HTML5 Speedtest

Version 4.0, March 2 2017

Introduction

In this document, we will introduce an XHR based HTML5 Speedtest and see how to use it. This test measures download speed, upload speed, ping and jitter.

First of all, the requirements to run this test:

- Only modern browsers are supported: Microsoft Edge 12+, Mozilla Firefox (last 2 versions), Google Chrome/Chromium (last 2 versions), Apple Safari (last version). Javascript must be enabled. The test can run on any browser that supports XHR Level 2 and Web Workers.
- Client side, the test can use up to 500 megabytes of RAM
- Server side, you'll need a fast connection (at least 100 Mbps recommended), and your web server must accept large POST requests (up to 20 megabytes). Apache and PHP are recommended, but not mandatory.

If this looks good, let's proceed and see how to use the test.

Installation

To install the test on your server, upload the following files:

- speedtest_worker.min.js
- garbage.php
- getIP.php
- empty.dat

You may also want to upload one of the examples to test it.

Later we'll see how to use the test without PHP.

Important: keep all the files together; all paths are relative to the js file

Usage

To run the test, you need to do 3 things:

- Create the worker
- Write some code that handles the responses coming from the worker
- Start the test

Creating the worker

```
var w=new Worker("speedtest_worker.min.js");
```

Important: use the minified version, it's smaller!

Response handler

First, we set up a timer that fetches the status of the worker continuously:

```
var timer=setInterval(function(){
      w.postMessage("status");
}.bind(this),100);
```

Then we write a response handler that receives the status and updates the page. Later we'll see the details of the format of the response.

```
w.onmessage=function(event){
      var data=event.data.split(";");
      var testState=data[0],
            dlStatus=data[1],
            ulStatus=data[2],
            pingStatus=data[3],
            jitterStatus=data[5],
            clientIp=data[4];
      if(testStatus>=4) clearInterval(timer); //test is finished or aborted

      //update your page here

}.bind(this);
```

Response format

The response from the worker is composed of values separated by ; (semicolon) in this format:

testState;dlStatus;ulStatus;pingStatus;clientIp;jitterStatus

- **testState** is an integer 0-5
  - 0=Test starting
  - 1=Download test in progress
  - 2=Ping+Jitter test in progress
  - 3=Upload test in progress
  - 4=Test finished
  - 5=Test aborted
- **dlStatus** is either
  - Empty string (not started or aborted)
  - Download speed in Megabit/s as a number with 2 digits
  - The string "Fail" (test failed)
- **ulStatus** is either
  - Empty string (not started or aborted)
  - Upload speed in Megabit/s as a number with 2 digits
  - The string "Fail" (test failed)
- **pingStatus** is either
  - Empty string (not started or aborted)
  - Estimated ping in milliseconds as a number with 2 digits
  - The string "Fail" (test failed)
- **clientIp** is either
  - Empty string (not fetched yet or failed)
  - The client's IP address as a string
- **jitterStatus** is either
  - Empty string (not started or aborted)
  - Estimated jitter in milliseconds as a number with 2 digits (lower=stable connection)
  - The string "Fail" (test failed)

Starting the test

To start the test, send the start command to the worker:

```
w.postMessage('start');
```

This starts the test with the default settings, which is usually the best choice. If you want, you can change these settings and pass them to the worker as JSON with like this:

```
w.postMessage('start {"param1":"value1", "param2":"value2", ...}');
```

Test parameters

- **time_dl**: How long the download test should be in seconds

  Default: 15   Recommended >=5
- **time_ul**: How long the upload test should be in seconds

  Default: 15   Recommended >=10
- **count_ping**: How many pings to perform in the ping test

  Default: 35   Recommended >=20
- **url_dl**: path to garbage.php or a large file to use for the download test

  Default: "garbage.php"
  Important: path is relative to js file
- **url_ul**: path to ab empty file or empty.dat to use for the upload test

  Default: "empty.dat"
  Important: path is relative to js file
- **url_ping**: path to an empty file or empty.dat to use for the ping test

  Default: "empty.dat"
  Important: path is relative to js file
- **url_getIp**: path to getIP.php or replacement

  Default: "getIP.php"
  Important: path is relative to js file
- **enable_quirks**: enables browser-specific optimizations. These optimizations override some of the default settings below. They do not override settings that are explicitly set.

  Default: true
- **garbagePhp_chunkSize**: size of chunks sent by garbage.php in megabytes

  Default: 20   Recommended >=10
- **xhr_dlMultistream**: how many streams should be opened for the download test

  Default: 10   Recommended >=3
  Default override: 3 on Edge if enable_quirks is true
  Default override: 1 on Safari if enable_quirks is true
- **xhr_ulMultistream**: how many streams should be opened for the upload test

  Default: 3   Recommended >=1
  Default override: 1 on Firefox if enable_quirks is true
  Default override: 1 on Safari if enable_quirks is true
- **allow_fetchAPI**: allow the use of Fetch API for the download test instead of regular XHR. Experimental, not recommended.

  Default: false
- **force_fetchAPI**: forces the use of Fetch API on all browsers that support it

  Default:false

Fetch API are used if the following conditions are met:

- allow_fetchAPI is true
- Chromium-based browser with support for Fetch API and enable_quirks is true OR force_fetchAPI is true and the browser supports Fetch API

Aborting the test prematurely

The test can be aborted at any time by sending an abort command to the worker:

```
w.postMessage('abort');
```

This will terminate all network activity and stop the worker.

Important: do not simply kill the worker while it's running, as it will leave pending XHR requests!

Using the test without PHP

If your server does not support PHP, or you're using something newer like Node.js, you can still use this test by replacing garbage.php and getIP.php.

Replacements

Replacement for garbage.php

A replacement for garbage.php must generate incompressible garbage data.

A large file (10-100 Mbytes) is a possible replacement. You can get one here: http://downloads.fdossena.com/geth.php?r=speedtest-bigfile

If you're using Node.js or some other server, your replacement should accept the ckSize parameter (via GET) which tells it how many megabytes of garbage to generate. It is important here to turn off compression, and generate incompressible data.

A symlink to /dev/urandom is also ok.

Replacement for getIP.php

Your replacement must simply respond with the client's IP as plaintext. Nothing fancy.

JS

You need to start the test with your replacements like this:

```
w.postMessage('start {"url_dl":"newGarbageURL", "url_getIp":"newIpURL"}
```

Known bugs and limitations

The only limitation currently known of is the high CPU usage from XHR requests on Chrome with very fast connections (like gigabit).

License