Mobile Load Testing Challenges

A Neotys Whitepaper

Table of Contents

Introduction

With the advent of the app economy, mobile apps have revolutionized how business gets done. Mobile applications are not only the primary channel for the millennial generation but also are a significant driver of the Internet of Things. Mobile apps and websites process billions in business transactions, provide the primary communication conduit of brands, gather reams of data that fuel analytics initiatives, and transform the customer experience. In short, mobile apps undergird much of the economy.

In the past, when mobile played a smaller role in the business application ecosystem, performance issues and outages were considered an inconvenience; this is no longer the case. Today, performance issues that impact mobile applications tie directly to revenue loss, brand damage, and diminished employee productivity. The ubiquity of mobile apps that instill delight in the minds of customers is now the benchmark of broader app performance. Besides, as organizations rely on mobile channel transactions to enhance revenues, they apply relentless pressure on development teams to deliver meaningful consumer experiences. These factors together present a moving performance target for app DevOps teams and drive the need to create tests that accurately represent real users regarding network conditions, specific devices, and geographic locations.

Application developers have long understood the need for load testing conventional web applications to ensure proper behavior under load. These same principals of load testing apply to mobile apps and mobile websites. There are, however, challenges specific to mobile load testing that must be addressed by the load testing solution.

Because mobile apps and applications accessed by desktop web browsers use the same underlying technologies, there is good news-- most load testing tasks and challenges are the same. Testers don't necessarily need a new, mobile-specific load testing tool. They do, however, require a high-quality web-based load testing tool capable of handling the nuances of load testing mobile apps. Using a tool that enables testing of both traditional and mobile web applications allows QA to leverage existing in-house skills for designing and parameterizing test scripts, running tests, and analyzing the results.

Note that there are three fundamental differences to consider when executing traditional vs. mobile load testing:

Simulating network for wireless protocols: Mobile devices can experience slower, lower quality Internet connections than their desktop counterparts depending on the speed of wireless protocols available. 4G wireless protocols and future low-latency/high-speed networks, such as 5G, will impact client-side response times and the server itself, which testers will need to account for as they define tests and analyze results. Additionally, latency and packet loss become more of a factor with mobile applications.

Recording on mobile devices: Obviously, mobile apps run on mobile devices, making it difficult to record test scenarios, particularly for secured applications that use HTTPS.

Supporting a wide range of devices: The plethora of mobile devices and their respective operating systems demands that web application designers tailor content based on the rendering capabilities of the client platform, presenting challenges for recording and playing back test scenarios. To further complicate matters, each device has a different hardware configuration (i.e.CPU, memory), cache size, the method for handling requests, etc. All these factors conspire to increase ecosystem complexity, making it difficult to ensure a consistent user experience 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 that include tap, swipe, zoom, and text entry on a real device or an emulator. A specialized 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 a single user. The goal is to evaluate how the server infrastructure performs when handling requests from many users and to understand how other users are interacting with the application impacts response times.

A practical and realistic load test simulates a high volume of simultaneous users accessing the application server. 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 actual devices or machines running emulators.

The current, hyper-competitive market climate underscores the importance of mobile application performance as experienced by the user. Unfortunately for testers, the mobile user experience is profoundly influenced by the characteristics of the device and the quality of the telecommunication provider network, making it difficult to accurately replicate all production conditions with conventional testing tools and approaches.

Fundamental questions that can be 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 apply a protocol-based load testing approach 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 quickly scale up to millions of users while simultaneously checking errors and response times for each user.

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

- Record the network traffic between the device and the server
- Replay the network requests for a large number of virtual users
- Analyze the results

However, a protocol-based approach will not retrieve every metric of the mobile user experience, because it excludes the rendering time of the mobile device in the response time metrics. Remember that every mobile device has its operating system version, cache, the method of sending requests, memory capacity, processor capability, etc. All these variables influence the user experience across IOS, Windows, and Android devices. 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. Only in this manner can testers assess the impacts of bandwidth, latency, packet loss and throughput on the actual user experience.

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 and involve:

Native Applications - These apps are coded using a programming language (Objective-C for iOS, Java for Android) and an API specific to the device. As such, they are tied to a mobile platform and are installed from an online store or market.

Web Applications - 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, such as geolocation or web storage for data or even include customizations to match better the browser used. Two common web apps are http://touch.linkedin.com and http://m.gmail.com.

Hybrid Applications – A web app embedded in a native app is known as a hybrid app. The native component of the application is limited to a few user interface elements such as the menu or navigation buttons, and functions such as automatic login. The main content displays in an embedded web browser component. Facebook's app, installed from an online store or a market is a typical sample.

Recording Tests for Native Applications
Because native apps either run on a device or within an emulator, testers must intercept the network traffic that originates from either the real device or the emulator to record a test.

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 connected to the intranet behind a firewall, it is not possible to record a mobile device connected via the 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 provides the easiest way to record an application's network traffic. With this approach, testers need to configure the mobile device's Wi-Fi settings to route traffic through the recording proxy. Many mobile operating systems, such as iOS and Android, as well as proprietary and open source Android offshoots, such as Samsung's Tizen, support making this change. Note that older versions of Android may lack requisite support. Some applications connect directly to the server regardless of the proxy settings of the operating system. In any of these cases, you need a tool that provides an alternative to proxy-based recording methods based on network capture or tunneling.

You can use the following simple test to check whether the application can be recorded using a proxy. First, configure the proxy settings on the device and record interactions with any website in a mobile browser. Then, try to record interactions in the native application. If the testing tool successfully records the browser generated traffic but does not record traffic generated by the native application, then testers can conclude that the native application is bypassing the proxy settings and that an alternative recording method is required.

Recording Tests for Web Applications and Mobile Versions of Websites

Because web apps use the same web technologies as modern desktop browsers, testers can record the application or the mobile version of a website from a modern browser on a desktop, which presents 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 the content request to a mobile version optimized with less text and compressed images. To test such an app from the desktop, testers need to modify these requests such that they appear to the server to be coming from a mobile device. When testers neglect to compensate for this variable, they risk not testing the mobile version of the application; the server may redirect to a desktop version.

Some browser plugins allow testers to alter the identity of the browser by modifying the UserAgent header of requests. Support for this feature is directly integrated into the recorder of advanced load testing tools. However, modifying the browser's identity is sometimes insufficient. Testers obviously cannot use this approach to transform Internet Explorer into an HTML compatible browser. The browser testers use on the desktop must be able to parse and render content created for mobile browsers, so it's best to record with the most recent versions of modern browsers, such as Chrome, Internet Explorer, Firefox, or Safari. If the application includes WebKit specific features, then testers should use a WebKit based desktop browser, preferably either Chrome or Safari.

Recording Tests for Hybrid Applications

Although, tests for native apps cannot be recorded using a desktop browser, tests for many hybrid apps can. Testers may be able to access the URL used for the application directly, for example, http://m.facebook.com for the Facebook application, and record tests as they 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. Such "attacks" instigate a non-blocking alert in a desktop or mobile browser and lead 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; testers can simply send the certificate via email and open the attachment on the device. For other platforms, including Android, the process may vary depending 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 specialized mobile device testing tools on several devices to validate the application functionality through user journeys. 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). Using this approach, the performance engineer should only have to configure the load testing tool to execute mobile testing scenarios and correlate final results with response times and real device metrics.

Running Realistic Tests

After completing a test scenario recording, testers must then parameterize it. This ensures that as it is played back, it emulates users with different identities and behaviors and produces a realistic load on the server. This step is required for both 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 that testers must address.

Simulating Network Conditions

Today's mobile devices 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 Speeds

To offer brief technical background, 4G is the fourth generation of mobile network technology. Made to replace 3G, 4G offers a more reliable connection and delivers higher speeds. 4G LTE is "fourth-generation long-term evolution," a 4G variant honed to deliver the fastest connection for a mobile internet experience – up to 10 times faster than 3G. It's important to note that regardless of the spectrum, the distance from the wireless source, interference from other wireless networks, and the number of users sharing the same bandwidth may all conspire to limit speed. Further, 4G LTE vs. cable. Verizon 4G LTE wireless broadband is ten times faster than 3G and can handle download speeds between 5 and 12 Mbps

and upload speeds between 2 and 5 Mbps, with peak download speeds approaching 50 Mbps.

Network Conditions and Response Times

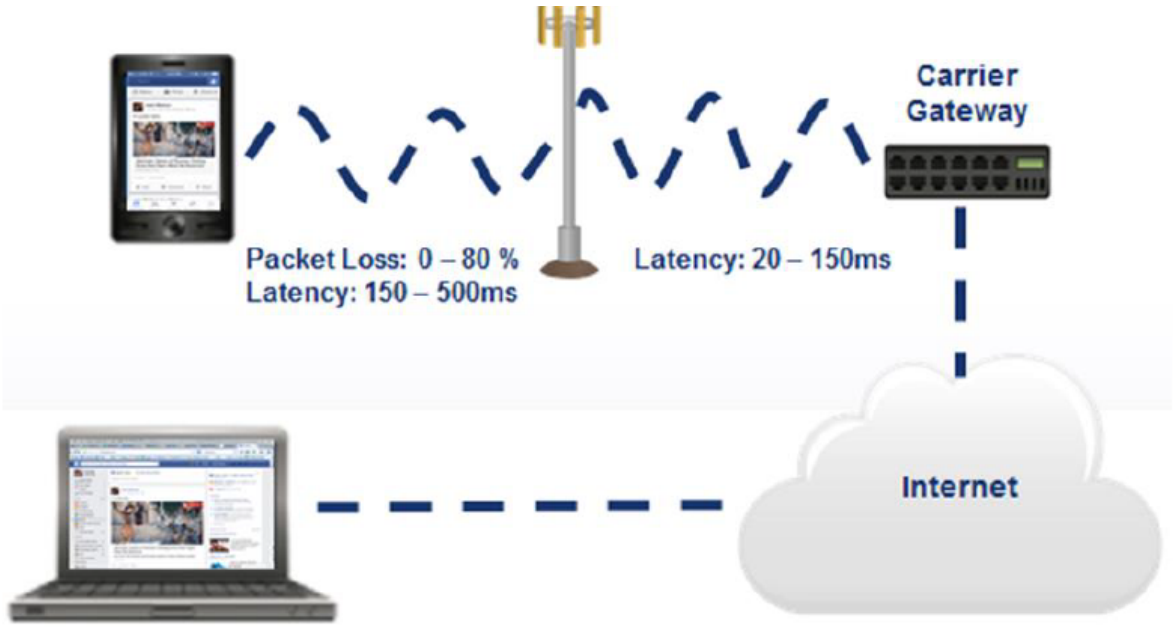
Network bandwidth directly correlates with how long it takes to download data from the server. The lower the bandwidth -- the slower the response time. A server that provides acceptable response times for desktop users using DSL or another high-speed broadband service will likely deliver a poor end-user 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 web pages 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 service-level agreements (SLAs) and performance objectives with tests that use the same network conditions as mobile users to avoid conclusions 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 with poor network connectivity can impact server performance. Lower bandwidth invokes longer connection time. Longer connections can lead to more simultaneous connections on both the web and application servers. For this reason, mobile users tend to consume more connections than their wired counterparts. Most servers have settings that limit the number of simultaneous connections they can support. Without a testing tool that realistically simulates bandwidth, these settings cannot be properly validated.



Emulating Network Conditions for Individual Virtual Users

To ensure accurate load testing, testers require effective network emulation that provides the ability to limit network conditions individually for each user or group of users, independent of the others.

Consider a situation in which testers need to verify performance when thousands of mobile users access the server. In this scenario, they want to simulate 1000 virtual users, with each user limited to a 10Mbps 4G connection. In this case, the total bandwidth for all users is 10000Mbps (1000 users * 10Mbps/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 10000 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 with the load testing tool itself to enable network limits to be applied to individual virtual users.

To conduct an even more realistic test, QA will want to simulate a mixed population of users accessing the application with a variety of bandwidths, latency, and packet loss. With a tool capable of network emulation on a per virtual user basis, testers can assess response times for users at each bandwidth across a range of network conditions in a single test. Such flexibility makes it faster and easier 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 with a user-agent header sent with each request. The 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 and its capabilities. Some servers 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. Variances in content delivery can result in bandwidth consumption and loading times that vary widely across devices, depending on the browser version and platform. 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 unable to manipulate the user-agent header 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, the browser 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. Tools that lack this capability are not performing realistic tests, putting the results they deliver into question.

Identifying the Most Appropriate Settings for Realistic Tests

Identifying appropriate values for key test settings, such as the user-agent, bandwidth, and the 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 automatically inform the tester about which user-agent string to use and the recommended number of parallel connections based on the browser name, version, and platform. The process is further streamlined when the tools can recommend the most appropriate upload and download bandwidth settings to use based on the technology in play. For example, insight about the most appropriate connection speed (e.g., Wi-Fi, 3G, 3G+, 4G, 4G LTE, etc.) and signal quality (e.g., poor, average, or good) can increase testing efficiencies.

Retrieve Relevant Mobile KPIs

During mobile application testing, it's important to track KPIs that relate to the hardware usage of the mobile application on devices, battery consumption, and end-user experience insights on real devices. Robust mobile testing tools gather KPI metrics throughout the mobile load testing phase.

By using a mobile device testing tool with an integrated load testing tool, the performance engineer can evaluate whether mobile users will experience high CPU or battery usage from the application on the device. In this way, the performance engineer can ensure that the mobile end user experience 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 that applications will have to support. Traffic spikes that result from a marketing promotion or campaign, a new product release, or even unexpected social network buzz can be substantial. To generate a similar load in-house, application providers would normally need to invest heavily in hardware. However, using the cloud, testers can generate the same high load using on-demand resources at a much lower cost.

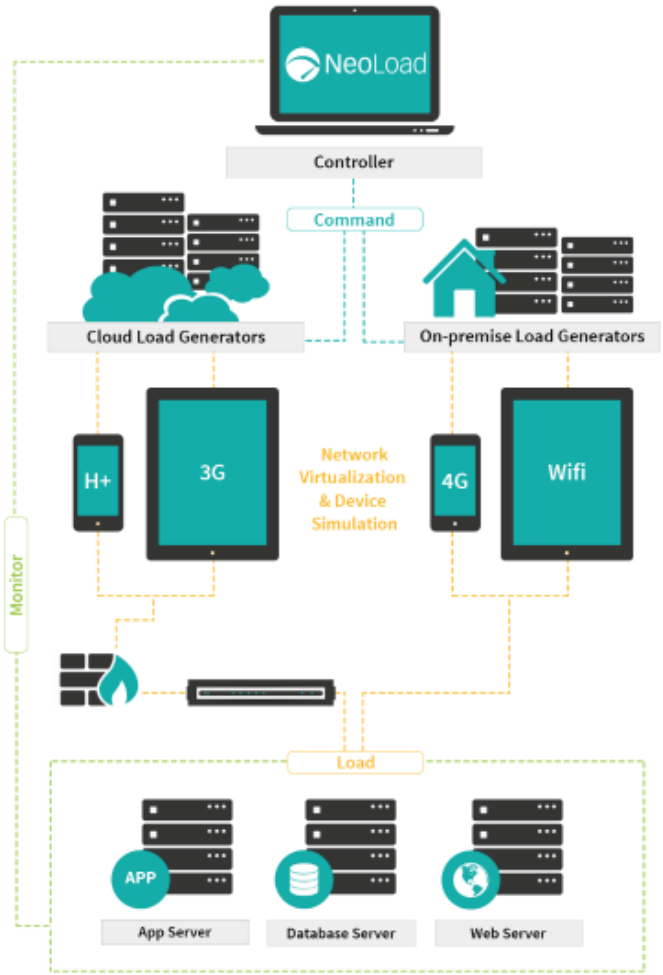
Testing from Different Geographies

Consumers of web applications often access the server from different geographical locations using different networks. To properly validate both the application and the server infrastructure, the virtual users under test should operate under similar real world "cloud" conditions that include integrated network emulation.

Testing the Entire Application Delivery Chain

When real users are located outside the firewall, testers should run virtual users from the cloud to validate the facets of the application delivery chain that are not under test in the lab, including the firewall itself, load balancers, and other network equipment.

For additional information regarding Neotys and the Cloud, please download our white paper, "Load Testing with the Cloud."



Tools for Testing with the Cloud

While the cloud represents an opportunity to increase scale and improve the realism of load testing at low costs, cloud testing is most effective when used to complement internal load testing. Note that the

primary success factor in cloud-based load testing is not the move to the cloud itself, rather the tool selected and how well it leverages cloud technology. It's best to select a solution that integrates with multiple cloud platforms, enables the reuse of in-house test assets in the cloud, and supports realistic, large-scale tests across multiple geographical regions.

Analyzing Results

Default results of load tests are frequently delivered as averages. For example, load testing tools reveal what errors occurred and the average response times for requests, web pages, and business transactions, regardless of the types of users being simulated or the bandwidth available to them.

Because bandwidth can vary widely across user simulation scenarios, respective errors and response times can vary widely as well. Interpreting the average of results with significant variation does not provide an accurate picture of what is really happening. To gain meaningful insights and to validate SLAs and performance requirements for each network condition, it is important to go beyond the default results and analyze the results for each type of user.

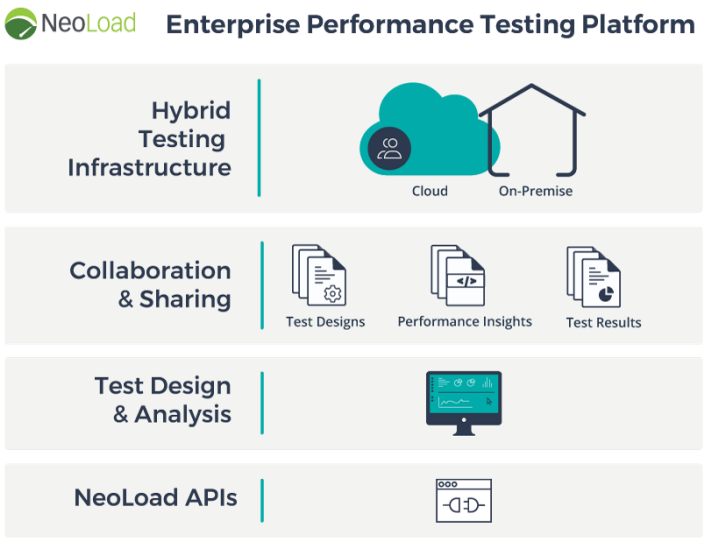
Conclusion

The demand for business agility and the opportunity to secure new revenue streams compels organizations to compete in the app economy. The complexity of the resulting application ecosystems places incredible burdens on QA and testing teams. The good news is that, in many ways, mobile load testing is similar to load testing classic web applications; testers can leverage much of their knowledge and reuse existing techniques, such as using the cloud for realistic, large-scale tests. There are, however, 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 test results are some of the key areas that require special attention when mobile application testing. Addressing challenges in these areas is essential to ensuring mobile web applications are sufficiently tested before release and that they will perform well and support business goals under load in production.

About Neotys

The success of your digital strategy relies on your ability to deliver fast and reliable software, regularly. Creating great software quickly, using an optimized performance testing process is your competitive advantage – Agile and DevOps are part of the solution.

Neotys has nearly 15 years of development investment into NeoLoad – the performance testing platform designed to accelerate Agile and DevOps processes. It's built by engineers who recognized that to achieve their Agile adoption objective; they needed to create a product that could facilitate superior load and performance testing continuously.



The result – up to 10x faster test creation and maintenance with NeoLoad.

We genuinely believe that the Performance Engineer can become the critical application performance partner providing the best testing coverage while respecting the cadence of the Continuous Delivery process. As performance becomes the responsibility of the wider team, continued delivery of an optimized performance testing platform is what drives our work every day.

For more information about Neotys and NeoLoad visit: www.neotys.com or contact sales@neotys.com