NEOTYS

Mobile Load Testing Challenges

A Neotys White Paper

www.neotys.com

Table of Contents

Introduction

Mobile applications and mobile websites have become a major channel for conducting business, improving employee efficiency, communicating, and reaching consumers. In the past, mobile played a smaller role in business applications, so performance issues and outages were less of a concern.

This is no longer the case. Today, performance problems with mobile applications lead directly to revenue loss, brand damage, and diminished employee productivity.

Application developers have long understood the need for load testing conventional desktop web applications to ensure that they will behave properly under load with the expected number of users. With the advent of mobile apps and mobile websites the principles of load testing have not changed. There are, however, challenges specific to mobile load testing that must be addressed by your load testing solution.

Since mobile apps and applications for desktop web browsers use the same underlying technologies, the good news is that most load testing tasks and challenges are the same. You don't necessarily need a brand new, mobile-specific load testing tool, but you do need a quality web load testing tool capable of handling the nuances of load testing mobile apps. Using a tool that enables testing of traditional *and* mobile web applications enables you to leverage existing in-house skills for designing and parameterizing your scripts, running your tests, and analyzing the results.

Aside from the similarities between traditional and mobile load testing are three key differences:

- **Simulating network for wireless protocols**: With 3G wireless protocols, mobile devices typically connect to the Internet using a slower, lower quality connection than desktops and laptops. This has an effect on response times on the client side and on the server itself, which you'll need to account for as you define your tests and analyze your results. Additionally, latency and packet loss becomes more of a factor with mobile applications and needs to be considered.
- **Recording on mobile devices**: Obviously, mobile apps run on mobile devices, and this can make it difficult to record test scenarios, particularly for secured applications that use HTTPS.
- **Supporting a wide range of devices**: The many different kinds of mobile devices on the market have led web application designers to tailor content based on the capabilities of the client's platform. This presents challenges for recording and playing back test scenarios. Also, each device has a different hardware configuration (CPU, memory), cache size, way to handle requests, etc. This makes it difficult to ensure that the user experience is the same on each device.

This paper discusses the challenges associated with mobile load testing, as well as solutions and best practices for recording mobile load test scenarios, conducting realistic tests, and analyzing the results.

Mobile load testing basics

As you may know, a typical automated *functional* test for a mobile application emulates user actions (including tap, swipe, zoom, entering text, and so on) on a real device or an emulator. A real mobile device testing tool measures the time needed to render a page on a specific mobile device. The objective of *load* testing, however, is not to test the functionality of the application for just a single user. Rather, the goal is to see how the server infrastructure performs when handling requests from a large number of users, and to understand how response times are affected by other users interacting with the application.

An effective load test simulates a high volume of simultaneous users accessing your server via your application. Using real devices or emulators for this task is impractical because it demands acquiring, configuring, and synchronizing hundreds, thousands or even hundreds of thousands of real devices or machines running emulators.

The current market climate underscores the importance of mobile application performance as experienced by the user. Unfortunately for testers, the user experience of mobile users is highly impacted by characteristics of the device and the quality of the network of the user making it difficult to accurately replicate production conditions with conventional testing tools and approaches. Key questions that are difficult to answer include: How is it possible to validate the stability of the platform, the stability of the response times, and a good user experience under load? How can you measure the user experience while staying under budget for the project?

The solution is to use a protocol-based load testing approach that is designed to scale as needed in conjunction with real mobile device testing tools. With a client-based approach, user actions in the browser or the native application are recorded and played back. In contrast, a protocol-based approach involves recording and reproducing the network traffic between the device and the server. To verify performance under large loads, tools that enable protocol-based testing are superior to those that support only client-based testing because they can scale up to millions of users while checking errors and response times for each user.

The basic process for protocol-based mobile load testing is:

1. Record the network traffic between the device and the server
2. Replay the network requests for a large number of virtual users
3. Analyze the results

But the protocol-based approach won't retrieve every metric of user experience of the mobile users meaning it won't include the rendering time of the mobile device in the response time metrics.

Each mobile device has its own cache size, way to send requests, memory size, processor, etc., so the user experience could slightly different from an IOS user to a Windows phone or Android user.

In order to measure the complete mobile user experience while the application is under load, it is important to combine a small number of users from a real mobile device testing tool with the protocol-based approach.

It may appear straightforward, but there are challenges at every step. The good news is that these challenges can be addressed with an effective load testing approach.

Recording mobile load testing scenarios

To generate a mobile test scenario, you first need to identify the type of mobile application under test. Challenges associated with capturing the data exchanges between a mobile application and the server depend on the design of the application:

- **Native apps -** These apps are coded using a programming language (Objective-C for iOS, Java for Android) and API that is specific to the device. As such, they are tied to a mobile platform and are installed from an online *store* or *market*.
- **Web apps -** Built with web technologies (such as HTML and JavaScript), these applications can be accessed from any mobile browser. More sophisticated web apps may use advanced features like geolocation or web storage for data or include customizations to better match the browser used. Two popular web apps are http://touch.linkedin.com and http://m.gmail.com.
- **Hybrid Apps** – A web app embedded in a native app is known as a hybrid app. The native part of the application is limited to a few user interface elements like the menu or navigation buttons, and functions such as automatic login. The main content is displayed in an embedded web browser component. The Facebook application, installed from an online *store* or a *market* is a typical sample.

Recording tests for native apps

Because native apps run on your device or within an emulator, to record a test you need to intercept the network traffic coming from the real device or the emulator.

To intercept this traffic, the equipment that records the traffic must be connected to the same network as the device. When the recording computer is on the intranet behind a firewall, it is not possible to record a mobile device connected via a 3G or 4G wireless network. The device and the computer running the recorder must be connected to the same Wi-Fi network.

Most load testing tools provide a proxy based recorder, which is the easiest way to record an application's network traffic. To use this approach, you need to configure the mobile device's Wi-Fi settings so that the traffic goes through the recording proxy. Some mobile operating systems, such as iOS and Android 4, support making this change, but older versions of Android may not. Moreover, some applications connect directly to the server regardless of the proxy settings of the operating system. In either of these cases, you need a tool that provides an alternative to proxy-based recording methods based on network capture or tunneling.

Note: You can use the following simple test to check if the application can be recorded using a proxy.

First, configure the proxy settings on the device and record your interactions with any website in a mobile browser. Then, try to record interactions in the native application. If your testing tool successfully records the browser generated traffic, but does not record traffic generated by the native application then you can conclude that the native application is bypassing the proxy settings and that an alternative recording method is required.

Recording tests for web apps and mobile version of websites

Web apps use the same web technologies as modern desktop browsers. As a result, you can record the application or the mobile version of a website from a modern browser on your regular desktop computer, which is an easier and faster alternative to recording from the device.

Many web applications check the browser and platform used to access them. This enables the application, when accessed from a mobile device, to redirect to a mobile version of the content that may contain less text or fewer images. To test such an app from the desktop, you need to modify the requests to make them appear to the server to be coming from a mobile device. Otherwise, you will not be testing the mobile version of the application as the server may redirect to a desktop version. Some browser plugins provide the ability to alter the identity of the browser (by modifying the User-Agent header of requests). Support for this feature is also directly integrated in the recorder of advanced load testing tools.

Modifying the browser's identity is not always enough. You obviously cannot use this approach to transform Internet Explorer 6 into an HTML5 compatible browser. The browser you use on the desktop must be able to parse and render content created for mobile browsers, so it's best to record with a modern browser like Internet Explorer 9, Firefox 5, Chrome 15, or Safari 5 (or a more recent version of any of these if available). If the application includes WebKit specific features, you should use a WebKit based desktop browser, preferably either Chrome or Safari.

Recording tests for hybrid apps

Obviously, tests for native apps cannot be recorded using a desktop browser. However, tests for many hybrid apps can. You may be able to directly access the URL used for the application, for example http://m.facebook.com for the Facebook application, and record your tests as you would for a classic web app.

Recording tests for secured native applications

There are additional challenges to consider when recording tests for a secured native application, that is, an application that uses HTTPS for the login procedure or any other processing.

By default, all HTTPS recording methods, whether proxy or tunnel based, are seen as man-in-the middle attacks by the device. This raises a non-blocking alert in a desktop or mobile browser but it leads to an outright connection refusal in native applications, making it impossible to record the secured traffic.

The only way to record tests for secured native applications is to provide a root certificate that authorizes the connection with the proxy or tunnel. While this feature is currently supported by relatively few load testing solutions, it is essential for load testing any native application that relies on HTTPS.

Note: The root certificate must be installed on the device. This operation is simple for iOS devices; you can simply send the certificate via email and open the attachment on the device. For other platforms, including Android, the procedure is not as straightforward and may depend on the version of the operating system and the manufacturer of the device.

Creating scenarios for real device testing

For mobile applications, the QA team should use real mobile device testing tools to validate the functionality of the application through user journeys on several devices. Those tools offer the capability to create scripts that are portable on several devices.

As we all know, the testing cycles for mobile applications are typically much shorter than for normal desktop applications. Therefore, the performance engineer will want to reuse the testing scenarios the functional testing team created in the real device mobile testing tool.

The scenario will be then be used during the test with full integration between the load testing tool and real mobile device testing tool. The performance engineer will select the population of representative mobile devices (the devices that match the user base).

With this approach the performance engineer should only have to configure the load testing tool to execute mobile testing scenarios and correlate the final results with response times and real device metrics.

Running realistic tests

Once you've recorded a test scenario, you need to be parameterize it so that it can emulate users with different identities and behaviors as it is played back to produce a realistic load on the server. This step is required for traditional and mobile web applications, and the tools used to complete it are the same. When playing back the test scenarios, however, there are several challenges specific to mobile load testing.

Simulating network conditions

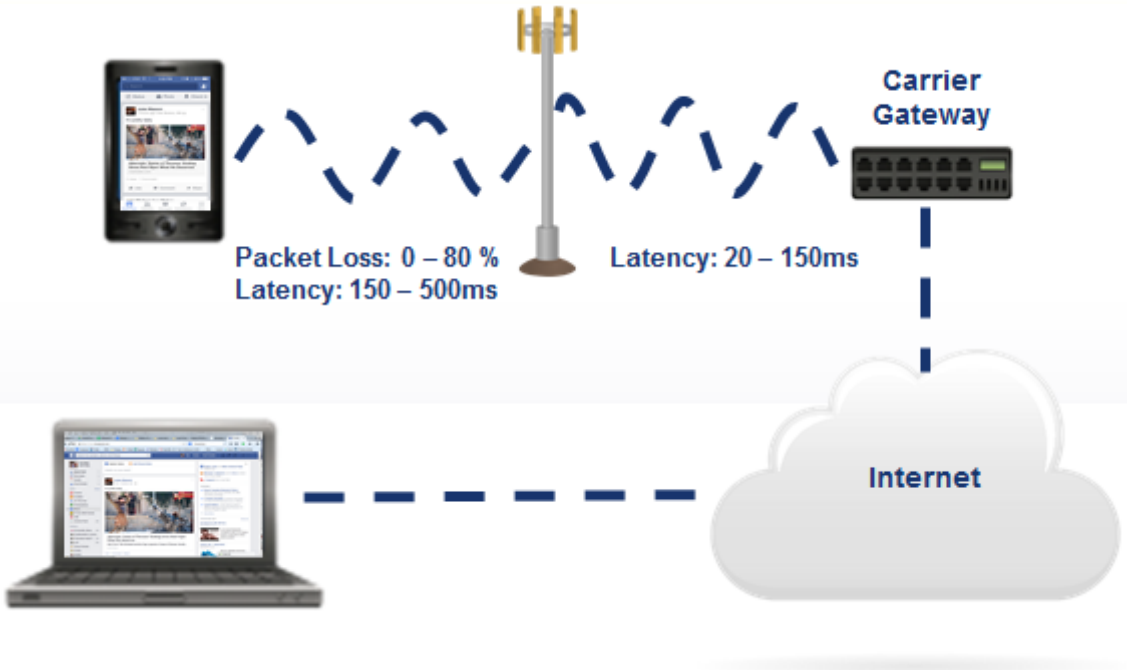
Today's mobile devices generally access the server over networks that are slower than those used by desktop computers. Network characteristics including bandwidth, latency and packet loss have a huge impact on client response times and on the way the server is loaded. By simulating different network conditions in a test lab environment you can forecast the effects of changes in the network infrastructure on the application's performance. Doing so also allows you to discover application issues in the development cycle, therefore reducing costs.

Network conditions and response times

The bandwidth is directly correlated with how long it takes to download data from the server. The lower the bandwidth, the higher the response time. A server that provides acceptable response times for desktop users using DSL or another high-speed broadband service may deliver a poor end-user experience to mobile users with lower bandwidth.

Additionally, mobile networks have high latency compared to WiFi and broadband. Since the latency is time added to each request and webpages are composed of many sub-requests, the time required to load a webpage on a mobile device greatly depends on the latency.

It is important to validate your service-level agreements (SLAs) and performance objectives with tests that use the same network conditions as your users to avoid making decisions based on misleading test results. Such tests must incorporate *network emulation,* which is the process of artificially slowing down the traffic and adding latency during a test to simulate a worse connection.



Network conditions and server load

Clients using bad network connections also affect the server. The lower the bandwidth, the longer the connections. Longer connections, in turn, lead to more simultaneous connections on your web server and your application server. Thus, mobile users tend to consume more connections than their wired counterparts. Most servers have settings that limit the number of simultaneous connection that they can handle. Without a testing tool that realistically simulates bandwidth, these settings cannot be properly validated.

Emulating network conditions for individual virtual users

When load testing, effective network emulation requires the ability to individually limit the network conditions for each user or groups of users, independent of the others.

Consider a situation in which you need to verify performance when 100 mobile users are accessing the server. In this scenario, you'd want to simulate 100 virtual users, with each user limited to a 1Mbps 3G connection. In this case, the total bandwidth for all users is 100Mbps (100 users * 1Mbps/user). Though it is possible to use WAN emulation software or a network appliance to globally limit the bandwidth for the load generation machine to 100 Mbps (or any other arbitrary limit), in practice this does not provide a realistic test because it does not impose a strict 1Mbps constraint on each user. Network emulation support must be integrated in the load testing tool itself to enable network limits to be applied to individual virtual users.

To conduct an even more realistic test, you'll want to simulate a mixed population of users accessing your application with a variety of bandwidths, latency, and packet loss. With a tool capable of network emulation on a per virtual user basis, you can determine the response times for users at each bandwidth across a range of network conditions in a single test. This saves times when you need to compare the response times of web applications and business transactions for clients who have different network conditions.

Simulating devices, browsers and browser capabilities

When a browser requests a resource from a web server, it identifies itself via the user-agent header sent with each request. This header contains information about the browser and the platform on which it is running. Servers use this information to deliver different versions of the content based on the client system. As noted earlier, many web applications deliver different content to mobile users and desktop users. Some further differentiate mobile users into subgroups based on information in the user-agent header, delivering less text and smaller images to devices with small screens. This can lead to bandwidth consumption and loading times that vary widely with the browser and platform being used.

As a result, the ability to manipulate the user-agent header is essential not only for recording test scenarios, but also for playing them back. Tools that lack this capability will fail to retrieve the appropriate content from the server.

Simulating parallel connections

Mobile browsers, like desktop browsers, can generate the HTTP requests needed to retrieve the static resources of a web page in parallel. Rather than waiting for each image to finish loading before requesting the next, this approach requests multiple images at once to shorten the overall page load time. To measure response times accurately, load testing tools must replicate this behavior by generating multiple requests in parallel. Moreover, they must simulate the *appropriate* number of parallel connections as this number may differ from one mobile browser to another. Again, tools that lack this capability are not performing realistic tests, placing the results they deliver into question.

Browser	#TCP Connections
IE 6 & 7	2
IE 8 & 9	6
IE 10	8
Firefox 2	2
Firefox 3	6
Opera 10	8
Opera 11 & 12	6
Chrome 1 & 2	6
Chrome 3	4
Chrome 4 to 23	6
Safari 3 & 4	4
Safari iOS7	5
Chrome Android 4	10
Android 4 Browser	6

Identifying the most appropriate settings for realistic tests

Finding the appropriate values for key test settings—such as the user-agent, bandwidth, and number of simultaneous connections—can be a challenge. More advanced load testing tools can help testers set these values. For example, test scenario playback is greatly simplified by tools that can automatically inform the tester of which user-agent string and number of parallel connections to use based on the browser name, version, and platform. The process is further streamlined when the tools can suggest the most appropriate upload and download bandwidth settings based on the technology used (for example, Wi-Fi, 3G, 3G+, and so on) and the quality of the signal (for example, poor, average, or good).

Retrieve relevant mobile KPIs

There are some important KPIs that must be measured during the testing of mobile applications. For example: What is the hardware usage of the mobile application on devices? Is it consuming too much battery? What is the real end-user experience on real devices?

All those KPI metrics should be gathered during mobile load testing with the help of a real mobile device testing tool.

By using the mobile device testing tool with the integrated load testing tool, the performance engineer will be able to guarantee that mobile users won't complain about high CPU or battery usage from the application on the device.

Today. mobile users are expecting equivalent or better response times than typical web applications, so the performance engineer will need to make sure that the end user experience of mobile user is not adversely affected by the load.

Using the cloud

You can use load testing with the cloud after (or in conjunction with) on-premise testing in the lab to improve the realism of your tests by generating high loads and testing from different locations, while saving time and lowering costs.

Generating a high load

For consumer-facing apps and websites, it is often difficult to predict the number of users your applications will have to handle. Traffic spikes that results from a promotion, marketing campaign, new product release, or even unexpected social network buzz can be substantial. To generate a similar load in-house, you would need a significant investment in hardware. Using the cloud, you can generate the same high load using on-demand resources at a much lower cost.

Testing from different geographies

Your web application's real users likely access the server from many different geographical locations and use different networks. To properly validate the application and the server infrastructure, your virtual users should operate under similar real world conditions with a cloud including integrated network emulation.

Testing the entire application delivery chain

When your real users are located outside the firewall, you should run your virtual users from the cloud to validate the parts of the application delivery chain that are not tested when testing from the lab, including the firewall itself, load balancers, and other network equipment.

Tools for testing with the cloud

While the cloud represents an opportunity to rapidly increase the scale and improve the realism of load testing at low costs, cloud testing is most effective when it used to complement internal load testing. Note that the primary factor in the success of load testing with the cloud is not the *move* to the cloud itself, rather it is the tool you select and how well it uses cloud technology. In particular, it's best to select a solution that is integrated with multiple cloud platforms, enables in-house test assets to be reused in the cloud, and supports realistic, large-scale tests across multiple geographical regions.

Analyzing results

The default results of a load test are frequently delivered as averages. For example, load testing tools will typically show what errors occurred and the average response times for a request, web page, or business transaction regardless of the type of users being simulated or the bandwidth available to them.

Because bandwidth may vary widely for the different kinds of users simulated, the errors and response times can also vary widely. Taking an average of results with significant variation does not provide an accurate picture of what is really happening. To gain meaningful insights and to validate your SLAs and performance requirements for each network condition, it is important to go beyond the default results and analyze the results for each kind of user.

Conclusion

In many ways, mobile load testing is similar to load testing classic web applications. As a result, testers can leverage much of their existing knowledge and reuse existing techniques—like using the cloud for realistic, large scale tests. However, there are specific requirements for testing mobile applications that are not addressed by traditional load testing techniques. Recording mobile test scenarios, conducting realistic tests that simulate real-world network and browser characteristics, and properly analyzing the results are some of the key areas that require special attention for mobile applications. Addressing challenges in these areas is essential to ensuring mobile web applications are sufficiently tested prior to release and that they will perform well under load in production.

About Neotys | www.neotys.com

Since 2005, Neotys has helped over 1,500 customers in more than 60 countries enhance the reliability, performance, and quality of their web and mobile applications.

The best-in-class NeoLoad load & performance testing solution is flexible, easy to use with infinite scalability from the cloud and support for all web 2.0 and mobile technologies. All this is backed by a dedicated team of Neotys professional services and support to ensure your success. For more information about Neotys and NeoLoad visit: **www.neotys.com** or contact **sales@neotys.com**

Contact for More Info

US: Tel: +1 781 899 7200 | **EMEA:** Tel: +33 442 180 830
Email: sales@neotys.com | **Learn More:** www.neotys.com