the store losing control of the product.

3 Related Work

In the original vision of thin client computing, such as the Java applet-based architecture (Gosling et al. 1996), the clients need only provide an execution platform and applications can be downloaded on demand and run locally. In fact, many mobile phone providers use exactly this technique to distribute mobile phone games and other applications. The downside of this technique is the complexity of the game that can be played is bounded by the capabilities of the mobile

phone that is executing it. Therefore this related work focuses on only providing the presentation layer on the phone. In particular, remote desktop software and video streaming software.

3.1 Remote Desktop Software

Remote desktop software allow a user of a less powerful machine to access their desktop that runs on a more powerful machine across a network. In our context, this would allow a user to run a game on a remote machine but use it as if it was running on their mobile device. There is a large number of remote desktop software applications available for a range of platforms including Windows, Linux, Mac OS X, embedded devices and mobile devices. However, most use variations of one of the following protocols for communication: Remote Desktop Protocol (RDP), Remote Frame Buffer (RFB) and X11.

RDP RDP (MSDN 2008) is the technology underlying Microsoft's remote desktop software for the Windows Server and Client operating systems. Once a connection is made between an RDP client and server, the server sends commands to render windows to the client, which is a mixture of drawing commands and images. User input via keyboard/mouse is sent back to the server from the client accordingly.

RFB RFB is the protocol that VNC (Virtual Network Computing) is built on top of. The RFB protocol operates in a client/server manner as well, but the server sends the display to the client as rectangular snapshots of the remote desktop (or frame buffer). The snapshots are simply bitmaps representing rectangular sections of the remote desktop. Instead of sending just the raw bitmaps, the rectangles can be encoded in a number of ways. Each specific encoding requires both client and server support, and one such encoding, supported by TightVNC (TightVNC 2008), is the *Tight* encoding with optional JPEG compression. Essentially, Tight performs compression on the rectangles before they are sent, reducing the amount of traffic generated.

X11 X11 is a client-server system, where the server is the application rendering the window, and the client is the application requesting to be rendered. When a window is to be rendered, the X11 client application sends the X11 server a request to create a window with an id. Further, when the client wishes to fill that window with content, it refers to it by id.

There have been at least two attempts at trying to reduce the amount of traffic required to use X11 across a network, namely NX and LBX.

NX NoMachine's NX technology (NoMachine 2008) utilises a number of different strategies for reducing the bandwidth of networked X sessions. One aspect is to compress the data sent with zlib (Greg Roelofs & Adler 2008). In addition to compression, NX selectively caches data received to effectively reduce the need to re-send data which is static in the short-term. NX is considered a worthy alternative to standard network based X11.

LBX Low Bandwidth X (LBX) (XFree86 Project, Inc 2008) is a proxy server (`libxproxy`) which caches often used data, such as window properties and font metrics. It also allows for compression of images and general stream data. `libxproxy` sat between the X11 clients and server, and didn't require changes to existing programs. Unfortunately, LBX provide small

performance enhancements over slow networks, and more recent versions of X11 system have been optimized such that LBX no longer gave any real benefits (Packard & Gettys 2003).

3.2 Video

Aside from the main remote desktop protocols discussed above, researchers are exploring the use of streaming video to remotely control a system. An example application of this are the model helicopters trialled by the Merseyside Police in Britain. An operator controls the helicopter and receives feedback on the helicopter's surroundings via CCTV cameras mounted on the helicopter itself (BBC 2007).

(Zhuang & Wang 2006) have developed an "IP-based real time video monitoring system with controllable platform". Their approach was to have a camera and microphone connected to an embedded system that compresses the incoming data and sends it to a server. When a client comes along, they request the stream from the server. Upon receiving the compressed data, the client decompresses and displays it. Cell phones, along with standard desktop machines, are supported as clients.

4 RemoteMe

In this section, we present the *RemoteMe* prototype and the underlying architecture.

4.1 Design

The RemoteMe architecture/prototype describes a set of connected components on both the client and the server. These components facilitate monitoring an application executing on the server-side, sending the required instructions to the mobile device for rendering, and receiving user input from the mobile device for processing by the server-side application.

Figure 1 shows our architecture, with specific Java-based technologies included. The architecture assumes a wifi network connection between the client and the server.

The server runs a mobile phone emulator system (such as the Sun Wireless Emulator) in which a mobile application is executed. Prior to execution, additional code (in the form of AspectJ aspects (Kiczales et al. 2001)) is woven into the mobile application source code. The additional code is responsible for monitoring the use of standard UI libraries within the targeted mobile application and is the same for all monitored applications, although the weaving itself would be application-dependent and automated. During execution on the server, the mobile application will ultimately make calls to the standard UI libraries provided by the *Mobile Information Device Profile* (MIDP) libraries. The aspects capture these events and transmit the method call information to the connected mobile device.

The client runs a small application (in this case on a JavaME virtual machine) that is completely independent of any particular server-side (JavaME-based) target application. The application receives the method call information from the server and then translates these into direct calls to the MIDP UI libraries stored on the mobile device. Since these standard application-independent UI libraries are the only classes that can make direct changes to the actual display, repeating the calls to the UI libraries on the mobile phone will produce the same user interface as would be shown in the emulator on the server-side.

The client-side prototype application is also responsible for sending any user input to the mobile

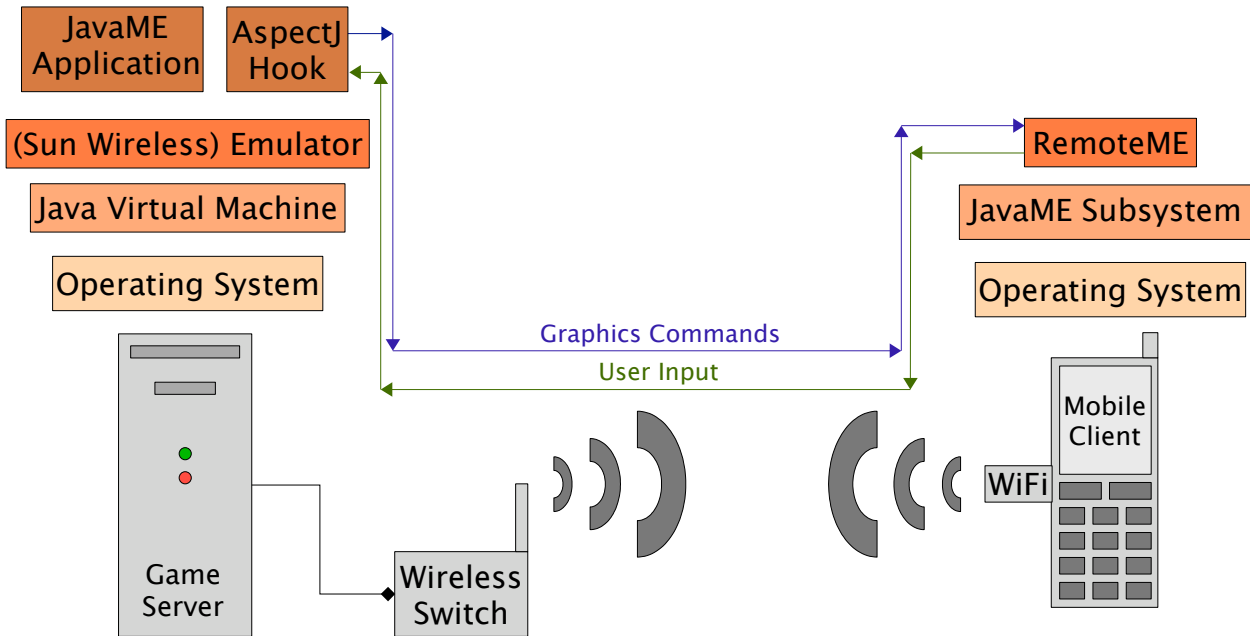


Figure 1: The client/server architecture of RemoteMe. The server runs an mobile device emulator that hosts the target application. AspectJ aspects are woven into the target application to capture the UI library calls that are ordinarily used to create the user interface. These calls are copied and transmitted to the mobile-based client, where a small application then forwards on the calls to the local UI libraries. This results in the server-side application’s user interface being shown on the mobile device. The mobile client then sends back any user input for processing by the server-side application.

device back to the server. The server takes this input and provides it to the target application in the emulator. The target application receives the input and processes it as if it had received it directly. A consequence of this may be that the applications makes subsequent UI library calls, which in turn are captured and sent to the client for presentation as the response to the user’s input.

Future versions of this architecture will also include client/server components for handling compression of transmitted data. This will lead to further experiments to see whether the time savings inherent in transmitting less information outweighs the time costs inherent in the compression / decompression algorithms.

4.2 Issues

We identified three non-trivial issues while developing for mobile devices, that are worth exploring further. These issues are: mobile user interface display abstraction; the wireless environment context for the mobile/server communication; and the issue of battery life.

We will now discuss these issues, with particular focus on the first two issues. The third issue — battery life — is still an item of future work.

4.2.1 Mobile User Interface Display

Mobile device user interfaces are specified differently than traditional desktop-based user interfaces. In one respect this difference is determined by our choice of which programming language to target with our prototype. There are a variety of programming languages used to develop mobile device software. Java is one of (if not the) most widely supported language on commonly available mobile phones, and for this reason we have made the decision that our prototype would target the *Java Micro Edition* (JavaME) platform.

One consequence of using JavaME (and developing for mobile devices in general) is that we must address the heterogeneity of mobile device screens and input devices. Different mobile devices may have different screen sizes, and equally importantly may have different keypads. These keypads may have a combination of fixed-function keys (where a particular key has a particular meaning) and programmable keys (where the meaning of a key can be application-dependent). This combination can be arranged differently on different keypads, and therefore our mobile applications may be more usable if the mobile application’s user interface can be tailored to take account of differences in mobile device platforms.

While not all mobile user interfaces are handled in this way, user interfaces that use the standard JavaME Command objects to represent possible user actions do so knowing that commands may be accessible through different constructs (such as menus, or direct mapping to a button) on different devices. This does differ from standard desktop user interfaces where, while visual style may vary and certain standard dialog boxes may be desktop-specific, the fundamental mapping of commands to widgets is relatively stable across desktops.

If interfaces do use these device-dependent techniques — where specific devices interpret exactly how to present the commands described in the user interface — then solutions such as VNC won’t easily work unless the emulator on the remote desktop is an exact match for the mobile device being used. Our approach is more general since the actual layout of user interface components in such situation is still left to the mobile device.

4.2.2 Wireless Environment

The number of people on the wireless network will have an impact on the performance of the system. When multiple people are on the same network, then they will share the available bandwidth. There is a

possibility that the network performance (from an individual’s perspective) will degrade beyond the point required to support the targeted response times.

4.2.3 Battery

One consideration that we have not yet investigated is the impact of remote interaction on the mobile device’s battery life. Given the significant use of the network, this will be a non-trivial problem worthy of future analysis — although improvements in battery technology may alleviate this somewhat in the future.

5 Experiments

The overall project hypothesis is that our proposed architecture will allow users to use remotely executing software on their mobile devices within the required response time parameter of 100ms. To further motivate our exploration of this hypothesis, we have undertaken preliminary experiments that capture representative response times for an existing mobile remote desktop software solution, along with the first prototype of our architecture.

In this section we will describe these experiments and discuss their results.

5.1 Null Hypothesis

The null hypothesis of our experiment is that the latency for the existing solution and our prototype (with a 90% confidence interval) will be strongly equivalent. We have chosen this null hypothesis as a starting point to see if our current prototype offers any performance improvements over existing solutions.

5.2 Experimental Setup

Overview We gained access to two mobile games for use as test applications. We then ran these games on a HTC Titan mobile phone through a VNC client, and our prototype. Both of these remote desktop solutions had minor modifications made to them to allow automatic monitoring of response times to user actions. This meant we required access to source code. Of the remote desktop software that we surveyed in our related work, only a single VNC client — J2ME VNC (Lee 2008) — was available under an open source license, and therefore suitable for our experiment. Alternative approaches would need to be developed to test techniques such as Microsoft’s Remote Desktop Mobile, and this is a worthy avenue to explore in future work.

Test Applications We chose two different JavaME games with different levels of graphics detail. The first game, Asteroid Zone (Doue 2008), is very lightweight in terms of graphics, opting for manual drawing of game objects rather than images (figure 2(a)). The second game, Bomber 2 (Yank 2008) is quite graphics intensive, using images for game objects as well as an animated background (figure 2(b)).

Server & Wifi The server used in the tests was a Dell Optiplex GX270 running the Ubuntu 8.04 OS and version 2.5.2 of the Sun Wireless Toolkit. The server had a Intel Celeron 2.4GHz processor and 1024MB of RAM. We used a private wireless network that ran on a different channel to the main University wireless network. We were the sole user of the wireless network, and the experiments were run while at a distance of 2 meters from the wireless router and 1 hop from the server.

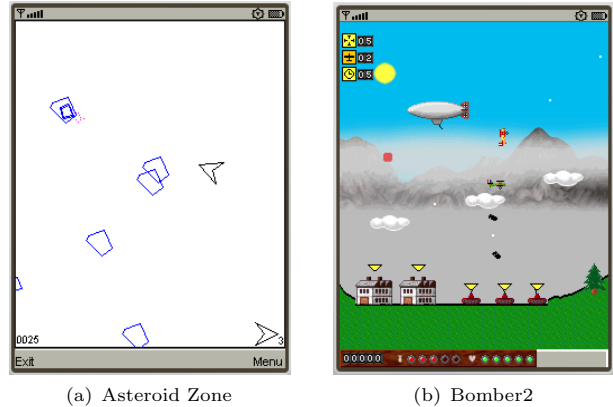


Figure 2: Screenshots of the Asteroids and Bomber2 games used during the preliminary experiments. Both games are real-time, in so much that noticeable lags between user input and the in-game response can detract from the game’s playability. Bomber2 has a more visually complex user interface.

5.3 Procedure

We initially planned to perform five runs of each game on each system. Unfortunately unrelated hardware difficulties precluded us from performing the Bomber2 runs on the RemoteMe system prior to submission of this paper. Subsequently, we present findings based on five runs of AsteroidZone on each of RemoteMe and VNC, and five runs of Bomber2 on VNC. We continued with testing Bomber2 on VNC as that would at least show a typical response capability for VNC, and provide some benchmark against which to test subsequent improvements in RemoteMe.

A run involved playing the game, and each play (for the same game) was similar although not strictly identical. The user endeavoured to play the game the same each time, although variations in the game itself caused some different actions to be taken during different game plays. Each run resulted in around 80–100 user inputs and associated responses by the utilised system.

Next we describe how the modifications needed to capture the timings for input/response pairs.

For the VNC-based tests, we modified the games to display a 16x16 square in the top-left corner of the screen, above all game objects. The square is initially solid black, and changes to the next color in the list of predefined colors upon a key being pressed. The J2ME VNC client has been modified to record the color of the center pixel of the aforementioned square, which is at (8,8).

When a key is pressed on the VNC client, the current time in milliseconds is recorded from the return value of Java’s `System.currentTimeMillis()` method. The time is again recorded when the center pixel of the square changes color on the client, and the difference between the key press time and the pixel update time is the effective response time.

The VNC server used was TightVNC (TightVNC 2008), running at a resolution of 240x391 and color depth 24. Inside the VNC server was an instance of the Sun Wireless Toolkit emulator running the respective games.

The procedure for RemoteMe was simpler. The game server is continuously communicating with the client, providing graphics updates. When a key is pressed on the client, the client records the current time in milliseconds given by Java’s `System.currentTimeMillis()` method, and sends the key press to the server, along with a unique iden-

tifier. When the update which contains the same unique identifier is sent to the client, the current time is recorded, and the difference between the two recorded times is the effective response time.

Note that in both cases the second timestamp is taken after the updates have been rendered on the screen.

5.4 Experimental Results

Our null hypothesis was that the 90% confidence interval for VNC will be strongly equivalent to the 90% confidence interval for RemoteMe. Figure 3 shows the results.

The interval for Asteroid Zone on VNC ended up being 370ms to 404ms. The interval for Asteroid Zone on RemoteMe ended up being 247ms to 287ms. Given the difference between the confidence intervals, the results reject the null hypothesis. The interval for Bomber2 on VNC was 898ms to 926ms. While this is of no use with respect to our null hypothesis, it is still useful to show that VNC cannot provide in this case the required response times to user input.

It is worth noting however that RemoteMe was unable to achieve response times to user actions under the required 100ms.

5.5 Experimental Limitations

This is the first round of experiments for this project. While the results are indicative that there are some situations where existing mobile remote desktop solutions are not suitable for our requirements, we have not tested a sufficiently large breadth of wireless environments to claim that there are no situations where these existing solutions would be suitable. Similarly, while our prototype performed reasonably well for Asteroid Zone, we are again aware that some wireless environments substantially differ to that we tested in, and we may see worse results in those environments.

Our experiments were only run with a single competing remote desktop solution, and only run on one type mobile phone. Further experiments will expand these two dimensions.

Lastly, we need the signal strength and the number of users on the wireless network to get results within a more realistic environment. We chose to run experiments in our simplified environment in part because if the systems could not work in the simplified environment, then they would not be able to work in more complicated environments either.

6 Summary

We have presented the RemoteMe architecture and prototype. We are using RemoteMe to explore a technique for interacting with remotely executing software as if it was executing locally on our mobile device. We are particularly interested in whether such a system can provide feedback to user actions within a 100ms response time. This paper discusses preliminary experiments that we have run to ascertain the suitability of our first prototype and an existing VNC-based solution. Results indicate that our prototype — RemoteMe — is an improvement in some situations over VNC (an existing solution). However, results also show that RemoteMe still hasn't satisfied the requirement of providing a response within 100ms.

Future work will include greater focus on the role that network distance and congestion play in the response times, and an exploration of whether compressing data communication between the client and server will improve response time. On this latter

point, we shall look at various compression strategies and contrast the time saved by transmitting less data against the time spent in compression and decompression.

References

- BBC (2007), 'Pilotless police drone takes off', <http://news.bbc.co.uk/2/hi/uknews/england/merseyside/6676809.stm>.
- Doue, J.-F. (2008), 'Asteroid Zone', <http://jfdoue.free.fr/index.html>.
- Gosling, J., Joy, B. & Steele, G. L. (1996), *The Java Language Specification*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Greg Roelofs, J.-l. G. & Adler, M. (2008), 'zlib', <http://www.zlib.net/>.
- Hinze, A. & Buchanan, G. (2006), The challenge of creating cooperating mobile services: experiences and lessons learned, in 'ACSC '06: Proceedings of the 29th Australasian Computer Science Conference', Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 207–215.
- Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. & Griswold, W. G. (2001), An overview of aspectj, in 'ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming', Springer-Verlag, London, UK, pp. 327–353.
- Lee, M. L. (2008), 'J2ME VNC', <http://j2mevnc.sourceforge.net/>.
- MSDN (2008), 'RDP', <http://msdn.microsoft.com/en-us/library/cc240445.aspx>.
- Nielsen, J. (1993), *Usability Engineering*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- NoMachine (2008), 'NX', <http://www.nomachine.com/documents/NX-XProtocolCompression.php>.
- Packard, K. & Gettys, J. (2003), X window system network performance, in 'USENIX 2003: Proceedings for the Annual Technical Conference, FREENIX Track', Cambridge Research Laboratory, HP Labs, pp. 207–218.
- TightVNC (2008), 'TightVNC', <http://www.tightvnc.com/>.
- XFree86 Project, Inc (2008), 'LBX', <http://www.xfree86.org/current/lbxproxy.1.html>.
- Yank, K. (2008), 'Bomber2', <http://j2mebomber.sourceforge.net/>.
- Zhuang, H. & Wang, Z. (2006), 'IP-based real time video monitoring system with controllable platform', *Mechatronic and Embedded Systems and Applications, Proceedings of the 2nd IEEE/ASME International Conference on* pp. 1–4.

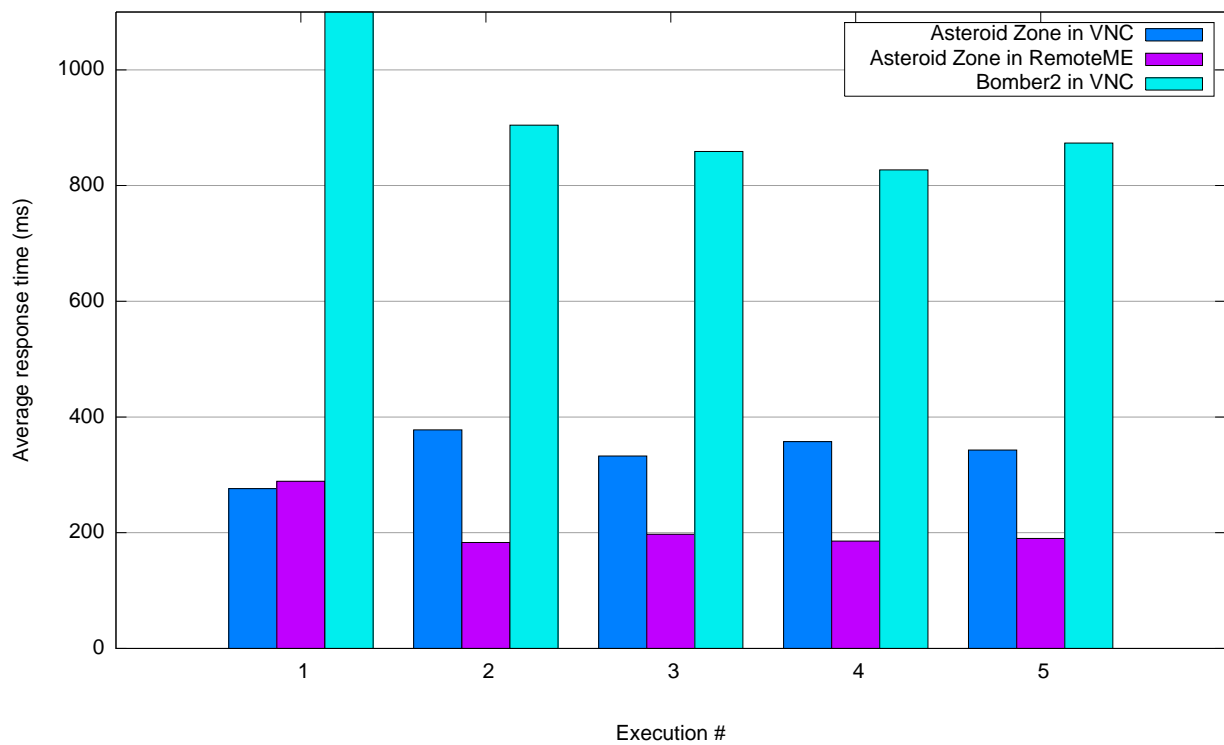


Figure 3: Average response times for Asteroid Zone and Bomber2 on VNC, and Asteroid Zone on RemoteME. We performed five runs on each system/game pair tested. Unfortunately unrelated hardware issues precluded the Bomber2/RemoteMe pair being tested at the time of submission. We then found the mean average for the individual input/response timings within each run. Note that the 1st run of Bomber2 on VNC was significantly out of line with subsequent runs, and we have discarded the value in our statistical analysis since it is an extreme outlier.