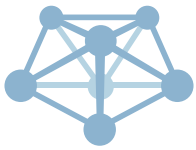


DDVTECH MEDIA SERVER PERFORMANCE MEASUREMENT INITIATIVE UPDATE, 2014 Q2

JARON VIËTOR, CTO, MISTSERVER / DDVTECH B.V.

Abstract. As a follow-up to our first media server performance measurement initiative release, we have further improved the testing methodology and done tests on more media server software products.

Our initial tests compared performance among three media servers: MistServer 2.0, Wowza Streaming Engine 4.0.0, and Nginx+RTMP module. This second release updates both MistServer to version 2.1 and Wowza Streaming Engine to version 4.0.3. We've also added Flussonic 4.2.5 and Adobe Media Server 5.0.3 to further widen the comparison.

There's a new test as well, dubbed the "burst test", which measures behaviour in sudden high load situations.

The updates and new results are encouraging and indicate we're on the right path.

1. Research question

In the previous performance measurement initiative release, we focussed on the HTTP server vs media server divide, and wanted to see if MistServer could bridge that gap.

Now we are updating those results with the latest findings, but have an additional question. When load balancing multiple servers, often a server will suddenly be under a lot of load (for example right after starting a new node to handle a surge of incoming viewers). This first load is often the highest load the server will be under, but only last a few seconds. We'll call this kind of load a "burst".

Our question this time is: how well can media servers handle these bursts?

This is useful information for determining server suitabilty in an automated load balancing system, as the ability to withstand burst behaviour is critical when switching many users to a new or different server. If a server fails to handle the initial burst, it will crash or slow down, often resulting in a domino effect of more and more powerful bursts throughout the network.

2. Methodology

The methodology has evolved slightly since the Q1 performance tests. Back in Q1, the metrics had quite a bit of wiggle room between pass and fail, because the measured servers had trouble living up to a more strict standard. Since then the average performance of media servers in general has improved greatly, and to account for this fact the new method is not as forgiving and demands good scores. For clarity, the entire methodology as it is now, is described in this chapter.

We created a simple tool that runs as a low-footprint system daemon, collecting generic system performance metrics. These metrics are for the whole system — not

Date: 2014 Q2 - revision 1.

just the media server. We opted for this method because there is no reliable way to collect these metrics for a single server.

The daemon attempts to poll the system usage once per second, but if the system is really busy the daemon may not be able to poll at this interval and the interval may become larger. Each interval, the following metrics are collected:

- Time since measurement start, in whole integer seconds (so larger intervals can be detected and accounted for)
- A one-minute rolling system load average
- Total memory use in bytes, excluding operating system buffers and caches
- The percentage of time since the last interval that the CPU spend in non-idle states
- Total number of bytes uploaded through all network interfaces since measurement start
- Total number of bytes downloaded through all network interfaces since measurement start

Additionally, a client process is defined, for the to-be-measured protocol and stream. This client process is intended to be run from one or multiple separate hardware machines that should have enough bandwidth and processing power available to retrieve the stream as many times as will be tested, simultaneously. The client process simulates a real client connecting to the specified stream over the specified protocol. It analyses the data coming through and performs corruption detection checks. If the data stream at any point becomes corrupted, the client process disconnects and exits without logging anything. Once the stream ends or the client process is killed, it logs the timestamp of the last media packet received, system time when the connection started, system time when the connection ended and the total connected time (calculated difference between start and end system time). This means the connection is either logged (clean state), or nothing is logged at all (error state).

Each test follows the following steps, in order:

(1) Enable auto-boot for the to-be-measured media server software
(2) Reboot the to-be-measured hardware
(3) Remotely enable the metric collection daemon
(4) Wait 2 seconds
(5) Start X client processes on one or more systems that are not measured
(6) Wait for Z seconds
(7) Kill all client processes
(8) Wait 10 seconds
(9) Remotely stop the metric collection daemon and collect the stored measurements.

3. Test hardware

We ran our initial tests on a single hardware configuration. It's specifications are:

- AMD FX(tm)-8120 Eight-Core Processor
- 2X8GB RAM: Kingston KHX1600C10D3/8GX
- ASRock N68-VS3 FX motherboard
- Eminent EM4028 gigabit ethernet PCI-E adapter
- Western Digital Blue WD5000LPVX, 500GB HDD

The installed operating system is Arch Linux, updated to May 2014. All optional system services and daemons have been disabled, except for sshd so the system can be remotely administered.

4. The completed measurements

Following the above methodology, we set up a 0.6Mbit RTMP stream, encoded from the Blender project's Big Buck Bunny movie. For reference, the exact sample clip used can be downloaded at the following URL: `http://dtsc.mistserver.com/example1_low.mp4` On each of the configurations described in the introduction, the following tests were run:

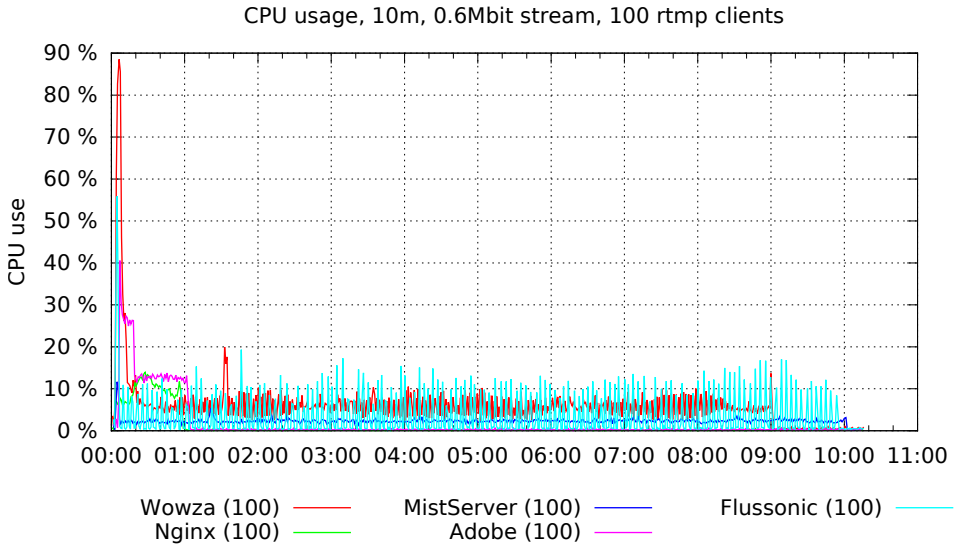
- Run 000: 1 client for 60 seconds (X=1, Z=60)
- Run 001: 1 client for 180 seconds (X=1, Z=180)
- Run 002: 1 client for 600 seconds (X=1, Z=600)
- Run 003: 100 clients for 60 seconds (X=100, Z=60)
- Run 004: 100 clients for 180 seconds (X=100, Z=180)
- Run 005: 100 clients for 600 seconds (X=100, Z=600)
- Run 006: 1000 clients for 60 seconds (X=1000, Z=60)
- Run 007: 1000 clients for 180 seconds (X=1000, Z=180)
- Run 008: 1000 clients for 600 seconds (X=1000, Z=600)
- Run 009: 1200 clients for 60 seconds (X=1200, Z=60)
- Run 010: 1200 clients for 180 seconds (X=1200, Z=180)
- Run 011: 1200 clients for 600 seconds (X=1200, Z=600)
- Burst run: 1000 clients for 5 seconds (X=1000, Z=5)

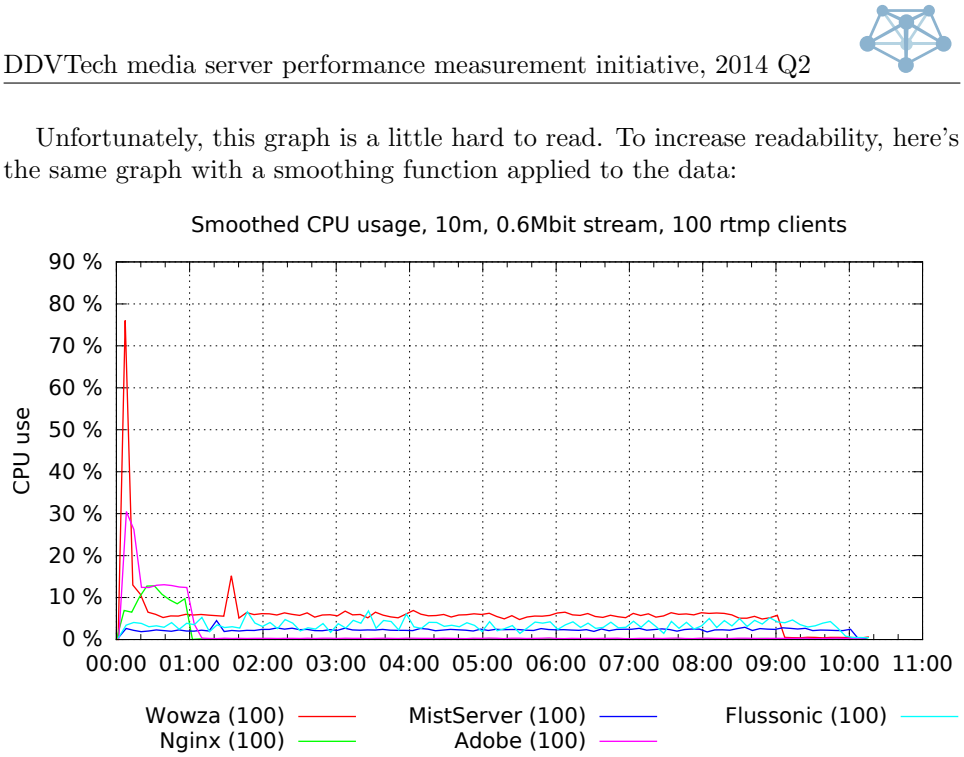
The raw collected data can be retrieved at this URL: `http://releases.ddvtech.com/raw_data_2014Q2.zip`

This archive contains a folder for each media server software, containing all the test runs in the format "run000_100_rtmp_60.csv". The first part is the number of the run, 000 through 011 as listed above. The second part is the total number of clients. The third part is the protocol tested and the last part is the test duration in seconds. Another file with the extension .times is available, containing the measurements taken by the client side, as above.

5. Compared to the previous results

We created graphs out of the raw data collected above, to simplify analysing this huge amount of data. Let's go over these and note the more interesting points. All graphs are included in both PDF and PNG format with the raw data download mentioned in the previous section. We'll start with the 10 minute 100-client test:





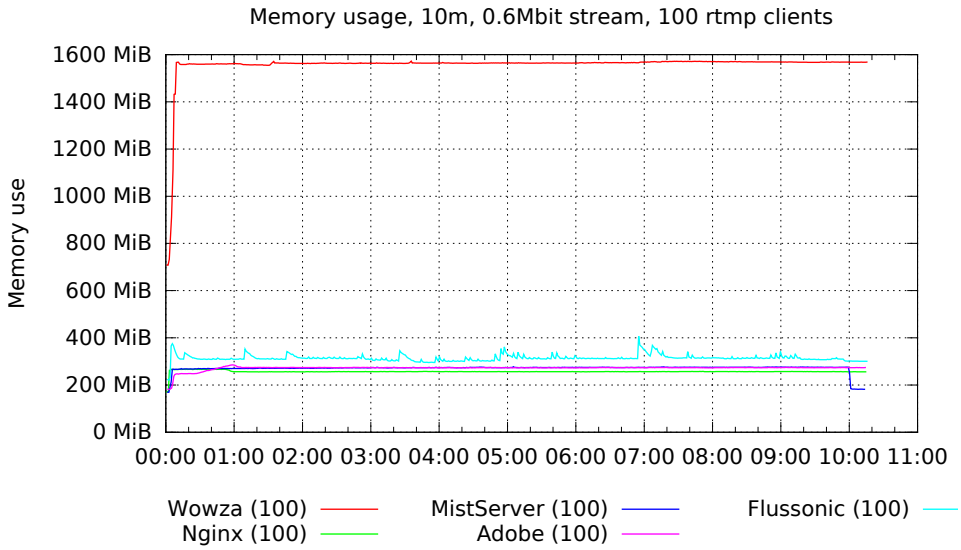
Unfortunately, this graph is a little hard to read. To increase readability, here's the same graph with a smoothing function applied to the data:

This is the CPU usage for a ten minute test of 100 clients, for all tested applications. **Less is better. The number between parenthesis is the count of successful clients.**

Compared to the Q1 performance tests, quite a bit has changed here: Wowza's CPU usage has been significantly decreased by a factor of almost 8, though they remain the least efficient in terms of CPU usage. MistServer's average and startup CPU usage have also gone down, though it's hard to tell since at 100 clients the numbers are so small. Nginx has obviously remained the same, as the software version hasn't been updated since the last tests.

The performance of the newly added products from Adobe and Flussonic isn't very surprising. Flussonic, being erlang-based, was expected to be among the best, and performs roughly as well as MistServer does. Their non-smoothed graph is much more spiky, but this isn't neccesarily a bad thing. Adobe's performance is similar to Nginx, though it has more trouble handling the sudden surge of connections. It quickly stabilizes to roughly the same level Nginx does.

Lets take a look at the memory use for this same test run:

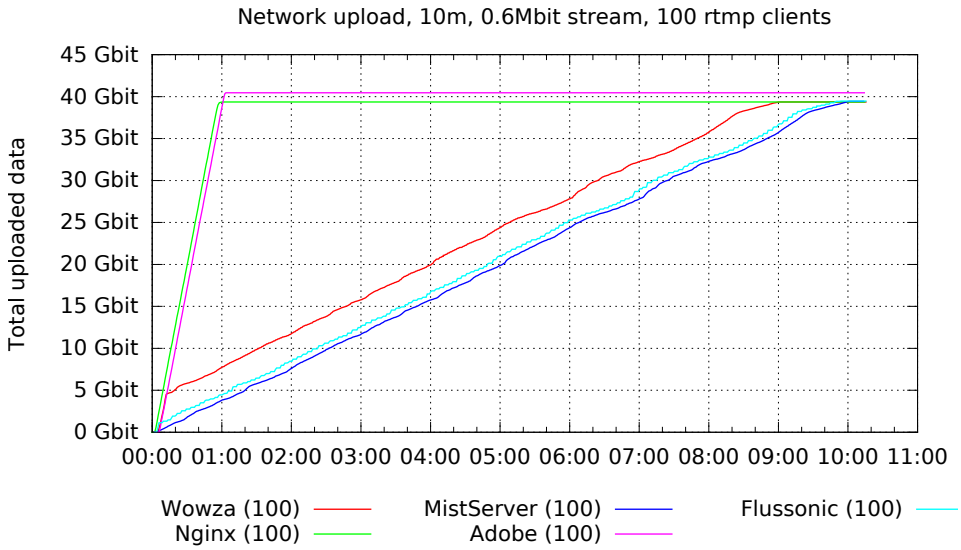


Again, **less is better, and the number between parenthesis is the count of successful clients.** In this case, Wowza has not improved but actually is using roughly 200 MiB more memory than in the Q1 results. The others haven't changed much, staying at approximately 300 MiB for 100 clients.

Adobe and Flussonic again show results in line with the rest. Flussonic uses slightly more memory, but also micromanages its memory usage more, which is visible from the small spikes in the graph.

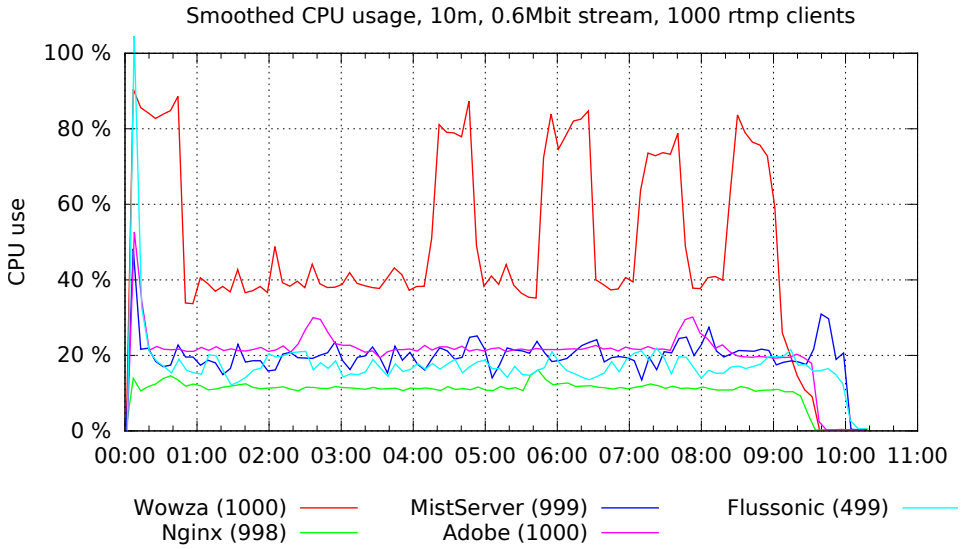
Just like in the Q1 tests, MistServer is the only software that frees all memory used when the 100 clients disconnect.

Lets take a look at the bandwidth usage:



Here too, **less is better.** We also recorded downloaded data, but since uploaded data accounts for almost all of the data usage in all cases, we only graphed the uploaded data here.

In the Q1 performance results, Wowza and Nginx didn't use bandwidth throttling while MistServer did. Now, Wowza has copied this technique and throttles the



bandwidth after an initial non-throttled period. Flussonic uses similar throttling to MistServer, while Adobe does not throttle at all just like Nginx.

Also noteworthy is that Wowza has fixed their Q1 bandwidth leak, and no longer sends more data than neccesary.

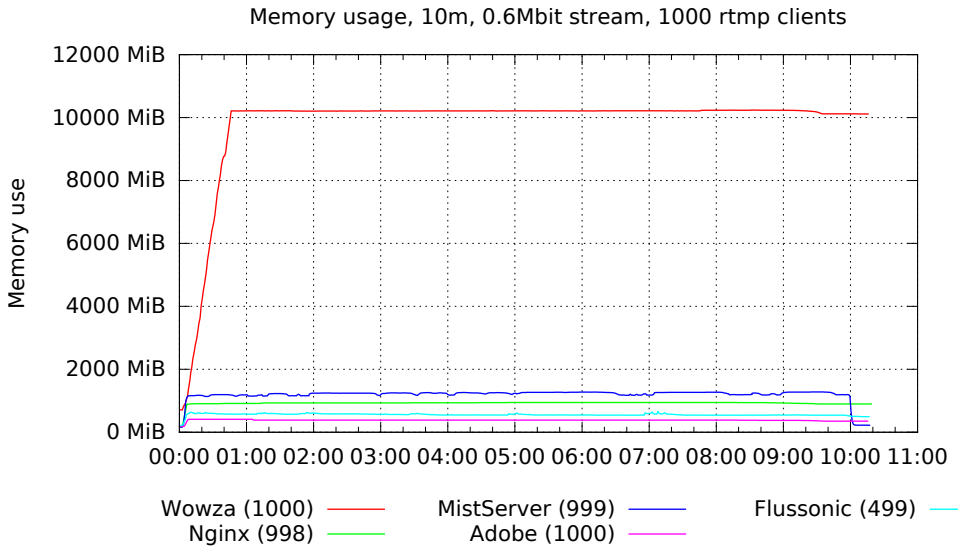
Time to crank it up a notch and show the graphs for 1000 clients - just a bit under the hardware limitations of the gigabit ethernet connection. We'll start with the smoothed CPU usage again:

Again major improvements for Wowza here - before 800 connections couldn't be handled at already, while now it handles all 1000 connections without a hitch. There's still high CPU usage spikes through, and the average use is roughly double what the others use.

Nginx remains the best performer at this load, but MistServer, Adobe and Flussonic are just a few percent away.

Unfortunately Flussonic is the only one not able to handle 1000 incoming RTMP connections - more on this in a later graph that focusses on this particular point.

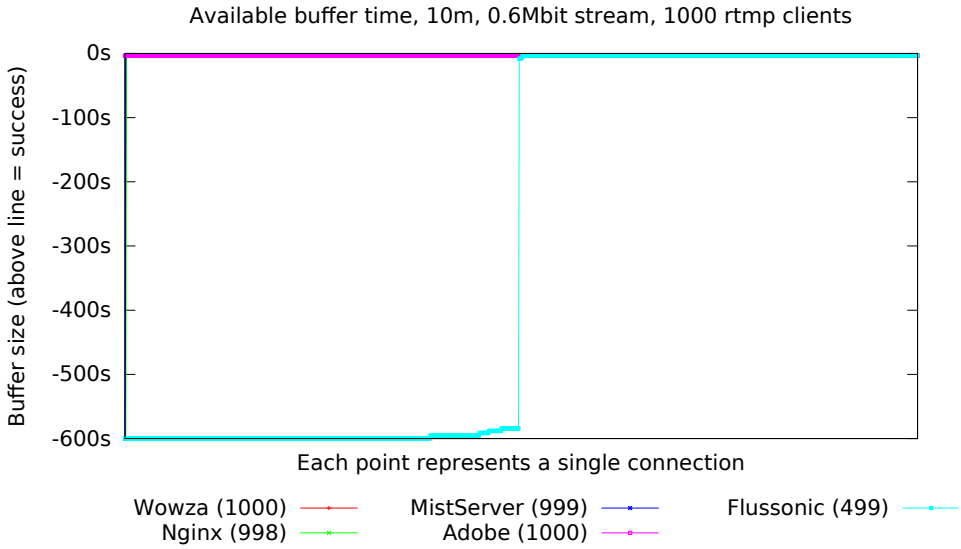
Let's take a look at the memory usage, first:



No surprises here, keeping the Q1 results in mind. Wowza still uses roughly 10GiB of memory for 1000 clients, while MistServer and Nginx use roughly 1GiB. Adobe and Flussonic manage even less, with Adobe taking the memory-efficiency crown.

We'll skip the bandwidth usage, since it's entirely non-surprising. Feel free to peruse the raw data download mentioned in the previous section, where all graphs are present.

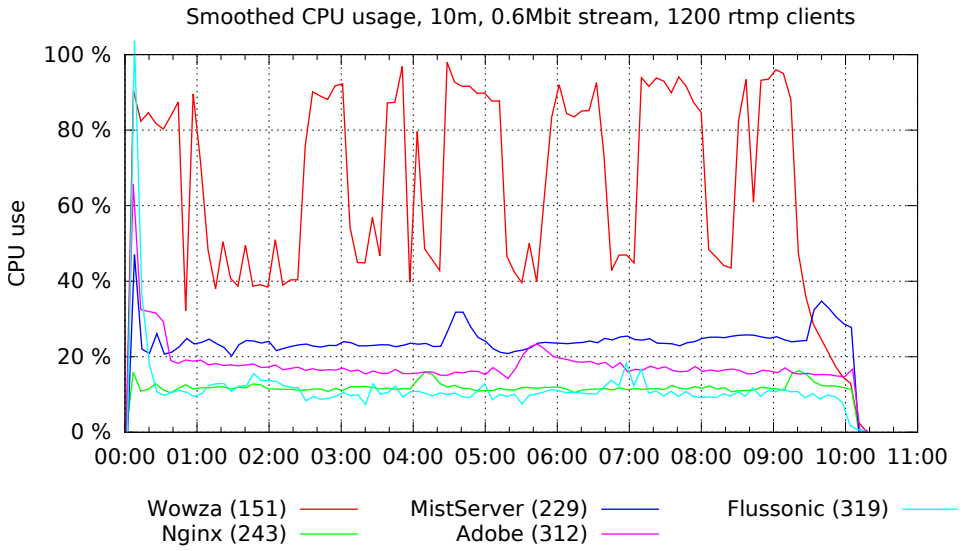
This brings us to the new buffer time graph. The new testing setup allows us to measure the amount of buffer time available on the client side, and the buffers look like this:

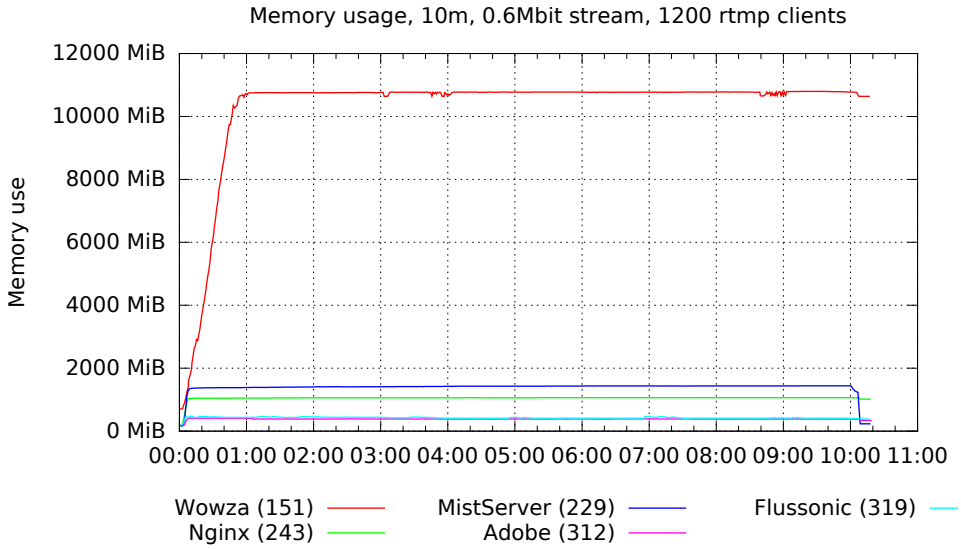


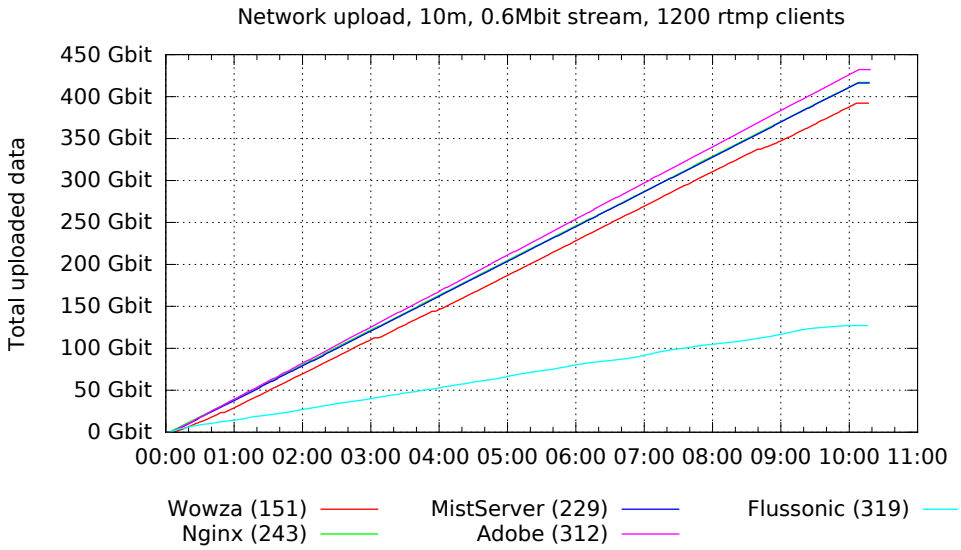
All measured software has all connections at 0 seconds buffer time after 10 minutes, which is exactly what you'd expect for a 10-minute video.

Except Flussonic, which is missing approximately 10 minutes of data for half the connections. It seems to have dropped them right at the start, as is clear in the graph.

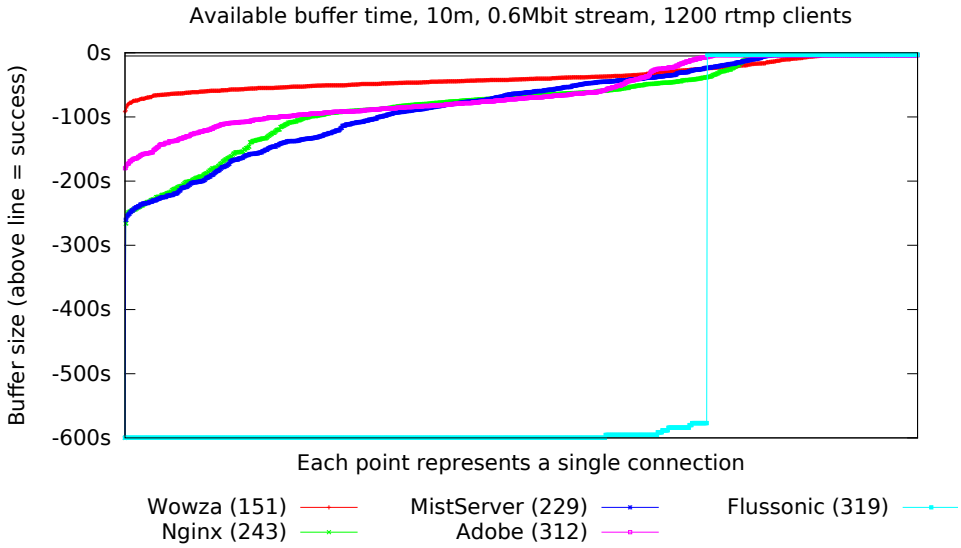
So - what happens at the point of 1200 clients, now? Let's take a look:


Smoothed CPU usage, 10m, 0.6Mbit stream, 1200 rtmp clients
CPU use
100 %
80 %
60 %
40 %
20 %
0 %
00:00 01:00 02:00 03:00 04:00 05:00 06:00 07:00 08:00 09:00 10:00 11:00
Wowza (151)
MistServer (229)
Flussonic (319)
Nginx (243)
Adobe (312)


Memory usage, 10m, 0.6Mbit stream, 1200 rtmp clients
Memory use
12000 MiB
10000 MiB
8000 MiB
6000 MiB
4000 MiB
2000 MiB
0 MiB
00:00 01:00 02:00 03:00 04:00 05:00 06:00 07:00 08:00 09:00 10:00 11:00
Wowza (151)
MistServer (229)
Flussonic (319)
Nginx (243)
Adobe (312)



These graphs don't show many surprises. CPU and memory usage are right where you'd expect them, and the upload graph shows Flussonic still dropping many users. Lucky for Flussonic, this actually makes them more stable in this overload situation, and they manage the most successful connections (319), closely followed by Adobe (312). MistServer (229) and Nginx (243) take the middle and Wowza (151) performs worst. Let's bring out the buffer time graph again, and see how these number visualize:



It's hard to tell, but those lines are actually 1200 dots, each. Each dot represents a single connection, with the best connections on the right and the worst connections on the left. Ideal would be a straight line at 0s, which is physically impossible in this test.

Depending on your preferences, you could name several servers winner in this category:

Flussonic has the highest count of successful connections, but also the highest count of hard failures (no data received at all). You either get a perfect stream, or no stream at all.

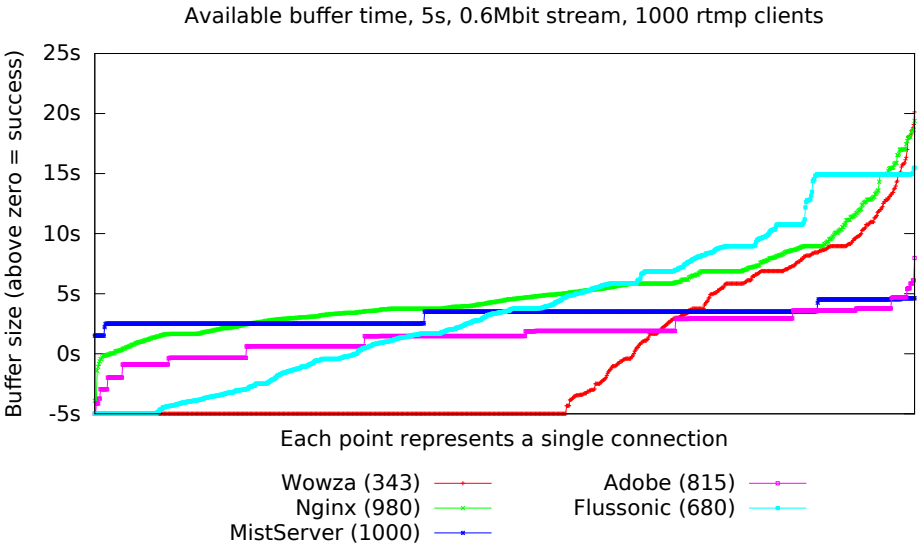
Wowza sends most data to most connections, but has the lowest count of successful connections. Everyone can receive the stream, but almost nobody will receive it without client-side buffering.

MistServer, Adobe and Nginx all take a middle point, serving some connections perfectly while most others can still receive the stream but will experience more buffering.

6. Burst test results

Finally, the new burst test. In this test, the same type of measurement is taken as above, but we only keep the connection open for 5 seconds. Any connection that receives at least 5 seconds of media data in those 5 seconds, is deemed successful.

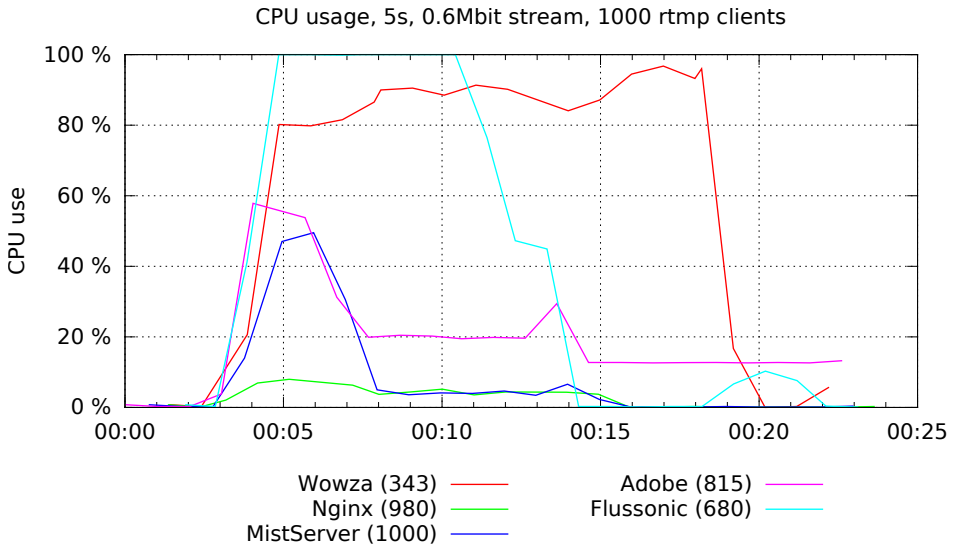
Here's the buffer time graph for this test at 1000 connections:



Oddly, MistServer is the only software handling this perfectly with 1000 successful connections. Nginx and Adobe follow closely, at 980 and 815 connections respectively. Next is Flussonic, able to handle 680 connections, and finally Wowza at 343 connections.

The shape of this graph really tells the whole story. It's clear that MistServer assures the connections are treated more or less equally, while Nginx and Wowza treat them on a first-come-first-serve principle, some of the first connections having received as much as 20 seconds of buffer time in the first 5 seconds. Flussonic seems to cap connections at roughly 15 seconds of buffer time, explaining the straight line on the right side. Adobe most closely follows the behaviour MistServer has, but prioritizes some connections slightly still, resulting in some others not getting enough buffer, but not very many.

Let's take a quick look at the CPU usage of this load too:



A pretty clear picture. Nginx is obviously optimized for this kind of load, which makes sense as webservers are often hit with exactly this kind of traffic. It picks up all 1000 connections without as much as a raised eyebrow. MistServer takes second place, showing a curve at the beginning, but stabilizing quickly. Then Adobe follows, showing a similar curve but taking longer to stabilize and still showing effects of the burst long after the connections go away. Flussonic shows the largest curve, but it does stabilize quickly after. And lastly Wowza, showing a big curve and taking a long time to stabilize.

7. Analysis

What do all these results mean with regards to our research question?

First let's talk about the updated tests compared to the Q1 performance data. There's a noticable improvement in most areas, particularly in Wowza's results. It looks like they read our previous performance report and took it to heart, which is something we definitely applaud and appreciate. The new Adobe and Flussonic results definitely help complete the picture of where the current state of the art for media servers is in terms of performance, giving us a solid baseline set by multiple media servers written in different programming languages to compare against.

The new burst test results are of course the most interesting part of this paper. They clearly show that most media servers aren't designed with this type of load in mind.

Should they be?

At DDVTech we believe that more and more automation is going to happen, whether we like it or not. Computers aren't subtle things, which means this type of load will become a common thing in the future as systems will be turned on and off as needed. We do these kinds of tests to see if our solutions220.19 are ready for the future and to make sure we're not lacking in any areas we may not have paid as much attention to. Results like this tell us we're on the right path, and we'll continue to push the envelope in this way.

8. Conclusion

These results are encouraging, but every time we further enhance our performance testing methods we find areas we wish we had taken a closer look at.

Yes, the data included with this publication is much more complete than last time around and nearly all major media servers are currently represented, but we're still only testing on a single system, using a single OS and measuring a single protocol (RTMP). Our next publication will focus on these points and widen the scope of this initiative.

We believe it's key for media servers to maintain or surpass the performance standards set by the web servers we all already are familiar with, while adding domain knowledge about streaming media to perform tasks either better than web servers can or tasks that web servers are not able to perform at all. This research initiative is all about striving to meet and exceed this vision for the future.

Anyone with feedback on these results or interested in joining our quest for accurate and unbiased data is welcome to contact us at info@ddvtech.com — let us know what you think and what aspects you'd like to see data on!

9. Performance data summery in table form

Table 1. Burst test scores (1000 = perfect)

Wowza	Nginx	MistServer	Adobe	Flussonic
343	980	1000	815	680

Table 2. Successful client counts

Clients	1			100			1000			1200		
Time	1m	3m	10m	1m	3m	10m	1m	3m	10m	1m	3m	10m
Wowza	1	1	1	100	100	100	433	848	1000	39	284	151
Nginx	1	1	1	100	100	100	896	818	998	562	286	243
MistServer	1	1	1	100	100	100	991	998	999	615	300	229
Adobe	1	1	1	100	100	100	758	621	1000	351	372	312
Flussonic	1	1	1	100	100	100	332	595	499	849	266	319

Table 3. Total bandwidth used per client in KiB

Time	1m	3m	10m
Wowza	10674	21026	51580
Nginx	51569	51563	51587
MistServer	5228	15715	51697
Adobe	53147	52956	53033
Flussonic	6077	17102	51740

Table 4. Idle memory usage in MiB

Wowza	Nginx	MistServer	Adobe	Flussonic
710.42	170.40	169.86	182.05	220.19

Table 5. Memory usage per client in MiB

Client count	Wowza	Nginx	MistServer	Adobe	Flussonic
1	91.61	1.03	3.02	80.66	9.13
100	8.57	0.86	1.05	0.91	1.12
1000	8.50	0.77	1.06	0.20	0.35
1200	0.84	0.07	0.10	0.02	0.02
Average	27.63	0.68	1.31	20.45	2.65

Table 6. CPU usage per client in percentage

Client count	Wowza	Nginx	MistServer	Adobe	Flussonic
1	0.395	0.166	0.346	0.324	0.522
100	0.060	0.010	0.023	0.017	0.036
1000	0.048	0.011	0.020	0.021	0.018
1200	0.054	0.010	0.020	0.015	0.010
Average	0.139	0.049	0.102	0.94	0.146