



Load Testing Tool

The load testing tool is an efficient way of generating simulated user connections to a Wowza Media Server. It can be used to generate load to static .flv streams or live streams (one-to-many live streaming).

On the server side the tool is used to replicate a single .flv file across a disk array to simulate multiple .flv file playback. This feature of the tool is not needed if testing live streaming.

On the client side the tool is used to simulate user connections. You can use multiple machines to generate the load. The connections can be configured to spread the load across multiple flv files in a uniform or random fashion to simulate user connecting to different flv files. You can use multiple machines to generate load. On the client side the tool will generate a new connection every 750 milliseconds until it reaches the configured number of connections.

Server install and configuration (follow these steps to test static .flv streaming)

1. Install Wowza Media Server
2. Copy contents (make sure you copy the individual files and not just the three folders) of loadtool/bin, loadtool/conf, loadtool/lib folders into the corresponding Wowza Media Server installation folders.
3. Create a folder on your server that is going to hold the test files (from now on we will call this the “test folder”).
4. Copy a single test .flv file into this folder (from now on we will call this the “test file”).
5. Edit conf/Application.xml and change Streams/StorageDir to point to the test folder.
6. Edit conf/Tests.xml, locate the test configuration for “PerformanceMediaDuplicateTest” and change the value of the StorageDir property to the test folder. Next, change the streamName property to the base file name (excluding the .flv extension) of your test file. Next, change the fileCount property to the number of .flv files you want to simulate in your testing. So if you want to see how the server handles serving up 400 different flv files at the same time change this number to 400 (make sure you have enough disk space on your server to hold the duplicated test files).
7. Next, we are going to run a quick tool on the server to duplicate the test file in the test folder. Open a command shell and change directory to the bin directory of the Wowza Media Server installation. Run the following command:

`./performance.sh PerformanceMediaDuplicateTest` (Linux or Mac OSX)
`performance PerformanceMediaDuplicateTest` (Windows)

You should see log output that shows the test file being duplicated “fileCount” times in the test folder. These are the test files that are going to be used for testing. If you want to change the test file at any time you can go back and edit `Tests.xml` and rerun this tool.

8. If you want to run the server on multiple ports you can edit `VHost.xml` and add new `HostPort` entries to the `HostPortList`.
9. The server is now ready to go. To startup the server execute the `./startup.sh` script from a shell prompt.

Client install and configuration

1. Install the Wowza Media Server
2. Copy contents (make sure you copy the individual files and not just the three folders) of `loadtool/bin`, `loadtool/conf`, `loadtool/lib` folders into the corresponding Wowza Media Server installation folders.
3. Edit `conf/Tests.xml`, locate the test configuration for “`PerformanceRemoteClientTest`” and change the following properties:
 - a. `workerCount`: the number of client connections that this client is going to simulate
 - b. `fileCount`: the number of files that you want those client connections to use. If you are testing streaming to a live stream then set the `fileCount` to 0. This will instruct the test tool to NOT append a file number to the `streamName`.
 - c. `streamName`: for static `.flv` testing its the base test file name without the `.flv` extension. For live streaming, it’s the name of the published live stream.
 - d. `connectionString`: the name of the `[application]/[application-instance]` to connect to. On each client machine you must create a folder in the “`application`” folder for the `[application]` that you define here. So if your `connectionString` is “`fastplay/fastplay`” then you must create a folder name “`fastplay`” in your “`applications`” folder.
 - e. `doRandom`: true if you want each client connection to randomly choose a file to stream based on `fileCount` or false if you want an even distribution of connections across `fileCount`. Ignored if `fileCount` is 0.
 - f. `IpAddress`: the `IpAddress` of the server.
 - g. `Ports`: a list of ports separated by the pipe “`|`” character that you want to the client connections to use. The system will even distribute the connections across the ports that you specify.

Static .flv simulation: So what does this all mean? Basically,

workerCount is the number of connections to simulate. The fileCount and doRandom properties lets you simulate different load situations. If you want to test pure FileIO you might set fileCount equal to workerCount and doRandom to false. This will cause a worse case scenario on the FileIO subsystem since the most files will be active at the same time. If you want to simulate real world load you might set fileCount to a reasonable number that represent how many files you truly expect to serve and doRandom to true which will spread the load more randomly across the file set. If you want to see how much of the performance bottlenecks are FileIO related you might set fileCount to 1 and take the FileIO out of the equation.

Live streaming: So what does this all mean? Basically, workerCount is the number of connections to simulate. For live streaming that is it. It is very simple.

4. To run a test, first start the server. If you are testing live streaming, you need to use Flash to stream a live stream to the server. You can use the VideoChat example to stream media content from a web camera to the server. Next, on each of the client machines open a shell prompt and execute the command:

```
./performance.sh PerformanceRemoteClientTest (Linux and Mac OSX)  
performance PerformanceRemoteClientTest (Windows)
```

You should see a new connection created by the client every $\frac{3}{4}$ a second or so. You should see the server accepting those connections. The performance system monitors each of the simulated client connections to see if the server falls behind in delivering video and audio data to the client. You will see log messages in the client command window (and in the log files) when a connections gets starved for video/audio data. The log entries look something like:

```
Status running:100 currBehind:0
```

The “running” is the total number of streams that are running and “currBehind” is the total number of streams that are late in getting video/audio data from the server.

NOTE: For static .flv streaming a lot of the Wowza Media Server performance is driven by the underlying systems IO performance. We can only go as fast as the server can deliver flv bits from the disk to the server. To get the most throughput out of a server we suggest you use RAID 0 or RAID 10 configurations with as many disks as possible in the RAID array. Also if you are testing on Linux please take a quick look at the README at the "Performance Tuning - Linux" section.

NOTE: Do not try to generate too many simulated connections from a single client machine. I have found that about 200-400 is the maximum number of simulated a connections a single client machine can support (one of my test client machines is a

Windows XP box, AMD X2 3800+, 2GB RAM). Carefully watch the RAM and CPU usage on the client machines to determine if negative test results are due to client machine rather than server issues.