JDownloader - Feature #50863

Review: setRequestIntervalLimitGlobal(host, i, x, y)

09/03/2014 06:33 AM - raztoki

Status:	New	**Start date:**	09/02/2014
Priority:	Normal	**Due date:**	
Assignee:	jiaz	**% Done:**	0%
Category:	General	**Estimated time:**	0.00 hour
Target version:	020 - Next Release 2.0		
Resolution:			

Description

so setRequestIntervalLimitGlobal(host, i) has linear design, it allows new request i (ms) after previous request. When as site has a limit you have to setRequestIntervalLimitGlobalit(host, i) higher enough so that JD don't trigger problem or get IP blocked.

Now if the site allows for burstible requests to be sent before wait time mandated, we haven't been able support this until now. So I've added setRequestIntervalLimitGlobal(host, i, x, y) setter, and implemented new code in waitForPageAccess to keep track of each request.

setRequestIntervalLimitGlobal(host, i, x, y)
host = domain [1] (String) [1]
i = wait int between request (int) [1]
x = requests (int)
y = time interval in ms (long)

[1] host and i are set with old method setRequestIntervalLimitGlobal(host , i) (old implementation is reused)
x = upper threshold of requests allowed before triggers problem with host
y = time frame in which x is allowed before triggering problem with host
we think of this as x over y (x/y)

waitForPageAccess, original method changed a little.
Cleanup of ArrayList<Long> takes place, by analysing time stamps in array. if any are older than y value, its removed.
The ArrayList<Long>.size() is used to compare against x value.
If under x threshold, we do not have to obey any thread.sleep (this is what supports bursting), we return out of waitForPageAccess.
else when over x value, {
wait1 = checks oldest timestamp in ArrayList<Long> (should always be entry 0), to see when it will expire against REQUEST_INTERVAL_LIMIT_MAP,
wait2 = REQUEST_INTERVAL_LIMIT_MAP - (current system time - last global request[REQUESTTIME_MAP])
which either is the less of the two, it waits that time.
}
In the finally clause, a new timestamp added ArrayList<Long>, and then HashMap.put(host, ArrayList<Long>)

conclusion
Implementation allows for bursting of requests in short succession (without wait intervals), then on after allows for dynamic throttling of requests.