Neuron Topics using TCP

Performance Tuning and Enhancements in release 2.5.10

© 2011 Neudesic. All rights reserved.

Marty Wasznicky

March 2011

Version 1.0

Overview

The 2.5.10 release of Neuron ESB introduces significant changes in the behavior and performance characteristics of Neuron services. Specifically affected are database related functions, Tcp Publishing service (and its associated client Tcp Channel) and other internal Neuron services. These enhancements were introduced to allow Neuron to achieve higher throughput, greater stability, increased concurrency and improved overall performance.

To understand the scope of this paper, it's important to note that Neuron ESB is entirely built on .NET 3.5 SP1 and uses the Windows Communication Foundation (WCF) for all internal and external communication. Even though most communication details have been abstracted and simplified within the Neuron ESB Explorer, there will be circumstances that require more advanced tuning and configuration of Neuron than what was previously available in earlier releases. Although this paper has a discernable focus Tcp Publishing service, it does describe the advanced tuning parameters now available to Neuron administrators which affect other internal services as well. Various enhancements related to performance and reliability included in this release are also addressed. Where possible, general guidance is provided on when and how to use many of the new tuning parameters.

Using Tcp with Neuron Topics

The Tcp Publishing Service is used to configure a Topic's Network Transport property. Neuron ESB is unique in that it allows users to configure Topic based publish/subscribe to use either one of a number of transports that best fit their quality of service (QoS) and delivery requirements. Out of the box, Neuron ESB supports TCP, Peer, MSMQ and BizTalk as possible network transports. Each Topic can be configured with a different network transport and can live side by side within the Neuron ESB runtime environment.

NOTE: Tcp based Topics are the default in Neuron ESB. Tcp provides a low latency, low overhead solution for customers who desire monitoring and control, but need neither the durability or transaction support provided by MSMQ based Topics. Most customers use Tcp based Topics when configuring request/response message patterns on the bus, or when deploying an MSMQ infrastructure is too burdensome.

When Tcp is selected for a Topic's Network Transport property, a new instance of Neuron's Tcp Publishing Service is instantiated to service all requests for that Topic. Within the Neuron ESB ecosystem, Neuron Parties (Subscribers or Publishers) are used to communicate with Topics and are commonly associated with Service and Adapter Endpoints. However, Neuron Parties can also be hosted in .NET applications residing on remote machines.

Neuron Parties (clients) that either publish or subscribe to information from a Tcp based Topic manage their communication with the server internally using the Neuron ESB Tcp Channel. The ability to host Neuron Parties in .Net applications on remote machines, coupled with the transitory (non-permanent) nature of Tcp based subscriptions, introduces a unique set of management requirements. For instance, when Tcp is used as the network transport for a Topic, Neuron must be aware of all publishers and subscribers connected to the bus, so it knows when and who to route messages to. Tcp has no durability facility so if a client is not connected, they would not be a potential subscriber for an inflight message.

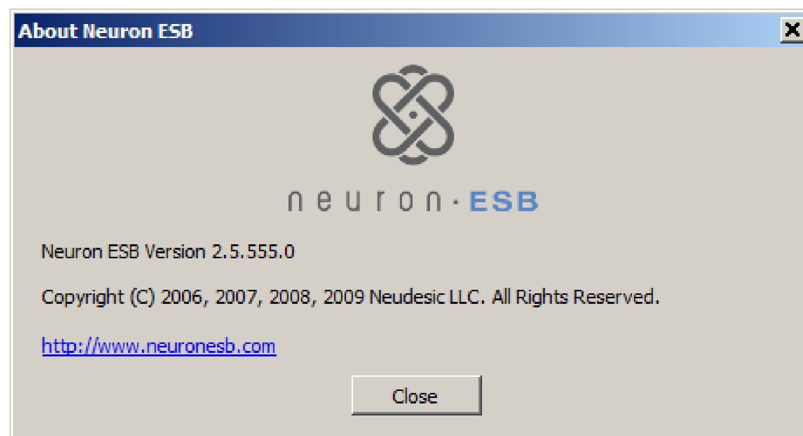
Internally, each Neuron Party that has a subscription to a Tcp based Topic will send a periodic "*Ping*" to the Neuron ESB Tcp Publishing Service. If, after a specified period of time, a ping is not received, the Neuron ESB Tcp Publishing Service will remove the Neuron Party from the list of potential subscribers.

This release offers a number of enhancements to the Tcp Publishing Service, its associated Tcp Channel as well as to other internal Neuron ESB Services which a Neuron Party may be dependent on. Collectively, this work greatly improves the performance and resource utilization of Tcp based Topics (CPU, Memory and Threads) as well as the number of remote concurrent Neuron Parties that can interact with a Topic. The areas covered in this paper are:

- Database Connection Configuration
- Ping Interval Communication
- Configuring Tcp Based Topics
- Configuring Neuron Services
- Connecting Neuron Parties
- WMI Client Configuration
- Start up and Shut down
- Performance Characteristics
- Tcp Optimization

What version is installed

Moving forward it's very important to know what version/build of Neuron is being used. The **2.5.10** release is represented by the internal build number **2.5.555.0**. Validating which build is currently installed can be determined by opening the Neuron ESB Explorer and selecting "*About Neuron*" from the "*Help*" menu. The following screen will be displayed:



Database Connection Configuration

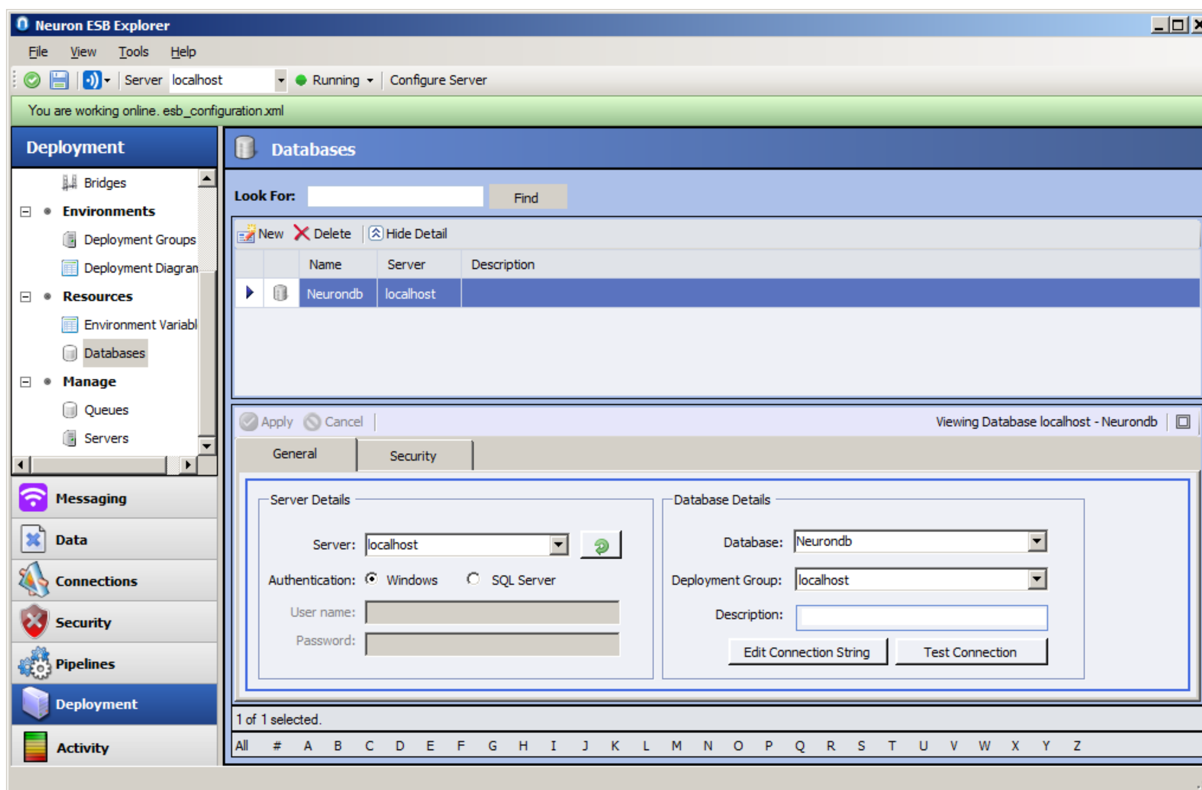
Within the Neuron ESB Explorer, databases may be created, configured and assigned to different deployment groups. Internally, Neuron ESB uses the database for Auditing (message tracking and failed message support) and Activity Session tracking. Both these are optionally configured by the user.

However the database is required for client and endpoint tracking when two or more Neuron ESB servers are configured to work as a server farm (where 2 or more servers share the same ESB Configuration).

In previous releases, the Neuron ESB Explorer did not allow for direct modification of the database connection string that the Neuron ESB runtime service uses. Normally, this would not be problematic. However, under conditions of heavy load where there may be 100's of connected clients, server farm configuration and/or heavy auditing (1000's of messages per second) or any combination thereof, exceptions indicating timeouts associated with retrieving or creating a database connection for the database connection pool could be generated. These exceptions may be found in the Windows Application and Neuron ESB Event logs, as well as the Neuron ESB logs. These exceptions occur because the default connection pool size is 100. Under these conditions it is possible for Neuron to exceed the default connection pool size.

In regards to auditing, this type of exception would cause auditing to fail, both for tracking and failed messages. However, this would normally not impact the ability for clients to publish or subscribe to a message. However, in server farm scenarios the database is required as a component of the "*Ping*" functionality. The "*Ping*" (similar to the mechanism used in the Tcp Publishing Service) allows each server to know which clients are connected to one another. Hence, these database exceptions would cause the "*Ping*" to fail, and the clients would be removed as eligible subscribers to the bus.

The 2.5.10 release corrects this by allowing users to modify the connection string directly through the Neuron ESB Explorer, by selecting the "*Edit Connection String*" command button located in the Database section of the Deployment menu as depicted below:



Selecting this command button will display a dialog box that permits overwriting the default connection string. The suggested values for a connection string follows:

> Data Source={server};Initial Catalog={database name};Integrated Security=True;Min Pool Size=5;***Max Pool Size=1000***;***Connect Timeout=15***;Load Balance Timeout=30;Network Library=dbmssocn

Due to the design of the Neuron database, it can comfortably handle 100s of active connections against it at any time. To determine how many active connections exist against a database, performance counters can be activated on the Neuron server (.Net has these performance counters disabled by default) by adding the following to the Neuron ESB Service configuration file (*C:\Program Files\Neudesic\Neuron ESB\esbService.exe.config*).

> *NOTE: Enabling performance counters will always have a negative impact on performance. These should not be enabled in production environments.*

```
<system.diagnostics>
  <switches>
    <add name="ConnectionPoolPerformanceCounterDetail"
         value="4"/>
  </switches>
</system.diagnostics>
```

Alternatively, a simple SQL query can be performed within the SQL Server Management Studio:

```
SELECT db_name(dbid) as DatabaseName, count(dbid) as NoOfConnections,
loginame as LoginName
FROM sys.sysprocesses
WHERE dbid > 0
GROUP BY dbid, loginame
```

Efficiently reusing and retrieving connections from the pool is directly impacted by the workload being done on the Neuron server, as well as any constraints or work being performed on the Sql Server.

Lastly, some modifications were made internally to our database schema and stored procedures to enhance the performance of the monitoring and reporting features that exist within Neuron. This will require that a new Neuron database be created to be used with this release. The SQL Server data import/export wizard can be used to transfer audit data from a previous version of the Neuron database, to the new one created with this release.

Ping Interval Communication

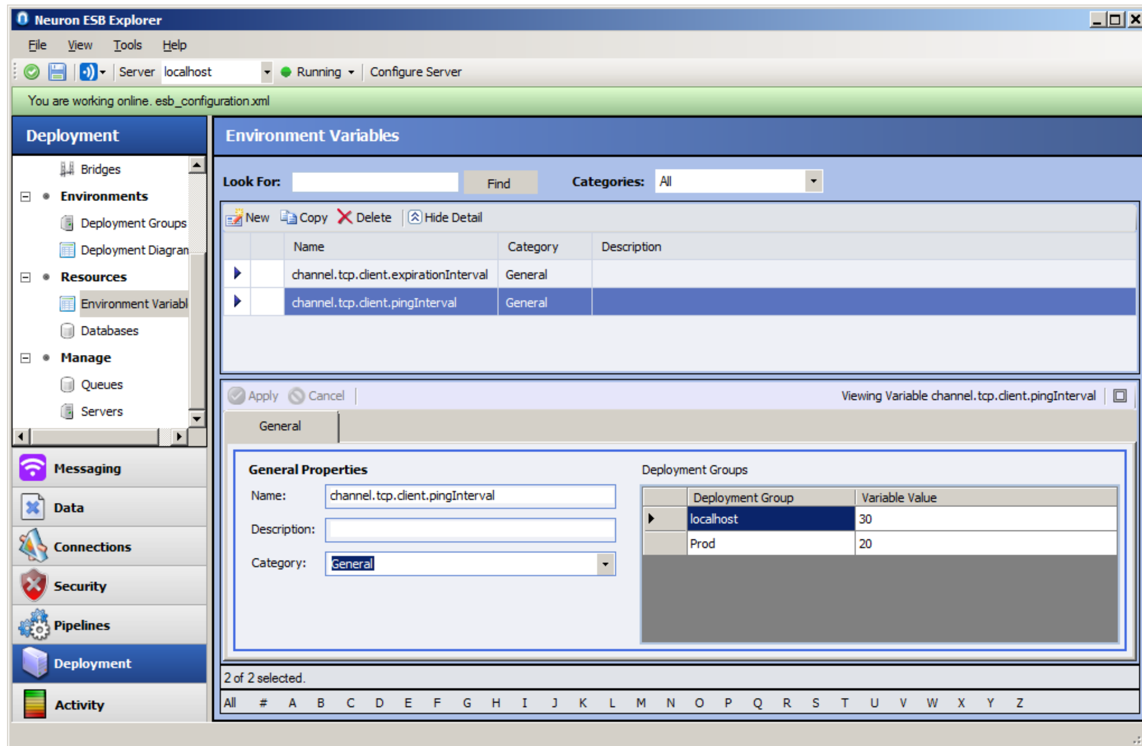
As described earlier in this paper, the *Ping* functionality is specific to the Tcp Publishing Service and Server Farm scenarios. It is used to accommodate the chance that a connected Neuron Party may fail to cleanly disconnect from bus. In regards to the Tcp Publishing service, the default time for each connected Neuron Party (whether remote, or associated with a service or adapter endpoint) to ping the Tcp Publishing Service is every 5 seconds. This is called the **Client Ping Interval**. The Tcp Publishing Service keeps track of every ping. If after 20 seconds, a ping reply is not received from a connected Neuron Party, that Neuron Party is assumed disconnected and removed from the eligibility list of recipients (those that can have messages routed to them). This 20 second timeout is called the **Client Expiration Interval**.

Under normal conditions, this 5/20 interval is not an issue. However, if a server is running at near maximum capacity, has many connected clients or, is experiencing high load conditions, these intervals should be increased to decrease unnecessary “chatty” traffic. This will increase overall performance and throughput.

In previous releases, we provided environmental variables which allowed users to override these values. Specifically, the following environmental variables are defined as follows:

channel.tcp.client.expirationInterval
channel.tcp.client.pingInterval

The advantage of defining these within the Environmental Variables section of the Neuron ESB Explorer are that different values could be assigned to different deployment groups as depicted below:



Unfortunately, a bug prevented the **channel.tcp.client.pinginterval** variable value from being used to control the Neuron Party ping interval. This has been corrected in the 2.5.10 release. Under high load conditions, it is recommended that these values be set to the following to decrease the chatter:

> **channel.tcp.client.expirationInterval = 120**
> **channel.tcp.client.pingInterval = 30**

> *NOTE: Regardless of what values these are set to, it is best practice that they are set at a 1 to 3 ratio or greater.*

An additional issue manifested itself under high load conditions. The issue would cause the individual Neuron Party as well as the Neuron ESB Server to timeout when executing their respective pings. Under these conditions, the ping did not actually timeout, it simply failed to fire. However, Neuron would erroneously report the error as a timeout. If this occurred it would result in the client being removed from the recipient list by the server, essentially treating it as if it were disconnected. The client would then automatically go in an offline state, but then immediately try to reconnect. If these failures continued to occur, it would result in a repeated, offline - reconnect scenario due to the resiliency built into the Neuron ESB client stack.

We isolated the cause of the issue to a bug in the .Net Framework, specifically how it allocates and manages IO threads. The 2.5.10 release addresses this issue by changing the mechanism for the *Ping* functionality to ensure that they always fire when expected and any errors raised are accurately handled and reported.

Configuring Tcp Based Topics

The 2.5.10 release of Neuron introduces advanced enhancements of the Tcp Publishing service that greatly extends its ability to scale to 100's of connected clients as well as provide greater concurrency in both datagram and request/response message patterns.

In previous releases customers would normally not notice aberrant behavior. However some could experience unexpected memory and thread growth if running continuous medium to heavy load over a protracted period of time. In time the memory growth could well exceed 2GIG and threads could climb to 4,000, at which point the Neuron ESB server could become unresponsive. In time, the Neuron ESB service would generate exceptions in the form of Socket Aborts and Timeouts.

This behavior was caused by a combination of default settings Neuron used internally (**Max Concurrent Calls** was set to a default value of Int32.MaxValue), as well as the design of the resource manager that previous releases of Neuron depended on to manage internal resource allocation and connections to Neuron Parties.

To resolve the previous behavior, a new resource manager has been introduced with the 2.5.10 release. Within the new resource manager, thread allocation and locking semantics have also been refined to allow for greater concurrency. Additionally, we introduced a number of tuning parameters located on the Networking tab of the Topic configuration section of the Neuron ESB Explorer as depicted below:

Many of the default values were raised (**Listen Backlog** from its previous default of 10), while others were lowered (**Max Concurrent Calls** reduced to 1,000 from its previous default of Int32.MaxValue). Almost all the new tuning parameters are specific to WCF and can be found here:

http://msdn.microsoft.com/en-us/library/ms731343.aspx

Neuron exposes the essential ones below. Although some of the parameters were previously available to be adjusted, many are *NEW*. The new tuning parameters are highlighted in ***blue bold italic***:

Name	***Default Value***
Client Pool Size	***100***
Client Port Base	61007
Client Port Range	1000
Close Timeout	***60***
Large Message Optimization	False
Listen Backlog	***1000***
Log No Recipients	***True***
Max Buffer Pool Size	***524288***
Max Concurrent Calls	1000
Max Connections	***1000***
Max Received Message Size	***2147483647***
Open Timeout	***60***
Ordered	False
Reliable	False
Secure	False
Send Timeout	***60***
Service Port	50010

Under normal circumstances, the only parameters which should be changed are **Max Concurrent Calls** and **Max Connections**. These will have an immediate impact on the number of threads and memory allocated to the ESB Service to support concurrency. In previous releases, the default value set for **Max Concurrent Calls** was Int32.MaxValue. However, a high **Max Concurrent Calls** value can negatively impact performance because of the extra work being done to allocate and manage more threads than are needed. In fact, if set too high, overtime both private bytes and threads allocated to the Neuron ESB service will grow to unstable levels. A comfortable default value to begin with is **16 * Processor Count**. Through testing, the correct value can be determined for the specific Neuron implementation. Since many of our internal services are datagram calls, **Max Concurrent Calls** can be configured quite low and yet still achieve a fairly high concurrency factor.

NOTE: The Neuron ESB Auditing service can now be configured to either function asynchronously (default) or purely synchronously in the 2.5.10 release. With the latter, if an error occurs during auditing, the message will not be submitted to the bus for processing.

Max Connections on the other hand controls the maximum number of connections to be pooled for subsequent reuse on the client and the maximum number of connections allowed to be pending dispatch on the server. Each of these properties uses a default value of 10, which was also the default in previous releases of Neuron. In the 2.5.10 release of Neuron, the default value has been set to 1,000. Modifying this upwards can improve performance when working with many hundreds of connected Neuron Parties, or when using a **Client Pool Size** greater than 1. This will allow connections to remain cached, rather than having to be created on demand. However, if working with only a small number of Neuron Parties, lowering the default value may be appropriate. If this setting is too low, it can cause Open Timeout and Abort errors to occur like those below:

> The socket connection was aborted. This could be caused by an error processing your message or a receive timeout being exceeded by the remote host, or an underlying network resource issue. Local socket timeout was '00:01:00'.
>
> Inner Exception: An existing connection was forcibly closed by the remote host
>
> The open operation did not complete within the allotted timeout of 00:01:00. The time allotted to this operation may have been a portion of a longer timeout.
>
> The socket transfer timed out after 00:00:59.9989999. You have exceeded the timeout set on your binding. The time allotted to this operation may have been a portion of a longer timeout.
> Inner Exception: A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond

The specific Neuron implementation should be tested to determine the right combination of parameter values to use that provide the best balance between performance, concurrency and resource utilization.

Under conditions of high load and stress there are 4 parameters which will mostly like need to be changed:

- **Max Concurrent Calls**
- **Send Timeout**
- **Listen Backlog**
- **Max Connections**

In the case of **Max Concurrent Calls**, the default value would mostly likely need to be reduced, perhaps to a few hundred. In contrast, **Max Connections** and **Listen Backlog** may need to be increased to several thousand. **Send Timeout** may need to be increased as internally, that value is also used to initialize the **Operation Timeout** for the WCF Channel, which collectively is used for associated call backs. Lastly, sometimes the **Send Timeout** may need to be increased due to lack of idle IO threads immediately available to service the internal requests. Tuning IO threads to increase and optimize concurrency will be addressed later in this paper.

As general guidance, if the following inner exception is written in either the control, management, configuration, or auditing Neuron logs, it usually indicates that Neuron cannot create any new Tcp Socket connections. This is very common in scenarios where there are more than 50 parties configured

or remotely attempting to connect. In this case, you should raise the **Listen Backlog. Listen Backlog** is a socket-level property that describes the number of "*pending accept*" requests to be queued. If the listen backlog queue fills up, new socket requests will be rejected. Neuron Services may handle a large number of socket connections when working with 100's of connected Neuron Parties. A larger backlog value will prevent client connection requests from being dropped:

> *System.Net.Sockets.SocketException:*
> *An existing connection was forcibly closed by the remote host*

In contrast, if a Timeout exception similar to those below is found, it usually means that either **the Send Timeout** or **Max Concurrent Calls** need to be changed. As a general rule, **Send Timeout** should never exceed 5 minutes. However, if **Max Concurrent Calls** is configured too low or too high it may cause a Timeout issue. Testing to find the right combination is essential.

> *System.TimeoutException: Sending to via net.tcp://localhost:50016/ESBTcpPubSub/ timed out after 00:05:00. The time allotted to this operation may have been a portion of a longer timeout*

> *Exception: The socket transfer timed out after 00:05:00. You have exceeded the timeout set on your binding. The time allotted to this operation may have been a portion of a longer timeout.*

There will be very few circumstances that will ever necessitate changing the value of the Open or Close Timeout properties.

> *NOTE: After testing 600 connected Neuron Parties, 300 of which were publishing at a steady rate, the other 300 subscribing to all messages, it was not necessary to increase either the Open or Close Timeout of the Tcp Publishing service.*

The Tcp Publishing service has many other tuning parameters exposed. One parameter that will need to be changed as load and concurrent connections increase will be the **Client Pool Size.** The **Client Pool Size** is a Neuron specific tuning parameter introduced as a way to optimize communication to Neuron Parties when only a handful of them are actively configured. The **Client Pool Size** determines how many WCF Connections will be made to a connected Neuron Party. When there are only a handful of Neuron Parties, performance can be improved by up to 10% using the default value. However, as you configure more Neuron Parties, performance degrades. As you increase the number of connections to a Neuron Party concurrency is increasing, but so are resources to manage it. To support this concurrency, the Max Concurrent Calls and Listen Backlog values will also need to be modified upwards. However, at a certain point, the benefits of supporting multiple connections to a single Neuron Party becomes detrimental to scale and performance. For example, if Pool Size is left at its default of 100, and you attempt to connect 100 Neuron Parties, Neuron could attempt to create up to 10,000 internal WCF connections. In fact, under normal conditions, this may not be successful, however, if the **Client Pool Size** is set to 1 (this is essentially identical to setting the **Ordered** property to True), several 100 (600 were tested) Neuron Parties can be easily connected to publish and subscribe to messages with a fraction of the resources.

It is strongly recommended that the **Client Pool Size** value be changed to either 1 or 2 when actively configuring more than a dozen Neuron Parties.

Request/Response Message Patterns

It is very important that the **Max Concurrent Calls** and **Listen Backlog** be tuned appropriately when employing request/response message patterns over the Tcp Publishing service. Determining the best setting will usually depend on 2 factors:

1. Number of concurrent calls your service receives.
2. Execution time for each call.

For example, if 10 request/response calls are received per second and the execution time for the service method is 2 seconds. While the service is processing the first call we will receive 19 other calls.

The **Max Concurrent Calls** setting in this case would be 10 x 2 = 20. In a live scenario you might want to add a 20% buffer to this and make it 24.

As in all cases, it's extremely important to test the specific scenario to determine the best combination of tuning parameters to adjust.

Increase the number of idle IO threads in the thread pool

If after the tuning parameters have been modified it's found that the response times remain either lower, or erratic (interspersed with long wait times), then the calls to the Tcp Publishing service may be getting queued because there are not enough IO threads in the ThreadPool to handle the requests.

Generally, if the calls are not taking longer to process by the target service (i.e. Neuron Service Connector) and very high CPU utilization is not seen then the increase in response time is likely the result of the requests being queued while WCF waits for a new IO thread to be made available to handle the request.

WCF uses managed IO threads from the CLR ThreadPool to handle requests and by default; the ThreadPool creates one IO thread for each CPU/Core. So on a single core machine that means you only have **ONE** available IO thread to start with, and when more IO threads are needed they're created by the ThreadPool (an expensive operation) on demand, with a delay:

> "The thread pool maintains a minimum number of idle threads. For worker threads, the default value of this minimum is the number of processors. The *GetMinThreads* method obtains the minimum numbers of idle worker and I/O completion threads.
>
> When all thread pool threads have been assigned to tasks, the thread pool does not immediately begin creating new idle threads. To avoid unnecessarily allocating stack space for threads, it creates new idle threads at intervals. The interval is currently half a second, although it could change in future versions of the .NET Framework.

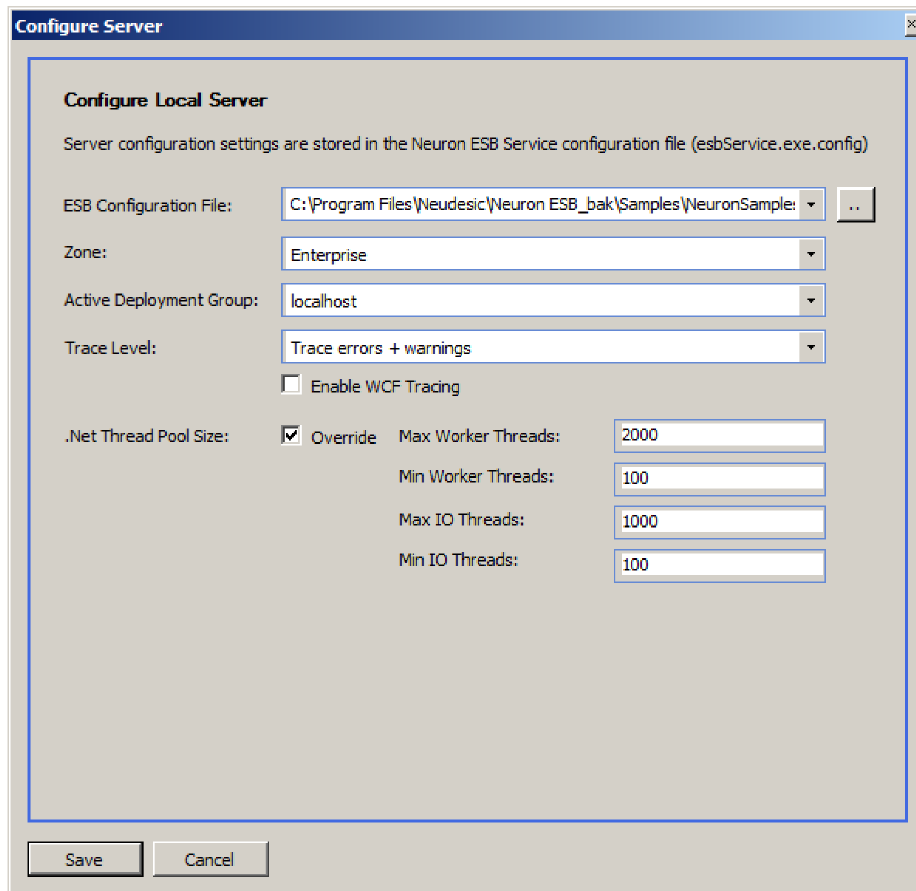
If an application is subject to bursts of activity in which large numbers of thread pool tasks are queued, use the *SetMinThreads* method to increase the minimum number of idle threads. Otherwise, the built-in delay in creating new idle threads could cause a bottleneck."

The lack of adequate IO Threads on startup can also cause delays in connecting large number of Neuron Parties (clients) concurrently connecting to the Neuron ESB service. It can also contribute to timeouts related to the initial connection and publication of messages from a large number of concurrent clients.

However, raising the *MinIOThreads* setting in the ThreadPool doesn't work as expected in .Net 3.5 because of a known issue with the ThreadPool. Microsoft has released a hotfix for this that you will need to install:

http://support.microsoft.com/kb/976898
http://connect.microsoft.com/VisualStudio/Downloads/DownloadDetails.aspx?DownloadID=32699

Once the hotfix has been installed, the default ThreadPool settings can be easily overridden by selecting the "*Configure Server*" toolbar button on the Neuron ESB Explorer to display the dialog box as depicted below:



Configuring Neuron Services

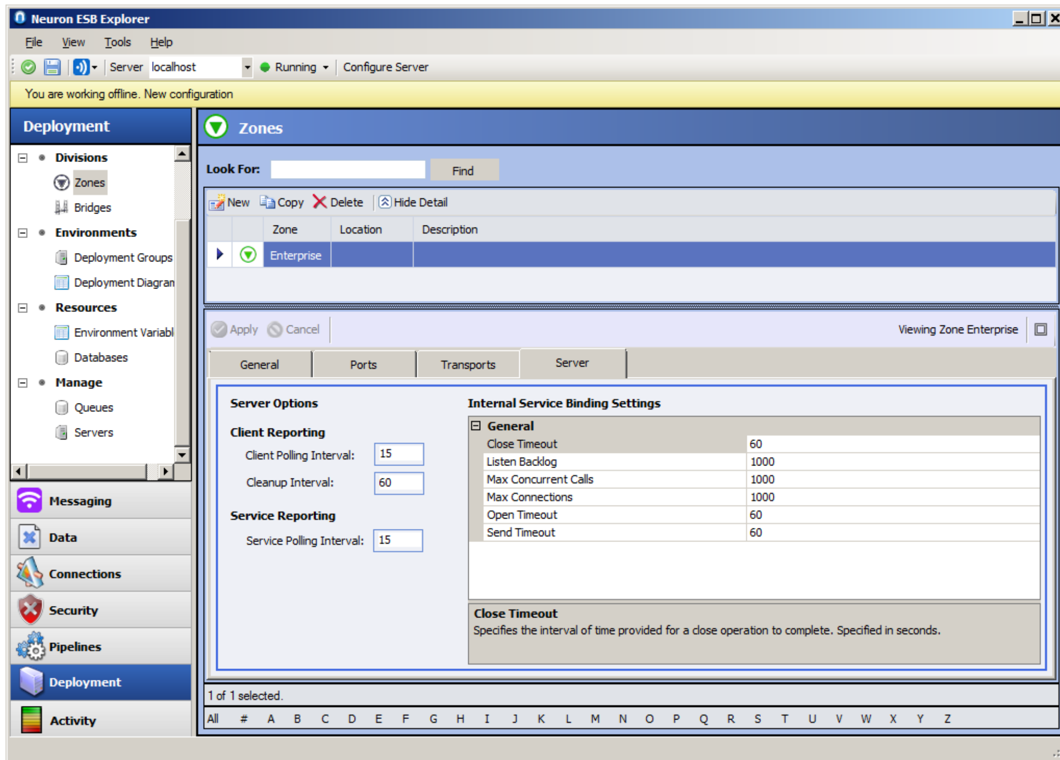
The Neuron ESB runtime uses a number of internal services for everything from management, control, configuration to auditing. Neuron Parties also access and connect to these services at runtime. For example, when a Neuron Party calls *Connect()*, several things happen:

- The Neuron Configuration Service is called to retrieve the Party's respective configuration information. This information contains the details of what Topics to connect to, how to connect, how each Topic is configured, retrieves the pipelines associated with the Party, etc.
- The Neuron Party then connects to each Topic using its respective Neuron Channel. Depending on the Topic configuration and its transport, this may entail calling into one or more services. For Tcp based Topics, this would mean calling into the Tcp Publishing Service instance associated with the Topic.
- At intervals configured within the Neuron ESB Explorer, the Neuron Party will call the Management Service to upload client Activity Session statistics.
- Additionally, at runtime, if either the Topic is configured for Auditing, or there is an Audit pipeline step within a pipeline assigned to the Party, or a failure occurs, The Party will call the Auditing Service

One of the benefits of an ESB is that internal services themselves are exposed as "*Services*". Neuron is fairly pure in that respect. However, in previous releases tuning parameters for these internal services, all of which are all based on the WCF Net.Tcp transport, were not exposed

Under normal conditions, this was not an issue. However, under conditions of medium to heavy load, failures could be experienced. These failures would present themselves in the form of timeouts or socket exceptions. Other symptoms would be reduced capacity and scale, coupled with erratic CPU utilization and, over time, increasing memory usage as well as extremely high thread counts. If left un-monitored, the ESB Service could become unresponsive.

In the 2.5.10 release, we introduced a number of tuning parameters located on the Server tab of the Zone configuration section of the Neuron ESB Explorer as depicted below. These are *NEW* settings which were not previously exposed:



Many of the default values were raised (**Listen Backlog** from its previous default of 10), while others were lowered (**Max Concurrent Calls** reduced to 1,000 from its previous default of Int32.MaxValue). The new tuning parameters are all specific to WCF and can be found here:

http://msdn.microsoft.com/en-us/library/ms731343.aspx

Neuron exposes the essential ones below:

Name	***Default Value***
Close Timeout	60
Listen Backlog	1000
Max Concurrent Calls	1000
Max Connections	1000
Open Timeout	60
Send Timeout	60

The information within the "*Configuring Tcp Based Topics*" section of this paper regarding conditions and effects surrounding the modification of some of the parameters listed above such as **Max Concurrent Calls**, **Max Connections, Listen Backlog** and **Timeouts** are applicable to the optimization of Neuron's internal services.

Connecting Neuron Parties

Timeouts can happen due to insufficient values configured for either the Topic's network setting (such as when using Tcp based topics i.e. Open/Close/Send Timeouts) or those settings located on the Server tab of the Zone configuration. However, a timeout can also occur when the client is initially attempting to connect to retrieve its configuration from the Neuron server. This happens before the client attempts to connect to any other service within Neuron, including the auditing, management or Topic related publishing service. The default timeout for obtaining configuration from the Neuron server is 60 seconds. In scenarios where there are 100's of competing clients trying to connect to Neuron at the same time, timeouts to the Neuron Configuration Service may occur. The value for these timeouts can be increased by setting the following properties before the *Connect()* method is executed on the Neuron Party:

```
Neuron.Esb.EsbService.ESBServiceFactory.BindingSettings = new Neuron.Esb.EsbService.BindingSettings();
Neuron.Esb.EsbService.ESBServiceFactory.BindingSettings.OpenTimeout = number of seconds;
Neuron.Esb.EsbService.ESBServiceFactory.BindingSettings.SendTimeout = number of seconds;
```

The **OpenTimeout** and **SendTimeout** property should not be set higher than 120 seconds. If Timeout errors are still occurring, attempt to increase the **Max Concurrent Calls** in the Server tab of the Zone Configuration within the Neuron ESB Explorer.

WMI Client Configuration

Internally, Neuron ESB exposes WMI performance counters (using a dynamic WMI provider) for every connected Neuron Party. However, when attempting to connect 100's of Neuron Parties, the memory limitations of WMI are encountered and will result in the following exception being recorded in the Application event log (provided client logging is enabled):

```
Log Name:      Application
Source:        Neuron ESB Client Context
Date:          3/20/2011 11:33:09 PM
Event ID:      5000
Task Category: None
Level:         Warning
Keywords:      Classic
User:          N/A
Computer:
Description:
Event Info: Failed to initialize performance counters. Custom counters file view is out of memory.
```

This can have a negative impact on overall performance of the system when working with hundreds of configured Neuron Parties. When this error is encountered, the memory limit may be increased by

following the guidance in the section entitled, "*Increasing Memory Size for Performance Counters*" in the following article.

http://msdn.microsoft.com/en-us/library/ms735098.aspx

Alternatively, in release 2.5.10, WMI performance counters for Neuron Parties can be disabled on the General tab of the Zone Configuration section of the Neuron ESB Explorer. If working with hundreds of Neuron Parties, it is recommended that this is disabled. Statistics for any individual connected Neuron Party can be viewed within the Active Sessions display located in the Activity section of the Neuron ESB Explorer.

Start up and Shut down

In previous releases of Neuron users would sometime experience the following when working with many configured Neuron Parties and or when experiencing load conditions. The term "*configured*" in this context refers to Neuron Parties either remotely connecting to the Neuron ESB Service or configured in either service or adapter endpoints:

- Unusually long start up times of the Neuron ESB Service
- Unusually long shut down times for the Neuron ESB Service
- Neuron ESB Service appears to hang when shutting down
- Tcp Topic appears to hang when be set to Stop or Restart in Endpoint Health monitor

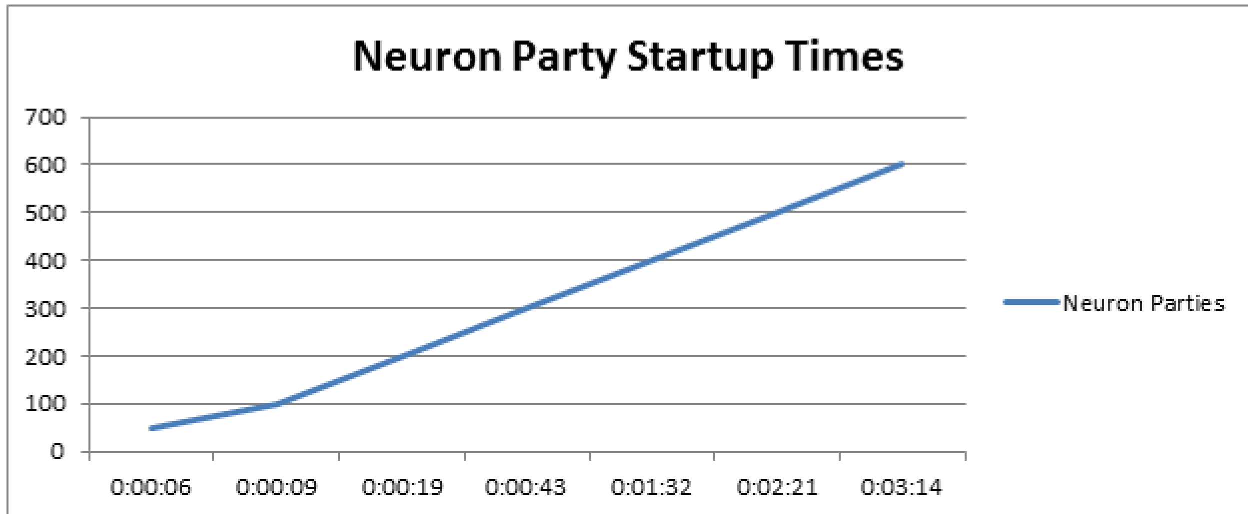
Several issues with the resource manager that previous versions of Neuron depending on was a contributing factors causing these conditions to occur. The core of these issues involved the individual startup times (instantiate a Neuron Party and call *Connect()*) of Tcp based Topics and the Parties that subscribed to them.

These issues are fixed in the 2.5.10 release with the introduction of the new resource manager.

Below is a chart of startup times measured on a laptop running Windows 7 64 bit, 8 GIG RAM/dual CPU/Quad Core. The Neuron Party tested had one Topic as the subscription and no pipelines attached. The Horizontal scale is the time in hours, minutes and seconds, while the vertical scale represents the number of Neuron Parties that were instantiated and connected to the Neuron ESB server.

NOTE: Startup times will be affected by other factors as well, including:

- *How many Topics the Party is associated with*
- *If pipelines are associated with the Party, and how many*



Performance Characteristics

A number of preliminary tests were performed to determine the characteristics and tuning parameter values for two scenarios. The first involved supporting up to 600 connected Neuron Parties. 300 of which would be publishing messages at a sustained rate, while the other 300 subscribed to ALL messages published to Neuron. The other involved simple request/response service calls routed through Neuron ESB and its relative performance when compared to calling the service directly.

All testing was performed on one Windows 7, 64 bit Laptop, 8 GIG RAM/dual CPU/Quad Core. The machine also ran SQL Server 2008, MSMQ as well as a host of other desktop programs and browsers. All agents as well as simulators used for load testing were hosted on this machine. No remote machines were used.

Concurrent Neuron Parties

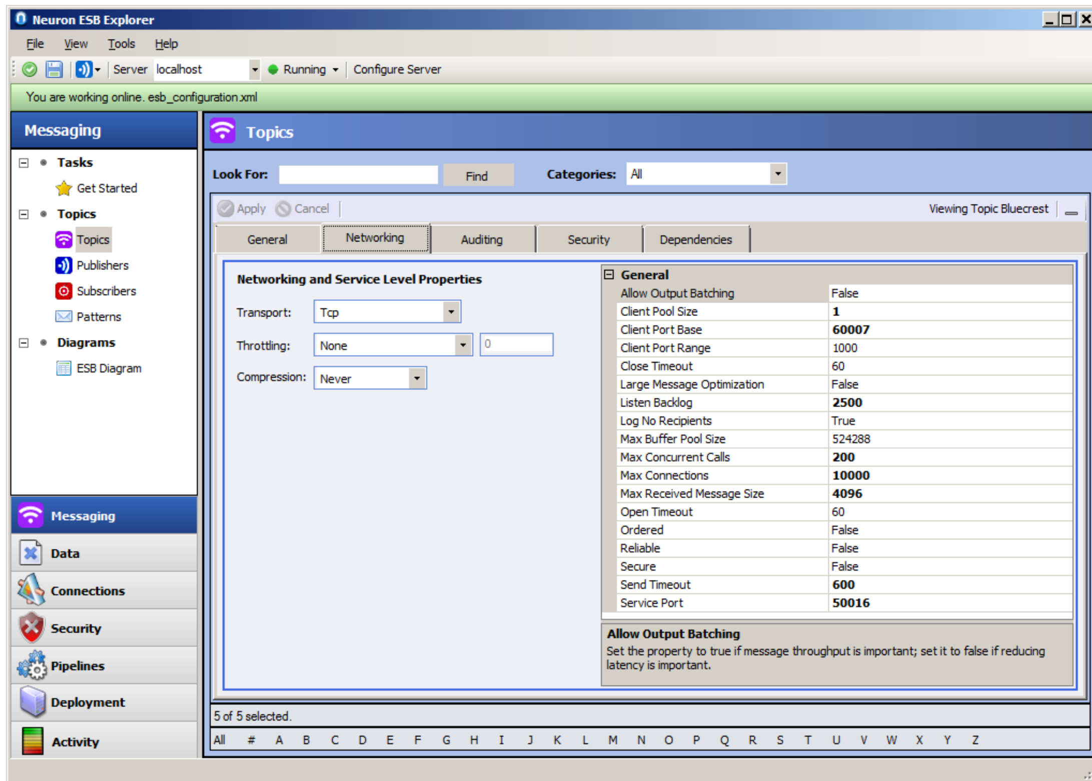
Many of the enhancements introduced with the 2.5.10 released were designed to not only increase performance and decrease resource utilization, but also increase our ability to support an increasing number of remotely connected Neuron Parties. In this test, a simulator was used to launch 600 Neuron Parties, each on their own thread. Each Neuron Party would be instantiated, Connect to the bus and then either wait to receive messages (OnReceive event) or continuously publish to the bus at a sustained rate of 12 messages/sec.

The Active Sessions report within the Neuron ESB Explorer (depicted below) was used to monitor when all Neuron Parties were connected to the bus (total number displayed on lower right corner). The Activity Sessions report displays the time the connection was made, last update; messages sent and received rates and other relevant statistics.

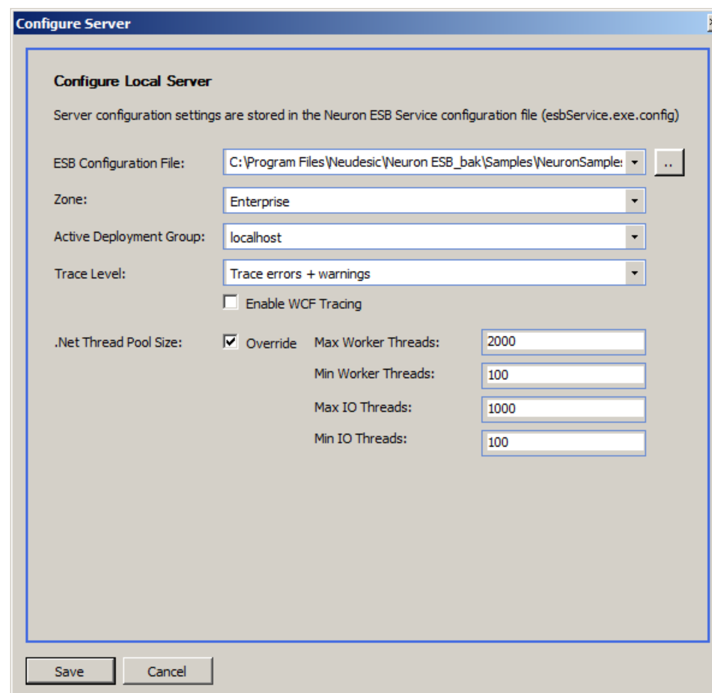
The Endpoint Health Monitor within the Neuron ESB Explorer was used to monitor the gradual increase in throughput as well as to detect exceptions if they occurred.

Neuron Configuration

For testing concurrent Neuron Parties, the Neuron Topic was configured as follows:



Additionally, the minimum IO threads were overridden with the following values using the Configure Server feature of the Neuron ESB Explorer. Neuron Tracing was set to Errors and Warnings only:



Observations

After all 600 connected clients ran continuously for 8 hours, 300 of which were publishing at a steady rate while each of the other 300 were subscribing to an aggregate of all published messages, CPU remained steady between 25-35%. Private Bytes for the ESB Service would move between 400-800MB and threads never went above 275.

No errors were experienced.

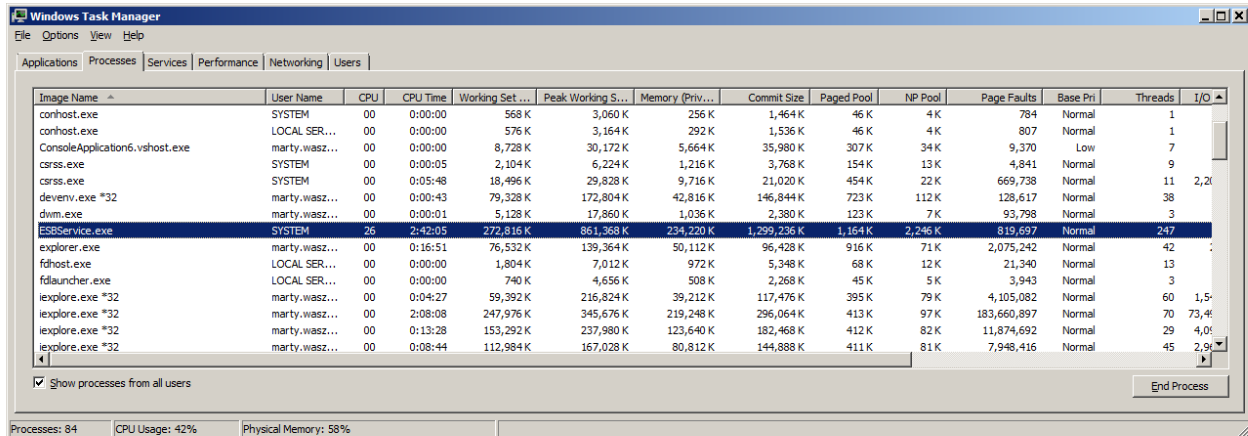


Image Name	User Name	CPU	CPU Time	Working Set ...	Peak Working S...	Memory (Priv...	Commit Size	Paged Pool	NP Pool	Page Faults	Base Pri	Threads	I/O
conhost.exe	SYSTEM	00	0:00:00	568 K	3,060 K	256 K	1,464 K	46 K	4 K	784	Normal	1	
conhost.exe	LOCAL SER...	00	0:00:00	576 K	3,164 K	292 K	1,536 K	46 K	4 K	807	Normal	1	
ConsoleApplication6.vshost.exe	marty.wasz...	00	0:00:00	8,728 K	30,172 K	5,664 K	35,980 K	307 K	34 K	9,370	Low	7	
csrss.exe	SYSTEM	00	0:00:05	2,104 K	6,224 K	1,216 K	3,768 K	154 K	13 K	4,841	Normal	9	
csrss.exe	SYSTEM	00	0:05:48	18,496 K	29,828 K	9,716 K	21,020 K	454 K	22 K	669,738	Normal	11	2,2
devenv.exe *32	marty.wasz...	00	0:00:43	79,328 K	172,804 K	42,816 K	146,844 K	723 K	112 K	128,617	Normal	38	
dwm.exe	marty.wasz...	00	0:00:01	5,128 K	17,860 K	1,036 K	2,380 K	123 K	7 K	93,798	Normal	3	
ESBService.exe	SYSTEM	26	2:42:05	272,816 K	861,368 K	234,220 K	1,299,236 K	1,164 K	2,246 K	819,697	Normal	247	
explorer.exe	marty.wasz...	00	0:16:51	76,532 K	139,364 K	50,112 K	96,428 K	916 K	71 K	2,075,242	Normal	42	
fdhost.exe	LOCAL SER...	00	0:00:00	1,804 K	7,012 K	972 K	5,348 K	68 K	12 K	21,340	Normal	13	
fdlauncher.exe	LOCAL SER...	00	0:00:00	740 K	4,656 K	508 K	2,268 K	45 K	5 K	3,943	Normal	3	
iexplore.exe *32	marty.wasz...	00	0:04:27	59,392 K	216,824 K	39,212 K	117,476 K	395 K	79 K	4,105,082	Normal	60	1,5
iexplore.exe *32	marty.wasz...	00	2:08:08	247,976 K	345,676 K	219,248 K	296,064 K	413 K	97 K	183,660,897	Normal	70	73,4
iexplore.exe *32	marty.wasz...	00	0:13:28	153,292 K	237,980 K	123,640 K	182,468 K	412 K	82 K	11,874,692	Normal	29	4,0
iexplore.exe *32	marty.wasz...	00	0:08:44	112,984 K	167,028 K	80,812 K	144,888 K	411 K	81 K	7,948,416	Normal	45	2,9

Request/Response

Neuron ESB provides both service routing and service mediation capabilities. As such, Neuron can host its own service endpoints (represented in the Neuron ESB Explorer as *Client Connectors*) for customer applications to submit their requests to. These requests are then intelligently routed to the appropriate existing back end service, represented in the Neuron ESB Explorer as a *Service Connector*. If a response is returned by that service, Neuron forwards it back to the *Client Connector* and the calling customer application.

By default, Neuron uses its Topic based Publish/Subscribe engine as the abstraction layer between the incoming requests received by the Client Connector and later routed to the Service Connector, as well as the returned response from the Service Connector, ultimately routed back to the Client Connector. The advantage of this is that incoming requests can be easily re-routed to other services or different subscribers, transparent to the customer application. Protocol and message mediation is managed by the Neuron service and additional processing logic can be inserted at any point without the need to modify existing customer applications or services.

> *NOTE: For increased performance **Direct Routing** can be used to circumvent the abstraction layer of the Topic based Pub/Sub engine. This is done by adding the following in a Code Step within the pipeline associated with the Client Connector:*
>
> *context.Data.SetProperty("DirectConnect", "To", <Name of Service Connector>);*

Expectedly, there will always be some overhead when inserting an abstraction layer between two or more service endpoints. This overhead usually can increase relative to concurrency and throughput.

However, the overhead incurred is usually compensated by the additional services offered by the bus such as:

- Accelerated development and deployment of services
- Configuration of new services
- Routing and intermediary services
- Mediation and protocol translation
- Process development and design
- Integration platform
- Event and service management

To better understand the performance characteristics testing was performed to compare the following:

- Calling a service directly
- Calling a service through Neuron ESB's Client Connector (Neuron Topic Routing)
- Calling a service through Neuron ESB's Client Connector (Direct Routing)

Sustainable Throughput

The key to performance testing is determining the correct balance between throughput and concurrency for a given set of hardware and resources. The goal of course is to determine the maximum sustainable throughput that a system employing Neuron ESB can sustain indefinitely, without seeing performance degradation over time. Consider that conducting a 60 second or 5 minute test is typically far less meaningful then conducting the same test that extends to 1 hour, 8 hours or 1 day. Tests that appear to exceed thresholds within 1 to 5 minutes can steadily degrade if allowed to run longer due to the back pressure created by machine resources or other services/processes that may be involved in the service call.

Many other, sometimes external factors can affect the sustainable maximum throughput of a system such as:

- Available server resources
- Auditing
- Encryption
- Compression
- Size of message
- Number of subscribers
- Business processing (pipelines) requirements
- External service performance
- Error conditions

The mediation of each of these factors can vary since each carries its own performance profile when subjected to duress. For instance, if Auditing is employed, I/O optimization (among other things) may be required on the SQL Server.

A critical step in determining the maximum sustainable throughput of the Neuron ESB System is to first understand the performance thresholds the system needs to support. These should not be arbitrary numbers since they directly impact the scale out/up strategy. The performance thresholds are defined by the load characteristics expected to be placed on the system. These are usually defined in terms of:

- Average Response Time (RT) expected
- Peak RT acceptable
- Median RT
- Maximum concurrent users
- Average concurrent users
- Median concurrent users
- Average Transactions (request/response calls)/per second (TPS)/per concurrent user
- Peak TPS expected

For simplification purposes, if we knew that the Peak RT requirement was .5 seconds, with an peak TPS of 100/per second per concurrent user, testing could be performed to determine exactly how many concurrent users could be supported given a specific machine/set of resources, while maintaining an Peak RT under .5 seconds and a peak TPS at 100+/per second. This information would be used as an input to determine the appropriate scaling strategy for Neuron to meet greater concurrency, if required.

Scaling Strategies

Scaling up or out to achieve specific performance thresholds will vary depending configuration, resource utilization of the machine, as well as the bottlenecks observed during testing. Things that may be considered and when are:

Tuning Parameters

Depending on the transport (i.e. Tcp) and routing option used (Neuron Topic Routing vs. Direct Routing), the tuning parameters in either the binding or in the property grid of the Networking tab for the Topic can be modified and tested to increase overall performance.

Adding additional CPUs

If during testing, the CPU is observed peaking at 75-80% adding additional CPUs should be considered.

Adding additional Client Connectors and Service Connectors

Client and Service Connectors may become an observable bottleneck. Client Connectors and Service connectors are independently hosted by Neuron, each with a limited set of resource and concurrency constraints. Adding additional ones may resolve the issue.

Changing default bindings

Neuron ships with many WCF bindings used for Client Connectors and Service Connectors. However, almost all the bindings use the WCF defaults for threshold and binding property settings. These may cause unnecessary constraints and actually, over time have a deleterious effect on performance. The best way to overcome this it to create custom bindings within the Neuron ESB Explorer that override these defaults and use those custom bindings in the Client Connectors and Service Connectors.

Adding additional Topics

Generally speaking each Neuron Topic is represented internally by a hosted runtime service called a Publishing Service. The exceptions to this are if Peer or BizTalk are used to configure the Network property of a Topic. However, if Tcp or Msmq are used to configure the Network property of the Topic, the Topic itself may become the bottleneck. In this case, adding another Topic can increase throughput on the system, as long as the existing server resources are not already constrained.

Direct Routing

Neuron ESB supports a variety of routing options using the pipeline processing designer. One option is called *Direct Routing. Direct Routing* uses a configured pipeline that directly routes the incoming message to the specific Service Connector, bypassing Neuron's pub/sub engine entirely. Overall performance gains of up to 50% can be obtained using this method of routing.

Adding additional Neuron Servers

If the maximum sustainable throughout is reached on an existing Neuron server, then one or more other Neuron Servers can be added to the existing Neuron configuration. A load balancer (hardware or software based) can be used to evenly distribute service request loads between the Neuron servers.

Neuron Configuration

For testing request/response services, the Neuron Topic was configured ALMOST identically to the configuration used for the Concurrent Neuron Parties testing, with the exception of the *Max Concurrent Calls* setting. This was set to 25.

Additionally, the *Max Concurrent Calls* setting on the *Client Connector* was also set to 25.

Test Description

The following tests were performed using SOAP UI to simulate load testing.

- Calling a service directly
- Calling a service through Neuron ESB's Client Connector (Neuron Topic Routing)
- Calling a service through Neuron ESB's Client Connector (Direct Routing)

The service endpoint tested was a simple WCF service, hosted by a console application and configured to use the basicHttp binding. It accepts a message and returns that same message as the response.

Directly against service

All tests were conducted against the actual service endpoint hosted in a console application with the following URL: http://localhost:8731/Design_Time_Addresses/ServiceTwo/Service1/.

Neuron Topic Routing

All tests were conducted against the service endpoint hosted by Neuron's *Client Connector*, represented by the following URL: http://localhost:9001/myService, hosted by the Neuron ESB runtime. When the service request is made, it is routed across the Tcp based Topic to the *Service Connector* which configured with the actual service endpoint address

(http://localhost:8731/Design_Time_Addresses/ServiceTwo/Service1/). The response is routed back to the load testing agent through the bus.

Neuron Direct Routing

Identical to the Neuron Topic Routing except that routing to the *Service Connector* over the Tcp based Topic was circumvented. Instead the call to the *Service Connector* was made within the pipeline associated with the *Client Connector* using Named Pipes.

Test Results

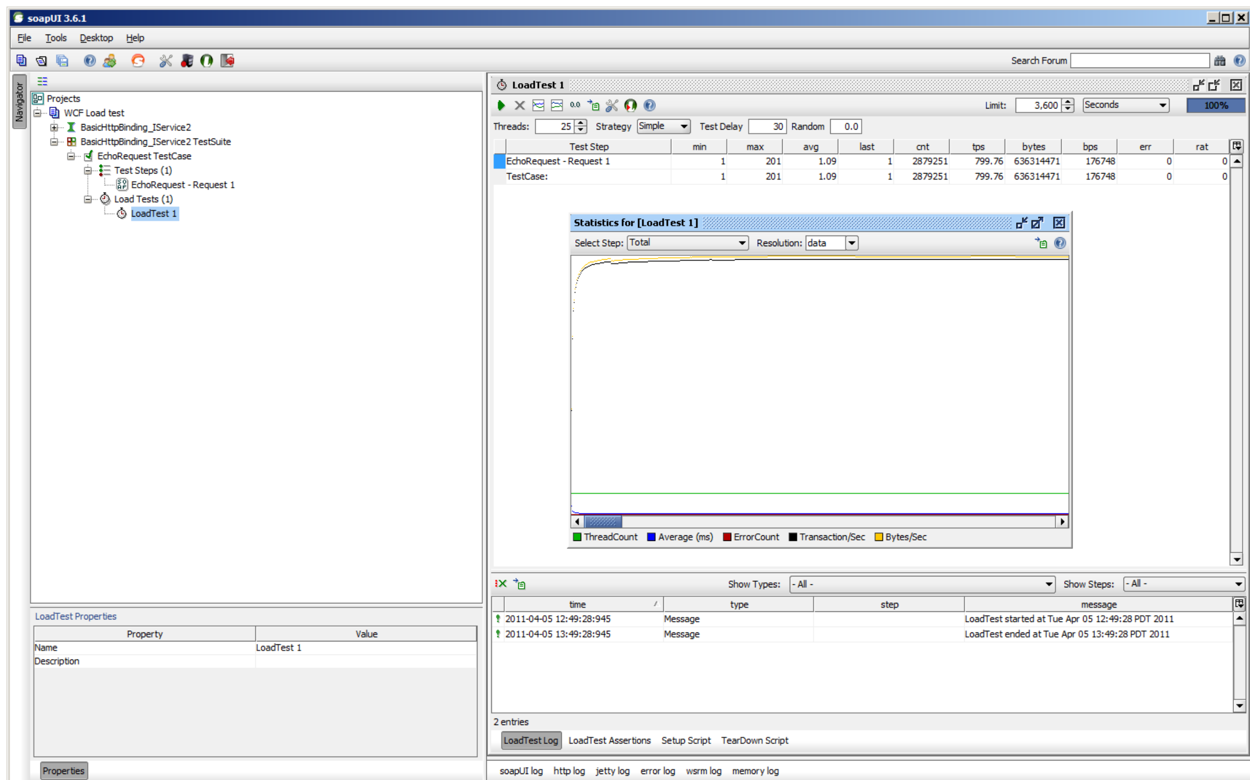
Test – 1 Hour – 25 agents – 30 millisecond delay

Each test lasted 1 hour and included 25 agents, simulating 25 concurrent users. The agents were gradually ramped up, 1 agent per second, until 25 agents were consecutively submitting requests. The interval between service request invocations for each agent was configured at 30 milliseconds. As a frame of reference, this would be consistent with 25 users concurrently submitting approximately 33 service requests/second.

Test goal: Determine sustainable throughput that achieved 25 concurrent users, while maintaining an Average RT of less than 5 milliseconds.

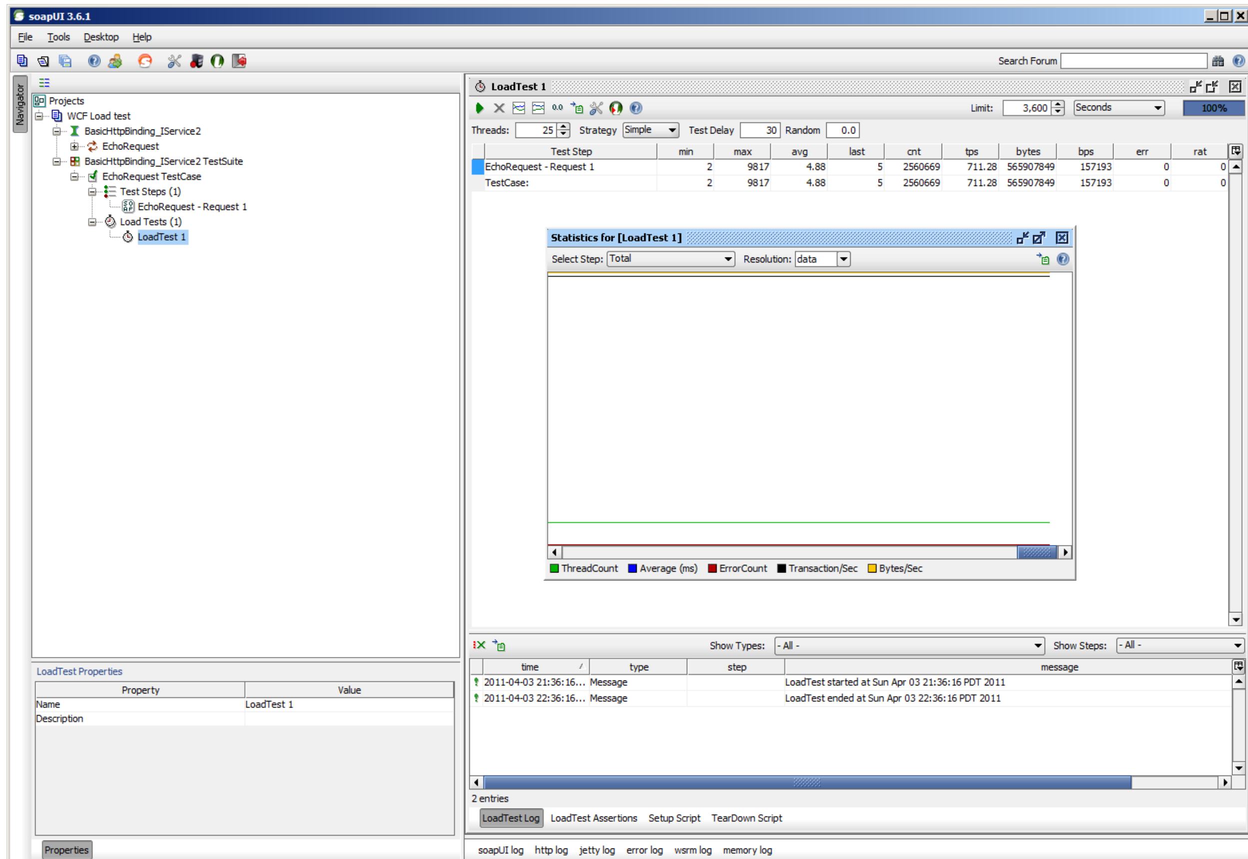
Directly against service

- Average TPS = 799
- Average RT = 1.09 milliseconds



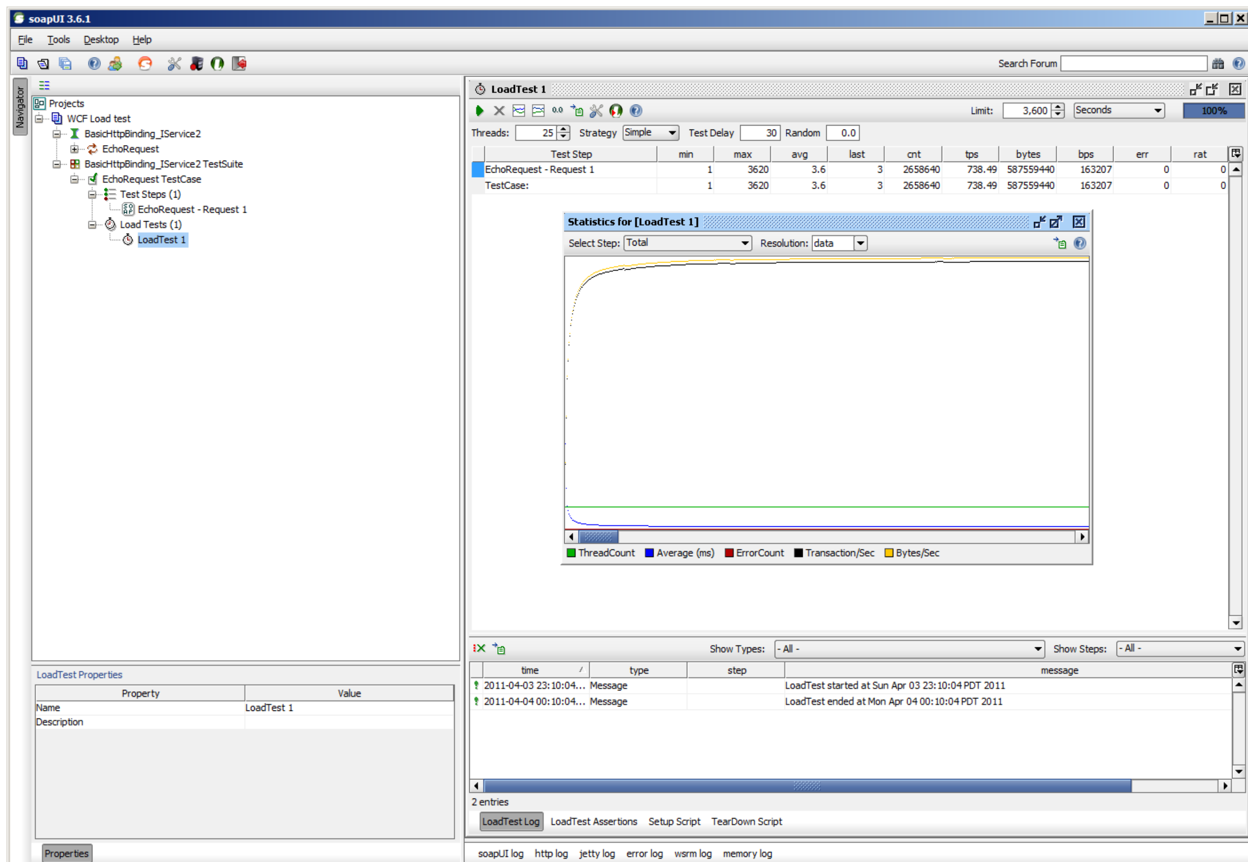
Neuron Topic Routing

- Average TPS = 711
- Average RT = 4.88 milliseconds



Neuron Direct Routing

- Average TPS = 738
- Average RT = 3.6 milliseconds



Observations

Every test reached the goal of 25 concurrent users while maintaining an average RT of 5 milliseconds or less. Additionally all TPS numbers were within the same 700 range. With every test, the goal was to also reach a point of straight horizontal recordings for all graphs, indicating sustainable throughput.

No errors occurred during the testing. In this test, the TPS for Neuron Direct Routing was approximately 3% greater than that of Neuron Topic routing. The average RT for Neuron Direct Routing was about 26% less than that of Neuron Topic Routing.

When comparing Neuron Direct Routing to calling the service directly (Neuron not used), TPS is approximately 8% greater with the latter. The average RT for calling the service directly was about 70% less than that of Neuron Direct Routing.

Processing utilization

As can be seen from the Task Manager images below, the average CPU% utilization was only about 28% when Neuron Topic Routing was used. Whereas, CPU% utilization was approximately half that when Neuron Direct Routing was employed.

Neuron Topic Routing

Windows Task Manager

File Options View Help

Applications | Processes | Services | Performance | Networking | Users

Image Name	User Name	CPU	CPU Time	Working Set ...	Peak Working S...	Memory (Priv...	Commit Size	Paged Pool	NP Pool	Page Faults	Base Pri	Threads	I/O
CCC.exe	marty.wasz...	00	0:00:16	13,304 K	58,872 K	8,788 K	62,168 K	415 K	68 K	347,940	Normal	17	
communicator.exe *32	marty.wasz...	00	0:31:14	88,352 K	97,136 K	44,220 K	68,812 K	661 K	96 K	8,320,138	Normal	53	1,1
conhost.exe	SYSTEM	00	0:00:00	280 K	3,060 K	204 K	1,464 K	46 K	4 K	794	Normal	1	
conhost.exe	LOCAL SER...	00	0:00:00	328 K	3,164 K	216 K	1,536 K	46 K	4 K	817	Normal	1	
conhost.exe	marty.wasz...	00	0:00:00	5,476 K	5,476 K	1,676 K	1,888 K	122 K	5 K	1,406	Normal	2	
csrss.exe	SYSTEM	00	0:00:10	3,244 K	6,224 K	1,872 K	4,192 K	167 K	13 K	10,267	Normal	10	
csrss.exe	SYSTEM	00	0:15:11	24,272 K	29,828 K	10,732 K	21,840 K	362 K	22 K	1,447,794	Normal	13	5,9
devenv.exe *32	marty.wasz...	00	0:41:52	276,920 K	319,280 K	191,020 K	247,484 K	930 K	167 K	2,664,195	Normal	48	1
dwm.exe	marty.wasz...	00	0:00:24	5,972 K	19,712 K	1,016 K	2,424 K	123 K	7 K	2,680,860	Normal	3	
ESBService.exe	SYSTEM	28	0:13:52	105,856 K	112,376 K	64,228 K	146,920 K	484 K	161 K	106,614	Normal	67	
explorer.exe	marty.wasz...	00	1:01:01	139,480 K	154,444 K	100,856 K	137,732 K	1,058 K	87 K	4,557,662	Normal	44	
fdhost.exe	LOCAL SER...	00	0:00:00	1,192 K	7,012 K	724 K	5,348 K	68 K	12 K	38,079	Normal	13	
fdlauncher.exe	LOCAL SER...	00	0:00:00	276 K	4,656 K	248 K	2,268 K	45 K	5 K	3,960	Normal	3	
iexplore.exe *32	marty.wasz...	00	0:05:57	77,424 K	216,824 K	55,148 K	129,808 K	394 K	92 K	4,242,278	Normal	61	1,5
iexplore.exe *32	marty.wasz...	00	2:46:12	416,656 K	556,128 K	379,324 K	482,448 K	440 K	163 K	188,093,149	Normal	161	73,5

Show processes from all users

End Process

Processes: 80 | CPU Usage: 34% | Physical Memory: 55%

Neuron Direct Routing

Windows Task Manager

File Options View Help

Applications | Processes | Services | Performance | Networking | Users

Image Name	User Name	CPU	CPU Time	Working Set ...	Peak Working S...	Memory (Priv...	Commit Size	Paged Pool	NP Pool	Page Faults	Base Pri	Threads	I/O
communicator.exe *32	marty.wasz...	00	0:31:22	89,496 K	97,136 K	45,192 K	70,076 K	652 K	98 K	8,331,549	Normal	54	1,1
conhost.exe	SYSTEM	00	0:00:00	280 K	3,060 K	204 K	1,464 K	46 K	4 K	794	Normal	1	
conhost.exe	LOCAL SER...	00	0:00:00	328 K	3,164 K	216 K	1,536 K	46 K	4 K	817	Normal	1	
conhost.exe	marty.wasz...	00	0:00:00	5,480 K	5,480 K	1,676 K	1,888 K	122 K	5 K	1,407	Normal	2	
csrss.exe	SYSTEM	00	0:00:10	3,244 K	6,224 K	1,872 K	4,192 K	167 K	13 K	10,280	Normal	10	
csrss.exe	SYSTEM	00	0:15:19	23,876 K	29,828 K	10,732 K	21,840 K	360 K	22 K	1,449,279	Normal	13	5,9
devenv.exe *32	marty.wasz...	00	0:42:37	284,028 K	319,280 K	198,192 K	255,644 K	933 K	171 K	2,724,818	Normal	51	1
dwm.exe	marty.wasz...	00	0:00:24	5,972 K	19,712 K	1,016 K	2,424 K	123 K	7 K	2,680,860	Normal	3	
ESBService.exe	SYSTEM	13	0:17:12	106,756 K	109,400 K	62,900 K	148,352 K	537 K	148 K	104,600	Normal	65	1,3
explorer.exe	marty.wasz...	00	1:01:27	138,704 K	154,444 K	100,104 K	137,212 K	1,065 K	88 K	4,578,792	Normal	50	
fdhost.exe	LOCAL SER...	00	0:00:00	1,192 K	7,012 K	724 K	5,348 K	68 K	12 K	38,420	Normal	13	
fdlauncher.exe	LOCAL SER...	00	0:00:00	276 K	4,656 K	248 K	2,268 K	45 K	5 K	3,960	Normal	3	
iexplore.exe *32	marty.wasz...	00	0:06:10	77,500 K	216,824 K	55,184 K	129,912 K	395 K	92 K	4,243,124	Normal	62	1,5
iexplore.exe *32	marty.wasz...	04	3:01:30	417,568 K	556,128 K	380,188 K	483,216 K	440 K	164 K	188,105,352	Normal	160	73,5
iexplore.exe *32	marty.wasz...	00	0:17:03	171,112 K	237,980 K	135,360 K	191,412 K	429 K	86 K	13,575,920	Normal	31	4,5

Show processes from all users

End Process

Processes: 79 | CPU Usage: 25% | Physical Memory: 48%

Test – 1 Hour – 10 agents – 1 millisecond delay

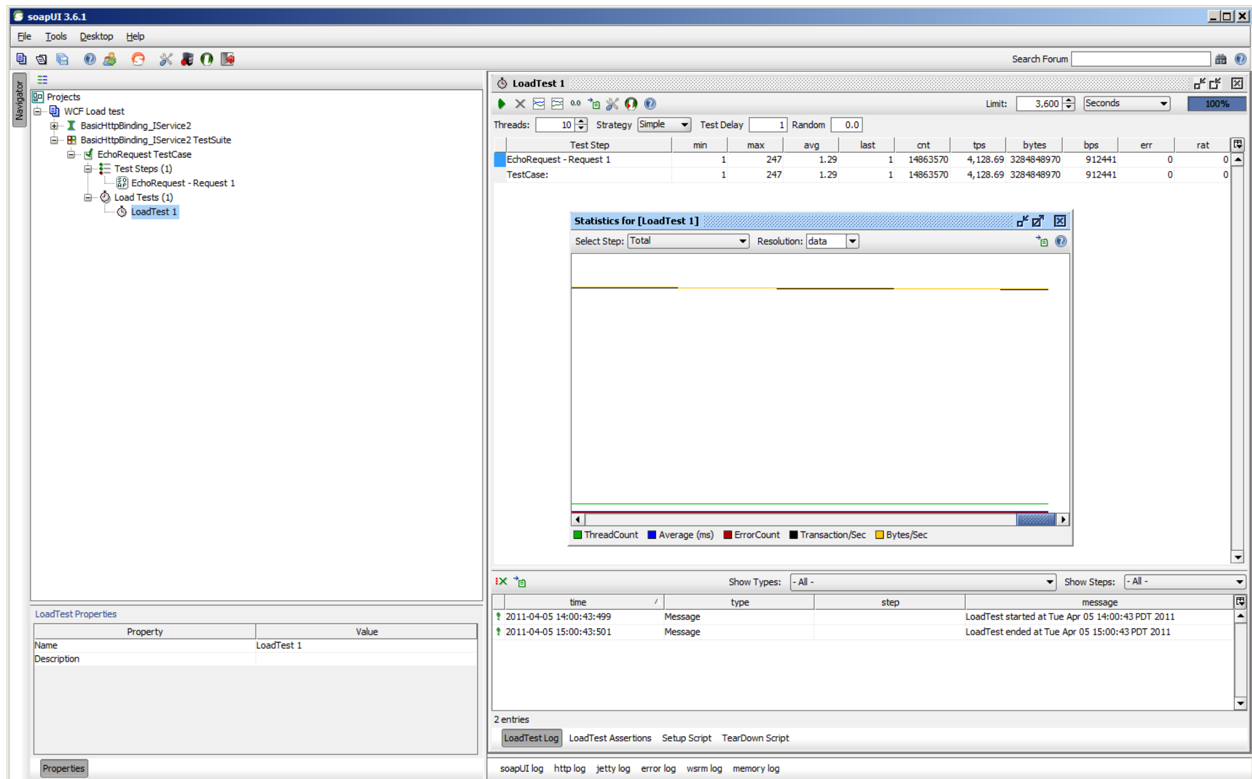
Each test lasted 1 hour and included 10 agents, simulating 10 concurrent users. The agents were gradually ramped up, 1 agent per second, until 10 agents were consecutively submitting requests. The interval between service request invocations for each agent was configured at 1 millisecond. As a frame of reference, this would be consistent with 10 users concurrently submitting approximately 1000 service requests/second.

This is an unlikely load anyone would encounter. The underlying purpose of the test was to attempt to increase the CPU utilization while maintaining a sustainable throughput.

Test goal: Determine sustainable throughput that achieved 10 concurrent users, while maintaining an Average RT of less than 10 milliseconds.

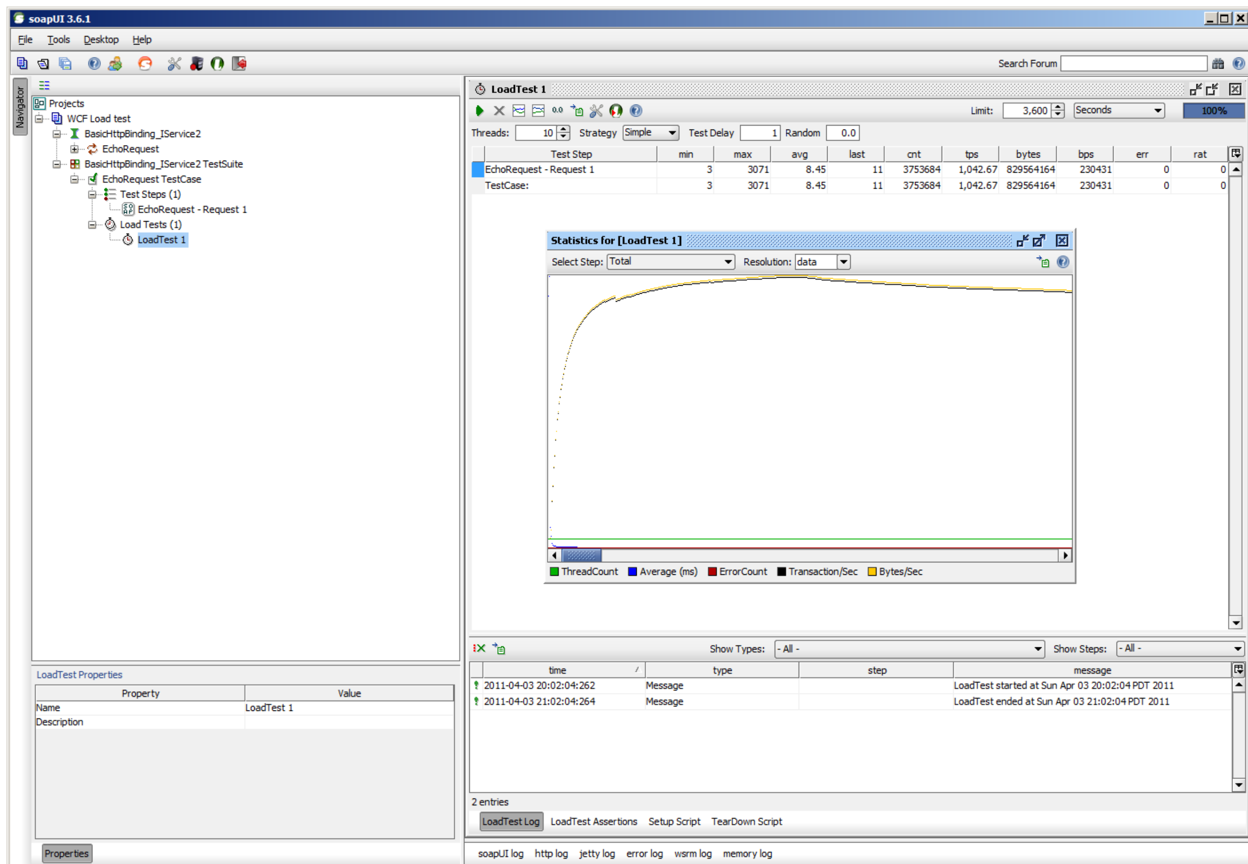
Directly against service

- Average TPS = 4,128
- Average RT = 1.29 milliseconds



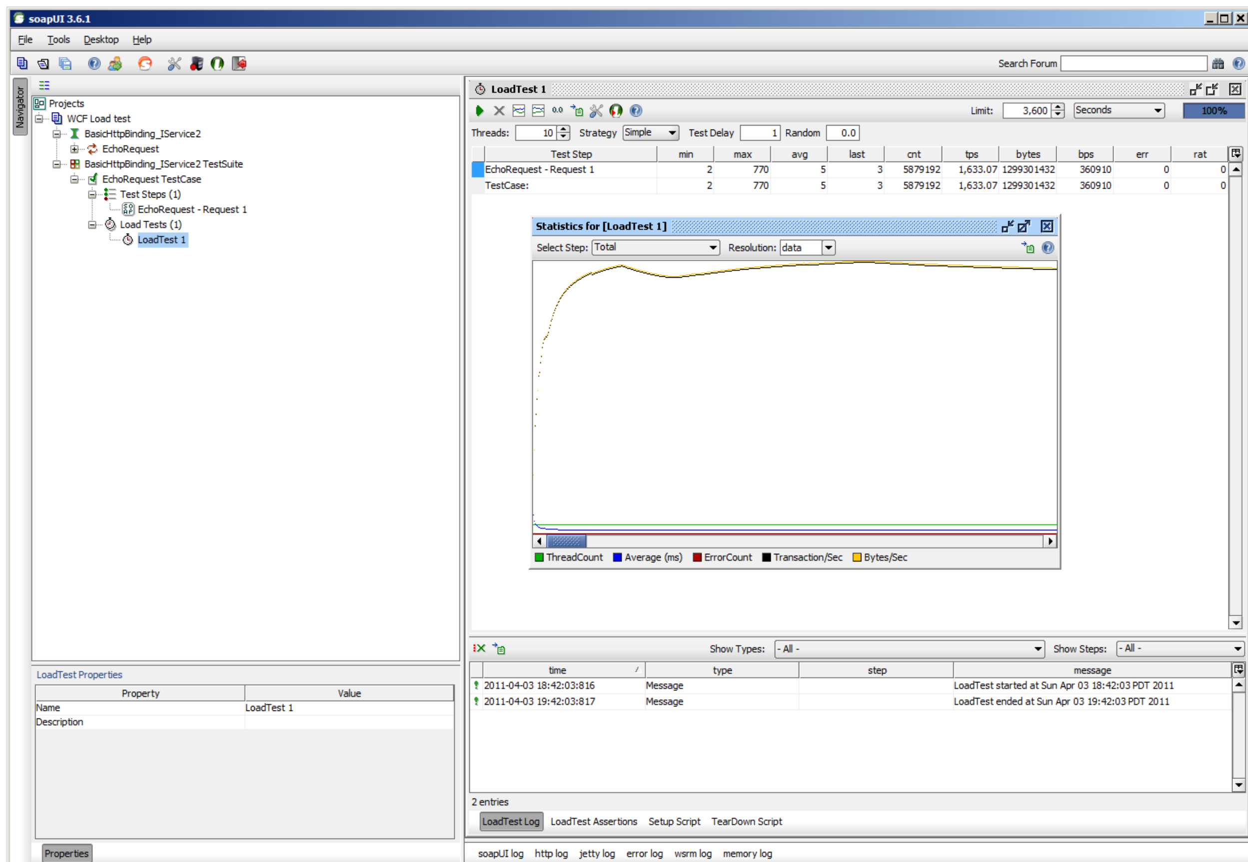
Neuron Topic Routing

- Average TPS = 1,042
- Average RT = 8.45 milliseconds



Neuron Direct Routing

- Average TPS = 1,633
- Average RT = 5 milliseconds



Observations

Every test reached the goal of 10 concurrent users while maintaining an average RT of 10 milliseconds or less. However, TPS numbers varied widely. With every test, the goal was to also reach a point of straight horizontal recordings for all graphs, indicating sustainable throughput.

No errors occurred during the testing. In this test, the TPS for Neuron Direct Routing was approximately 56% greater than that of Neuron Topic routing. The average RT for Neuron Direct Routing was about 41% less than that of Neuron Topic Routing.

When comparing Neuron Direct Routing to calling the service directly (Neuron not used), TPS is approximately 152% greater with the latter. The average RT for calling the service directly was about 74% less than that of Neuron Direct Routing.

NOTE: Calling the service directly behaved differently than when either Neuron Topic or Direct Routing was used. Specifically, within 10 minutes of starting each of the Neuron related tests, the TPS and average RT stabilized and remained consistent throughout the remainder of the hour, indicating that a sustainable throughput was reached. However, the test that involved calling the service directly achieved a TPS of approximately 4,755 within the first 10 minutes, but from there gradually declined during the remainder of the hour. Within 30 minutes this value decreased to 4,400. By the end of the test, the TPS value was still decreasing, terminating with a value of 4,100. This suggests that a sustained throughput was not achieved, and if left to run longer, TPS may decrease even more. An additional test was conducted in which calling the service directly was allowed to run for 2 hours. After about 1.5 hours the numbers stabilized. The final TPS number achieved was 3,548, approximately 117% greater than Neuron Direct Routing. The average RT was 1.6 milliseconds, or about 68% less than that of Neuron Direct Routing

In this test case, TPS was significantly less then when calling the service directly, yet average RT only varied by less than 3 to 4 milliseconds. Considering the CPU utilization of the Neuron ESB host, either tuning or adding an additional Topic and/or Service/Client Connector could be considered to increase the TPS while maintaining a sub 10 millisecond average RT.

Processing utilization

The average CPU% utilization was about 50% when Neuron Topic Routing was used. Whereas, CPU% utilization was approximately 35%-40% when Neuron Direct Routing was employed.

Tcp Optimization

Neuron ESB highly leverages Tcp for all internal services as well as Topics configured to use Tcp as the network transport. To ensure optimal performance when running under high load conditions, the Tcp stack should be tuned on all Neuron ESB servers as well as any machines remotely hosting a Neuron Party. Much of the tuning can be configured directly in the registry, while other tuning may be more specific to the machine and/or using NETSH commands. Below is a sample Registry file that can be merged on Window 7, Windows 2008 and later operating systems. Some may also be used for Windows XP, Vista and Windows 2003 (see Description section below to determine which ones to apply for the various operating systems).

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa]
"DisableLoopbackCheck"=dword:00000001

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Network\Connections\StatMon]
"ShowLanErrors"=dword:00000001

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management]
"DisablePagingExecutive"=dword:00000001
"SystemPages"=dword:ffffffff

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters]
"DefaultTTL"=dword:00000040
"TcpTimedWaitDelay"=dword:0000001e
"EnableWsd"=dword:00000000
"EnableTCPA"=dword:00000001
"TCPMaxDataRetransmissions"=dword:00000007
"DisableTaskOffload"=dword:00000000
"EnableDca"=dword:00000001
"SynAttackProtect"=dword:00000000

"ReservedPorts"=hex(7):31,00,34,00,33,00,33,00,2d,00,31,00,34,00,33,00,34,00,\
 00,00,35,00,30,00,30,00,30,00,30,00,2d,00,35,00,30,00,30,00,30,00,38,00,00,\
 00,35,00,30,00,32,00,30,00,30,00,2d,00,35,00,30,00,32,00,30,00,30,00,00,00,\
 35,00,30,00,30,00,31,00,30,00,2d,00,35,00,30,00,31,00,30,00,30,00,00,00,00,\
 00
```

Description

EnableWsd

Setting **EnableWsd = 0**, essentially disables Tcp Heuristics. When no faulty networking devices are involved, setting EnableWsd to 0 can enable more reliable high-throughput networking via Tcp receive window autotuning. However, if it is believed that either the NIC does not support Tcp scaling or is malfunctioning then set EnableWsd to 1 to enable heuristics. Windows then will automatically disable TCP receive window autotuning when heuristics suspect a network switch component may not support the required Tcp Receive Side Scaling. EnableWsd is used for Windows 2003 and Vista.

There is an associated NETSH command that can be used on Windows 7, Windows 2008 and later operating systems to disable Tcp Heuristics immediately.

DefaultTTL

This should be changed from its **default of 128 to 30**. This should be changed on the Neuron sever. It may also be changed on remote clients that run host the Neuron Party. However, testing should be performed to ensure it does not adversely affect any other applications that may exist on the machine.

TcpTimedWaitDelay

Short lived (ephemeral) Tcp/IP ports above 1024 are allocated as needed by the OS. However, in some instances under heavy load it may be necessary to lower the delay value to increase the availability of user ports requested by an application.

If the default limits are exceeded under heavy loads, the following error may be observed: "*address in use: connect exception*".

Neuron's Tcp Publishing service is based on underlying socket connections/ports. This setting should be changed from its **default of 120 seconds to 30 seconds** on the server and client machines to increase availability of ports.

EnableTCPA

Setting **EnableTCPA = 1** enables support for advanced direct memory access (NetDMA). In essence, it provides the ability to more efficiently move network data by minimizing CPU usage. NetDMA frees the CPU from handling memory data transfers between network card data buffers and application buffers by using a DMA engine. EnableTCPA is used for Windows 2003 and Vista.

There is an associated NETSH command that can be used on Windows 7, Windows 2008 and later operating systems to enable NetDMA immediately.

TCPMaxDataRetransmissions

This value controls the number of times that TCP retransmits an individual data segment (not connection request segments) before aborting the connection. The retransmission time-out is doubled with each successive retransmission on a connection and is reset when responses resume. Although the documentation states the default value is 5, when not present in the registry, the default behavior is 255 retransmissions. **This should be changed to 7**.

DisableTaskOffload

Setting **DisableTaskOffload = 0** allows for reducing CPU load by offloading some tasks required to maintain the TCP/IP stack to the network card. Theoretically, Windows should automatically detect capable network hardware.

The tasks offloaded are as follows:

- TCP/IP checksum calculation - each packet sent includes a verification checksum.
- TCP/IP segmentation - also known as "TCP Large Send" where Windows sends a large amount of data to the network card, and the NIC is then responsible for dividing it according to the network MTU. Experimental feature, not enabled by default.
- IPSec Encryption cyphers and message digests - provides encryption of packets at the hardware level.

EnableDca

Setting **EnableDca = 1** adds NETDMA 2.0 Direct cache access support. Direct Cache Access (DCA) allows a capable I/O device, such as a network controller, to deliver data directly into a CPU cache. The objective of DCA is to reduce memory latency and the memory bandwidth requirement in high bandwidth (Gigabit) environments. DCA requires support from the I/O device, system chipset, and CPUs.

Windows 7, Windows 2008 and later operating systems support this. Vista and Windows 2003 do not.

There is an associated NETSH command that can be used on Windows 7, Windows 2008 and later operating systems to enable DCA immediately.

SynAttackProtect

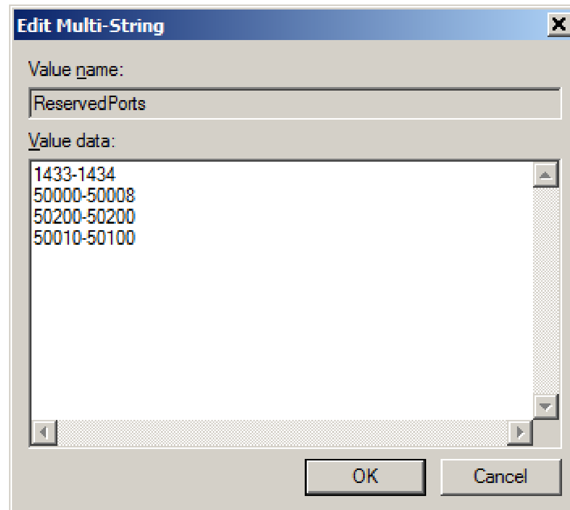
Should be disabled on the Neuron sever and remote clients that submit heavy volumes by setting **SynAttackProtect = 0**. This is used for Windows XP, Vista and Windows 7. It is no longer used in Windows 2008 and later operating systems. When enabled, the follow error may be observed: "*An existing connection was forcibly closed by the remote host*"

DisableLoopbackCheck

Should be disabled on the Neuron Server

ReservedPorts

The next to the last entry, is the list of reserved Neuron ports to ensure that they do not interfere with the Dynamic port range. This entry is only necessary on Vista, Windows 7 and Windows 2008 machines as the dynamic port range has changed with those Operating Systems. On the Neuron server, this should be set to the following:



NOTE: On the machines where the Neuron Party is hosted (which would also be the Neuron Server) the reserved port range that represents the client port range of the referenced Tcp based Topic should also be added. By default the port range for one topic is 61007 – 62007. Port ranges will be unique for each Topic.

NETSH Commands

In Windows 7, Windows 2008 and later operating systems the NETSH command can be used to modify Tcp/IP settings for the machine. These modifications will take place immediately. To check the current status of the Tcp/IP modifiable parameters, in elevated command prompt type the following command:

netsh int tcp show global

You will be presented with something like the following:

```
Administrator: C:\Windows\system32\cmd.exe

C:\>netsh int tcp show global
Querying active state...

TCP Global Parameters
----------------------------------------------
Receive-Side Scaling State          : enabled
Chimney Offload State               : enabled
NetDMA State                        : enabled
Direct Cache Acess (DCA)            : enabled
Receive Window Auto-Tuning Level    : normal
Add-On Congestion Control Provider  : none
ECN Capability                      : disabled
RFC 1323 Timestamps                 : disabled


C:\>_
```

The following NETSH commands may be executed on the machines to modify existing settings, depending on the level of support that exists within the current routers and NICs:

netsh int tcp set global congestionprovider=ctcp

NOTE: Should already be set to ctcp by default

netsh int tcp set heuristics disabled

NOTE: This is automatically disabled if EnableWsd is set to 0 in the Registry

netsh int tcp set global autotuninglevel=normal

netsh int tcp set global netdma=enabled

NOTE: This is automatically enabled if EnableTCPA is set to 1 in the Registry

netsh int tcp set global dca=enabled

NOTE: Need to ensure NIC supports this. This is automatically enabled if EnableDca is set to 1 in the Registry

netsh int tcp set global chimney= enabled

NOTE: Need to ensure NIC supports this. This is Tcp offloading (TOE). TCP Chimney offload enables TCP/IP processing to be offloaded to network adapters that can handle the TCP/IP processing in hardware.

netsh int tcp set global rss=enabled

NOTE: The Receive Side Scaling should be enabled for dual core CPU's for parallel processing. This also requires support at the NIC level. This is enabled out of the box for Windows Server 2008.

netsh int tcp set global ecncapability=enabled

NOTE: ECN is only effective in combination with AQM (Active Queue Management) router policy.

Other NIC Settings

Enabling bi-directional flow-control and setting fixed speed and duplex is also strongly recommended. Flow-control avoids TCP packet floods during periods of network congestion/contention. Fixed speed and duplex ensures the NIC ports won't go into auto-negotiate mode (where the network "disappears" for a brief period then mysteriously returns).

Lastly, it is very important to maximize the ability of the on-board NIC hardware to process messages. This requires increasing the driver buffering parameters to their maximum values. Buffering parameters include, depending on the hardware and driver versions, Receive Buffers/Descriptors, Transmit Buffers/Descriptors, and Coalesce Buffers/Descriptors. The defaults are set to sacrifice performance to save memory (especially non-paged pool). With the amount of memory available on modern H/W and especially on 64-bit architectures we always want to trade space for time.

Conclusion

The Neuron 2.5.10 release offers a number of advanced tuning parameters and enhancements that allow it to outperform and scale to greater concurrency levels than what previous releases of Neuron

supported. These enhancements allow administrators to configure Neuron to control resource utilization and the associated behavior of Neuron Parties connecting to the bus. These new enhancements also improve the overall performance of adapters and service endpoints hosted by Neuron that use Tcp based Topics.

As in all cases, when tuning a Neuron implementation diligent testing should always be performed in a controlled environment to determine the best balance between performance, concurrency, application requirements, and resource utilization to determine the maximum sustainable throughput.