

# The World According to .NET: Optimizing .NET Applications for the Production Environment

A Shunra Software White Paper



## Executive Summary

Microsoft's .NET Framework (.NET) is steadily gaining favor with IT organizations. The idea behind .NET is to flexibly leverage information resources across the enterprise and beyond – mixing and matching them to respond to changing business needs. However, this isn't just a software and platform issue. It's a networking challenge as well. After all, you can only tap into diverse resources located all over your network if your network is sufficiently robust everywhere to support such activity. If it isn't, you're going to have problems on the production Wide Area Network (WAN) that you never noticed in development.

The characteristics of these production networks must therefore be taken into account in the design and development of .NET applications. Otherwise, constraints and/or inadequacies in network connectivity may adversely impact application performance.

Specific networking-related considerations for .NET applications include:

- The ability of an application or service to connect with and retrieve required data from multiple resources across a WAN with sufficient speed and performance
- The ability of the network to support the large volumes of requests directed to commonly used resources/servers
- The ability of the network to support delivery of client-side software updates in real-time across long geographical distances

These considerations are critical for .NET-based applications being delivered over the production network.

It is extremely important that development teams identify and resolve potential network-related bottlenecks as early as possible in the coding process. If such problems are discovered early on, they can be addressed relatively easily, and at a low cost-to-fix. If they aren't discovered until actual deployment on the production network – after coding is essentially complete and major architectural decisions have already been made – remediation can be extremely expensive. In some cases, it may be virtually impossible to modify the application enough to fully overcome these networking problems. Studies shows that it cost **50-200 times** more to make changes made late in the development cycle than it does to make them early. So early analysis and remediation is a must.

The way to avoid the risks of costly re-coding and/or substandard performance is to thoroughly test the performance and behavior of .NET applications in a testing environment that accurately replicates conditions on the production network. By doing so, developers, QA and network professionals can assess the “networkability” of the application at each step in the development process – and avoid unpleasant surprises down the road.

Pre-production testing in an emulated network environment is always advisable. To reap the full benefits of .NET – including the ability to tap into resources and deliver services to end-users anywhere on the network – it is an absolute necessity.

## Microsoft .NET: Leveraging IT Resources Across the Network

Microsoft .NET is a set of technologies designed to facilitate the development, deployment and operation of applications and Web services that fully leverage IT resources across and beyond the enterprise. By making extensive use of industry standards such as Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL) and Universal Description, Discovery, and Integration (UDDI), .NET enables developers to more quickly and easily integrate disparate applications and extend them to address emerging business requirements.

Many of the advantages that .NET offers arise from the fact that it provides a highly flexible component-based development model. Component-based development allows the presentation, business logic, data access and data layers of a business system to be managed independently of each other. Components can be flexibly reused by different applications in different ways. This allows businesses to fully leverage new and existing IT resources and thereby maximize returns on their investments in data, development and infrastructure.

In addition to using a component-based architecture, .NET implements a variety of other mechanisms – such as dynamic client-side code updates – to optimize the flexibility of business systems and the ease with which they can be implemented across the network.

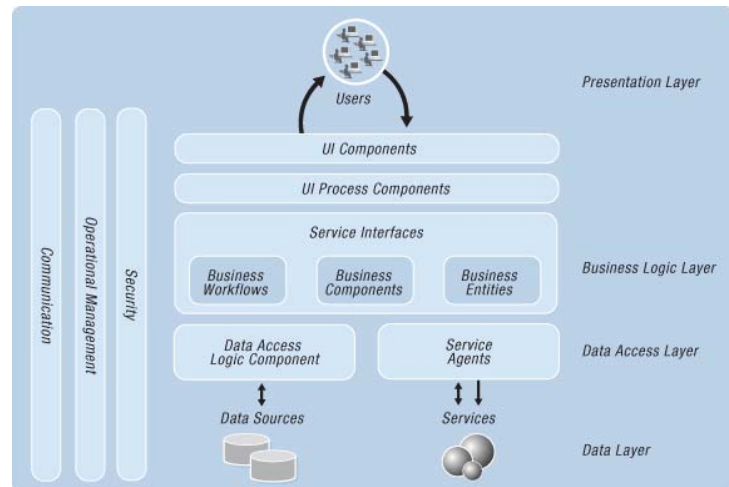


Figure 1. Typical Component-Based .NET Application.

In order to take full advantage of these .NET capabilities, developers, QA and networking professionals need to ensure that their applications are developed specifically for the network environments on which their .NET applications are going to be deployed. In the real world, network connectivity is often limited. These limitations must be discovered early on in the development cycle and accommodated to ensure optimum application performance in the production environment. After all, even the best application is of little value if end-users experience sluggish performance and/or unpredictable, intermittent periods of unavailability.

## Production Network Considerations for .NET Application Deployment

.NET applications depend on network infrastructure to deliver functionality to end-users, access back-end resources, synchronize data and update client-side code. This network infrastructure can include Local Area Networks (LANs) within a building or campus, WANs that link to remote locations in another state or another country, remote connectivity such as cable and DSL used by mobile employees and telecommuters, and – increasingly – wireless technologies such as WiFi and cellular commercial services.

The impact of this diverse network connectivity on the performance and behavior of .NET applications can be significant. Network links are characterized by:

- Bandwidth (amount of data that can be transmitted across a connection)
- Latency (the delay in transmission that results from physical distance and link utilization)
- Packet errors (the loss or re-ordering of data that inevitably occurs in the real world)
- Routing dynamics (the changes in the specific path packets take from point A to point B as a result of changing conditions on the network).

These characteristics all affect different applications in different ways. And, because these characteristics are always changing, the impact can change substantially over time.

Most important, the characteristics of WAN links are very different from those of a LAN. The large distances traversed by a WAN connection, for example, introduce much more latency than is present in a LAN connection within a single building. WAN typically provide far less bandwidth than LANs. And wireless connections can be particularly erratic and unreliable – creating a wide variety of problems for many kinds of applications. These problems will not be evident in a conventional application development environment, which only has highly reliable, high-bandwidth and low latency LAN connections.

The situation is somewhat analogous to driving a car. If you lived in Montana, where there is no one else on the road and no speed limits, you would be used to getting to places 100 miles away in under 90 minutes without having to tap your brakes or turn off your cruise control. That approach would obviously not work in New York City during rush hour – where roads are congested, construction delays are frequent, and turning on cruise control will guarantee that you will slam into the car ahead of you – and where you have to give yourself 90 minutes to travel 20 miles, not 100.

Similarly, .NET developers have to take traffic conditions on the network into account as they design and build applications. Specific considerations include:

### Component Distribution

The physical distribution of components across the network is an important consideration for optimizing application performance. When components are physically distributed across the network, end-to-end performance of any individual application task may depend on the cumulative

performance of many different network links. If these links are not all performing optimally at the same time, end-users may experience unacceptable sluggishness. If any individual link is performing particularly poorly at any given moment, the application may even fail entirely or lose critical data. That is why components and applications must be properly optimized for the specific types of conditions they will encounter on the production network.

### **Smart Clients**

Smart client .NET applications can use network connectivity for a variety of different purposes, including SQL client connections and SQL replication. Data access can be performed in either “Connected” (real-time) or “Disconnected” (synchronized) mode, depending on what is most appropriate for the given application. Connected mode, for example, is often required when end-users need to access information that changes regularly — such as a subscriber database or an inventory control application. In Connected mode, data is transmitted in real-time across the network. This real-time connectivity obviously makes application performance highly dependent on the network link to the end-user. However, network connectivity is important in Disconnected mode as well. In Disconnected mode, data is stored on the client machine until it connects to the network at which time the data is synchronized. During synchronizations a great deal of data may move between the client and the server — significantly impacting the network and/or the application itself. Application designers must take these factors into consideration to reduce the number of requests between the client and server and to avoid synchronization tasks where they cannot be supported by existing connectivity.

### **Client-Side Code Updates**

An important feature of .NET Win Forms is a new concept in client-side development referred to as “No Touch Deployment.” No Touch Deployment provides automated distribution and updating of smart client applications. For example, end-users can download and run a rate calculator on an as-needed basis. The end-user clicks a link on an HTML page, and the .NET assemblies are downloaded to the client machine and run.

No Touch Deployment has a fundamental reliance upon network connectivity. The ability to quickly execute this kind of download is again quite dependent on the performance of the network at every link between the end-user and the server where the code being retrieved resides.

### **Web Applications**

Developers can use ASP.NET to create Web applications and portals that run on a Web server and only transmit HTML to the client machine. Static components of these Web applications can be cached on the end-user’s machine, reducing the amount of data that needs to be transmitted. However, if the network connection is poor or over-utilized, then these Web applications cannot typically be used at all.

In some cases, these applications may be delivered over the Internet itself. This may require the application to be able to tolerate the latencies and performance issues that can occur as session traffic moves across multiple ISPs’ infrastructure. Also, if the application is being provided as a paid

service, developers must take into account the fact that customers' expectations for the performance of a paid service are very different from those of an internal end-user.

### **Portals**

Portal applications are a specialized form of ASP.NET applications. Windows SharePoint Services and SharePoint Portal Server are both built on .NET. SharePoint features web APIs for accessing SharePoint data and performing common tasks. These web APIs are ASP.NET Web Services that expose SharePoint to other applications.

Web APIs are highly dependent upon network connectivity — particularly if the SharePoint server is in a remote location. The other type of .NET functionality in SharePoint that developers need to consider with respect to network connectivity is that of SharePoint Web Parts. Web Parts are ASP.NET web pages that utilize additional features of the SharePoint portal UI framework. As such, the same network considerations that apply to ASP.NET pages apply to Web Parts.

In particular, some Web Parts may be cached while others may be dynamically loaded from the server every time the portal page is loaded. Web Parts may also be located on different remote servers. For example, financial dashboard Web Parts may be hosted by the finance department, while project management Web Parts can come from the Enterprise Project Management (EPM) server. If network connectivity is poor between the portal server and the location of various Web Parts, timeouts can occur and certain Web Parts may not be properly displayed.

### **Web Services**

Web Services are resources that can be used as needed, independent of location or host. Using SOAP over HTTP ties application performance and responsiveness to the underlying network's bandwidth and latency. Header-to-payload ratios, which can run relatively high in serialized XML formats, can also be a factor to consider in optimizing application performance.

Additionally, as part of the Web Services Enhancements (WSE), .NET now supports the transmission of file attachments. If large attachments are sent as part of a Web Service request and network connectivity is inadequate, performance problems may occur.

As with other distributed components, Web Services may involve the aggregation of data from different sources. The performance of such a Web Service will be dependent on the performance of the slowest network connection.

### **MS Office 2003 Applications**

With MS Office 2003 applications, developers can leverage network connectivity to automatically update spreadsheets, alert end-users about changes to a document, and automate a variety of business workflows. The use of the network for MS Office applications is similar to No Touch Deployment. Document templates and .NET assemblies are decoupled and can be easily deployed and redeployed to the end-user as appropriate. The assemblies could be on a local relative or absolute path, but are more likely to be stored on a central server and made available via a URL. Both the quality of

network connectivity and the size of these application assemblies are important considerations when designing and deploying these applications.

### Mobile Applications

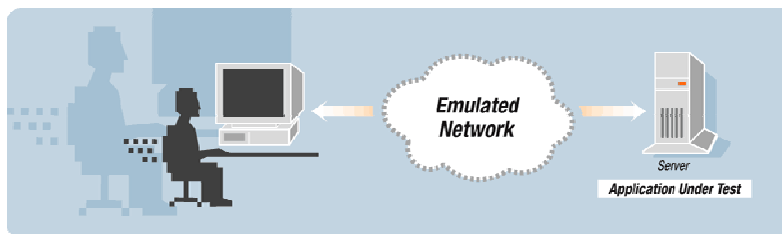
Mobile applications created using the .NET Compact Framework rely heavily upon network connectivity to grant access to enterprise data and real-time resources. .NET Compact Framework applications are similar to .NET Win Form applications in that both Connected/real-time and Disconnected/synchronized modes may be implemented. Mobile applications obviously raise serious connectivity issues. Those accessed via mobile phones, for example, will exclusively depend on wireless WAN connections where bandwidth is low and latency is often high. Applications must therefore be developed with this connectivity taken into consideration.

## Network Emulation: The Key to Trouble-Free .NET Deployment

It is precisely because .NET takes full advantage of today’s network-connected information resources and allows them to be applied so flexibly to various business challenges that it offers such a compelling value proposition. However, networking considerations make it abundantly clear that those responsible for developing and deploying .NET applications should take issues such as bandwidth, latency, packet errors and routing dynamics into account throughout the application development lifecycle. In fact, a clear understanding of .NET application performance on the production network is one of the keys to ensuring timely, cost-efficient delivery of high-performance, high-value services to the business.

Fortunately, there is a straightforward and effective way to discover and address potential networking problems with .NET applications and services early in the development process and throughout the application lifecycle: network emulation.

Network emulation enables developers, QA managers and network professional to create a testing environment that accurately replicates the production network on which .NET applications will run. By doing so, it allows them to discover potential bottlenecks as early as possible in the development process – before critical decisions are made about application architecture, server and client-side code, and/or interactions between components.



*Figure 2. Network emulation allows developers, QA, networking and performance managers to see how applications will perform under a variety of network conditions before deployment.*

An emulated test network allows developers to test and re-test applications to ensure that the “fixes” they’ve come up with actually resolve the problems they’ve found. And it allows subsequent tests to

be performed in order to determine if planned changes to an application or projected changes in network utilization will affect performance.

Emulation testing also enables a wide range of “what-if” scenarios to be played out in a safe, controlled environment – making it indispensable for capacity planning and other strategic tasks. In some cases, network emulation may even lead to the conclusion that a planned application simply cannot be implemented on the production network as it exists today. In such cases, network emulation provides the added benefit of allowing managers to determine if additional network capacity, or application or network re-configuration can solve the problem. Based on the cost of such a modification, they can then make a well-informed decision about whether there is, in fact, a good business case for doing so. If not, then the project can be aborted – long before IT’s limited financial and human resources have been wasted trying to do something that simply isn’t technically possible.

Essentially, emulation testing applies both specialized technology and proven best practices to the challenge of .NET development and deployment in three phases:

#### **Capture and Replication of Existing Network Infrastructure**

Effective pre-deployment testing of .NET applications requires more than just evaluating how they perform over a low-bandwidth link. To get an accurate test environment, it requires emulation of the real-life production network environment over which the .NET application will be deployed and end-users will access the application. This involves taking a “snapshot” of the production network, including parameters such as latency, bandwidth, packet errors, and routing dynamics – as well as current traffic levels already occupying those links.

#### **Code and Unit Testing, QA Testing, and Performance Optimization**

Once this testing environment is created, it can be used by developers, QA teams and network professionals to ensure the viability and performance of new or modified applications before their introduction into the production environment. This testing should begin early in the development cycle and be integrated into the code and unit testing that is performed on software modules, and repeated frequently along the way after the code is integrated to create the complete application – since the later problems are discovered, the more development work typically has to be re-done to fix it.

A flexible testing environment also allows IT professionals to test applications under a variety of end-user loads in a manner that accurately reflects potential conditions on the actual production network environment where those applications will run. This is a much more effective way to uncover shortcomings in application design than simple regression testing on a high-bandwidth, low latency LAN connection in a lab. In addition, iterative testing enables IT professionals to easily confirm that the “fixes” they’ve implement have, in fact, solved the problems they discovered – and that they haven’t unexpectedly created new ones. This is critical for avoiding unpleasant surprises at rollout.

### Baselining, Troubleshooting and Strategic Planning

In addition to its importance during QA and the pre-deployment phase, network emulation plays a key role in keeping .NET deployments trouble-free following deployment on the network. It can be used to ensure that planned changes or upgrades to .NET applications won't create any unexpected performance problems. If problems do occur, network emulation testing can be used to recreate those conditions in a lab setting where they can be analyzed and their root-cause more easily uncovered – and where proposed “fixes” can be validated before they're attempted in the production environment. And, as noted above, network emulation can be used to create a virtually unlimited range of “what-if” scenarios – allowing developers and network managers to project how the application will be affected by factors such as increased end-user loads and/or planned changes to the network's architecture.

The benefits of employing network emulation for pre- and post-deployment network testing are manifold. They include:

- Full assurance of performance, functionality, reliability and scalability *before* deployment on the production network
- Reduced development costs as a result of early problem detection
- On-time delivery of applications and faster time-to-deployment,
- Improved troubleshooting and root-cause analysis
- Improved change management processes
- Improved capacity planning

This process can be broadly applied to non-.NET applications as well, providing even greater return on investments in network emulation technology.

\* \* \*

Everyone knows what it's like to be stuck in his or her car and late for an important appointment because of a traffic jam, road construction or an accident. At times like that it doesn't matter how fancy a car you're driving. When you're stuck, you're stuck. That's why it's always smart to check on traffic conditions in advance – so you can plan an alternate route, leave the house early, or call ahead and push your appointment back.

Applications can get stuck in traffic, too. It doesn't matter how brilliantly they're designed from a functional point-of-view. If they aren't designed for their target network, or the network can't support them, they'll perform poorly and may even fail entirely.

The good news is that IT professionals can apply a disciplined engineering approach to application optimization, and discover these problems before they manifest on the production network and before development has proceeded too far. Traditional engineering requires that all existing and potential real-world operating conditions be factored into the application design and development from the outset. Network emulation provides these critical insights that are necessary to ensure the

networkability of even the most sophisticated .NET application and Web Services. By incorporating emulation into both pre- and post-production processes, IT organizations can ensure the success of their .NET deployments and optimize the ROI for all their development investments.

## About Shunra

Shunra provides solutions that empower organizations to address service level and performance concerns up front – before deployment. The Shunra Virtual Enterprise (Shunra VE) solution provides accurate, highly granular insight into how networked applications will function, perform and scale for remote end-users. It creates an exact replica of the production network environment, allowing IT professionals to safely develop, test and experiment with applications and infrastructure before rollout, and effectively plan for growth and change. With solutions tailored for networking and performance professionals, software developers, and quality assurance staff, Shunra VE facilitates collaboration across all IT disciplines – so IT organizations can quickly and more efficiently uncover and resolve problems before they impact the business. This results in more timely, higher quality and cost-efficient IT services, and the ability to “Deliver IT with Confidence”

### **Solutions for Any Enterprise**

More than 1500 customers, including hundreds of *Fortune* 1000 and Global *Forbes* 2000 organizations, from financial institutions to manufacturing companies, retail, energy, media companies, as well as independent hardware and software vendors and telecommunications service providers, have gained measurable returns from Shunra’s solutions. Among them are: 3M, Boeing, Cisco, Dow Chemical, EMC, FedEx, General Electric, General Motors, JPMorgan Chase, Kelly Services, Merrill Lynch, Motorola, Nestlé, Pitney Bowes, and Vodafone.

### **Corporate Information**

Shunra’s headquarters are located in New York City and Kfar Saba, Israel, with worldwide offices in Singapore, UK, The Netherlands and India. Shunra is also supported through a global network of channel partners.