**SPECS:**

CPU: Intel® Core™ i5-6200U CPU @ 2.30 GHz

Memory: 8.0 GB

**Php Socket.IO:**

**Time for server receiving client-data:**

**Method:**

Client-side code:

<**script**>**for**(**var *i***=0; ***i***<=220000; ***i***++){  
 **var *socket*** = **io**(**'http://127.0.0.1:3120'**);  
 ***socket***.emit(**'testEvent'**, ***i***);  
 }  
</**script**>

Client-side code is a simple script consisting of a for-loop emitting 220,000 events to server-side.

Server-Side code:

$io = **new** SocketIO(3120);  
$io->on(**'connection'**, **function**($socket)**use**($io){  
 $time\_pre = *microtime*(**true**);  
 $socket->on(**'testEvent'**, **function**($msg)**use**($socket, $time\_pre){  
 *//echo $msg . "\n";* **if**($msg == 220,000){  
 $time\_post = *microtime*(**true**);  
 **echo** ($time\_post - $time\_pre);  
 }  
 });  
});

$time\_pre was defined immediately upon a new client connection.

When $msg is equal to 220,000, time is recorded and the time difference is calculated to obtain the total time consumed to receive these events.

Note: I set the limit to 220,000 because my machine only allocates so many bits for PHP to use for calculations. Anywhere beyond 220,000 would receive the following error:

Fatal error: Allowed memory size of 134217728 bytes exhausted (tried to allocate 50033 bytes)

**Data:**

|  |  |
| --- | --- |
| Trial number | Seconds |
| 1 | 17.236232995987 |
| 2 | 17.676249980927 |
| 3 | 18.23201918602 |
| 4 | 17.616749048233 |
| 5 | 17.373829841614 |

Average time: 17.621 seconds

Time for client receiving server data:

Client-side code:

***socket***.on(**'testEvent'**, **function**(data){  
 **if**(data == 10000){  
 **var** time\_post = Date.now();  
 **console**.log((time\_post - ***time\_pre***) \* 0.001);  
 }  
})

Server-side code:

$io = **new** SocketIO(3120);  
$io->on(**'connection'**, **function**($socket)**use**($io){**for**($i = 0; $i <= 10000; $i++){  
 $socket->emit(**"testEvent"**, $i);  
 }  
 **echo "done\n"**;  
});

similar method was used to measure time cost for data sent from server-side to client-side, except the upper bound of for-loop was much smaller (10,000 in this case), anywhere beyond this causes my browser to freeze, this maybe due to the limitation of my machine.

**Data:**

|  |  |
| --- | --- |
| Trial number | Seconds |
| 1 | 2.031 |
| 2 | 2.051 |
| 3 | 1.981 |
| 4 | 2.004 |
| 5 | 2.03 |

Average time: 2.0194 seconds

**Client-side Load Testing:**

Client-side code:

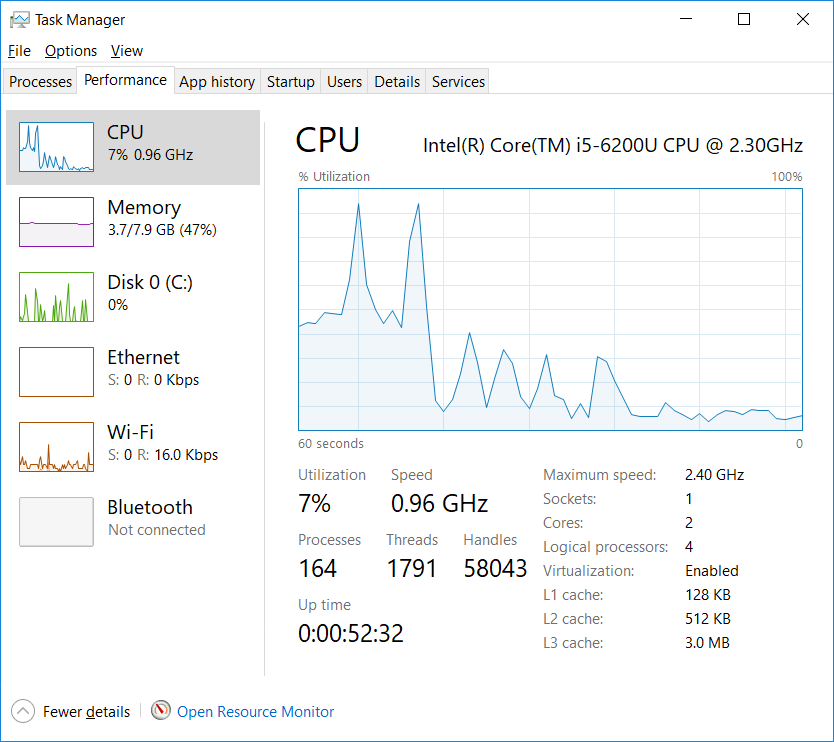
**function** *newConnection*(){  
 **if**(***totalUserCnt*** < ***maxUsers***){  
 ***totalUserCnt***++;  
 **var** socket = **io**(**'http://127.0.0.1:3120'**);  
 socket.emit(**'testEvent'**);  
 **var** time\_pre = Date.now();  
 socket.on(**'testReceiveEvent'**, **function**(data){  
 **if**(data == ***cutOffEvent***){  
 **var** t = ((Date.now() - time\_pre) \* 0.001);  
 ***avgTime***.push(t);  
 **console**.log(***avgTime***.**length** + **': '** + t);  
 **if**(***avgTime***.**length** >= ***maxUsers***) {  
 **var** time = 0;  
 ***avgTime***.forEach(**function** (t) {  
 time += t;  
 });  
 **console**.log(time / ***avgTime***.**length**);  
 }  
 }  
 });  
 setTimeout(*newConnection*, 100);  
 }  
}  
  
*newConnection*();

//avgTime is an array that keeps track of the response time for all the connections, when all the connections are done receiving server emitted events, the average of avgTime is calculated to obtain the average response time for all the connections.

Server-side code:

$io->on(**'connection'**, **function**($socket)**use**($io, $userCnt, $maxUserCnt, $cutOffEvent){  
 $userCnt++;  
 **if**($userCnt == $maxUserCnt){  
 **echo "Reached max user count!\n"**;  
 }  
 $socket->on(**'testEvent'**, **function**()**use**($socket, $cutOffEvent){  
 **for**($i=0; $i<=$cutOffEvent; $i++){  
 $socket->emit(**'testReceiveEvent'**, $i);  
 }  
 });  
});

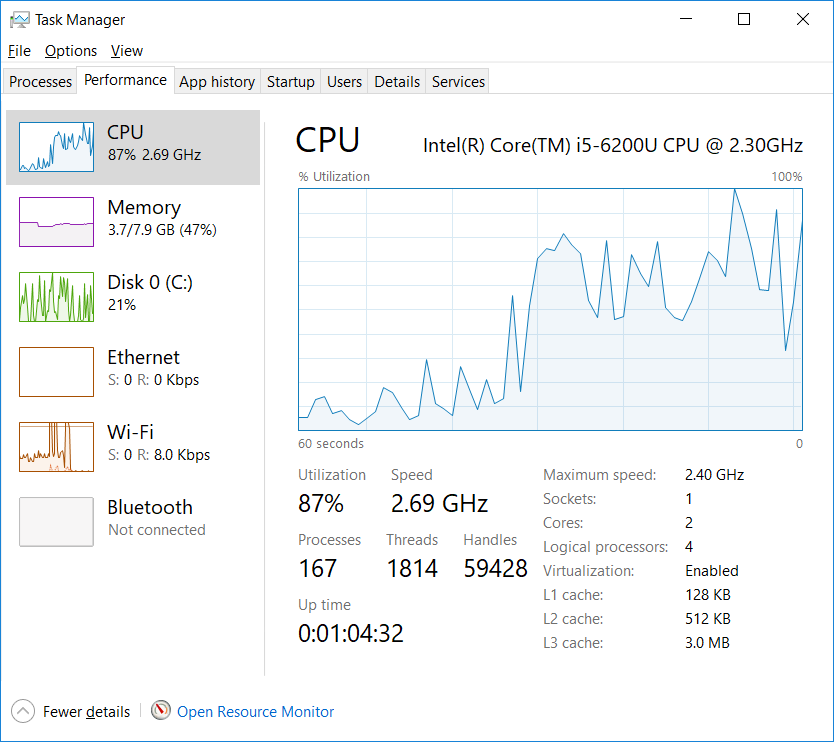
CPU and memory status before running:



**100 connections:**

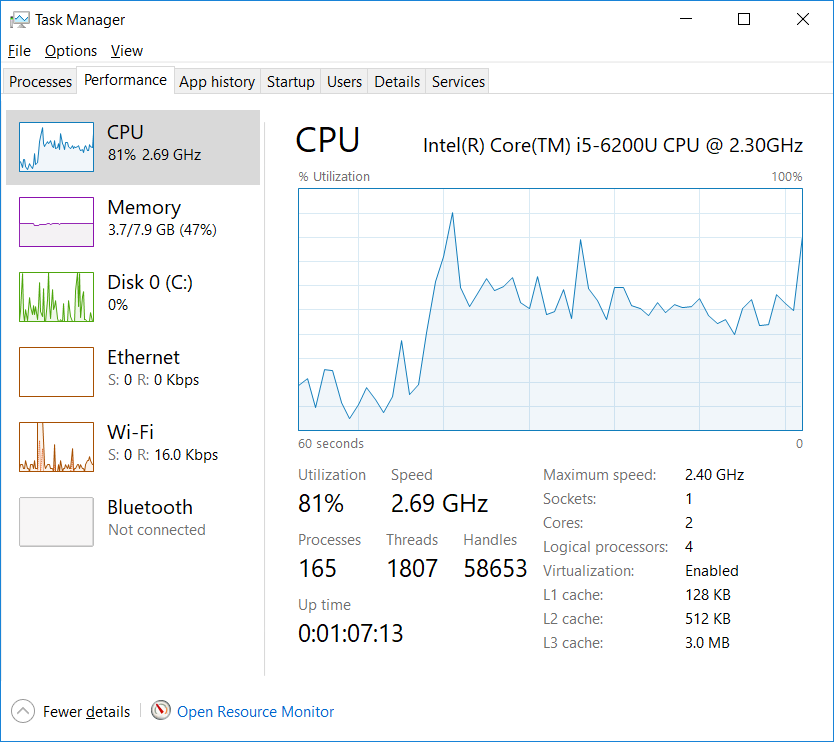
Average time for each connection to receive 5,000 events from server: 2.67 seconds

CPU and memory status:



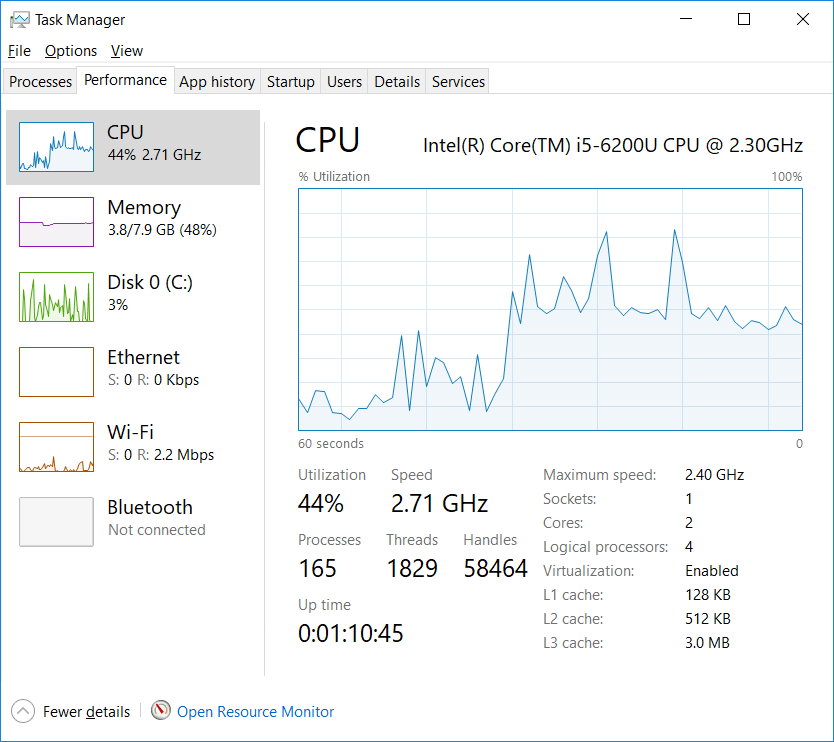
Average time for each connection to receive 8,000 events from server: 7.6858 seconds

CPU and memory status:



Average time for each connection to receive 10,000 events from server: Client failed and never completed the task.

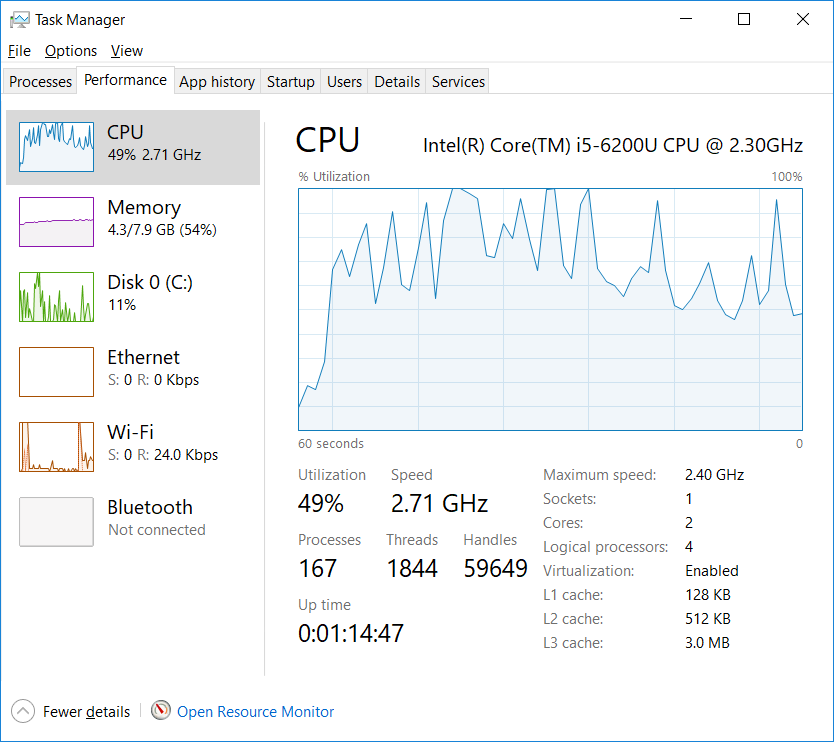
CPU and memory status:



**200 connections:**

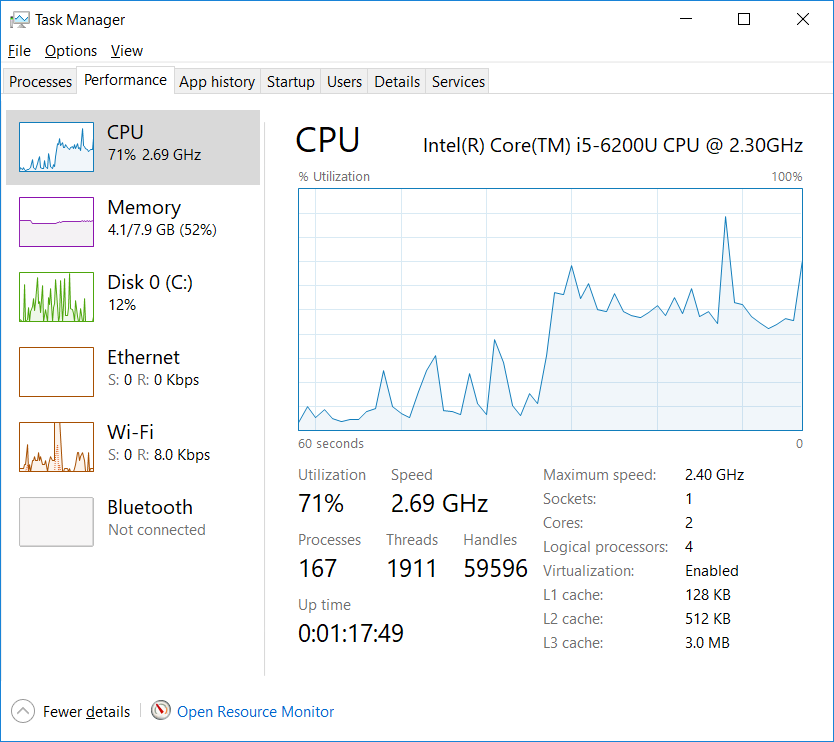
Average time for each connection to receive 5,000 events from server: 2.304 seconds

CPU and memory status:



Average time for each connection to receive 8,000 events from server: Server failed and never completed the task.

CPU and memory status:



**300 connections:**

Client failed the task and stopped at 255 connections.

**Server-side Load Testing:**

Client-side code:

**function** *newConnection*(){  
 **if**(***totalUserCnt*** < ***maxUsers***){  
 ***totalUserCnt***++;  
 **var** socket = **io**(**'http://127.0.0.1:3120'**);  
 **for**(**var** i=0; i<***cutOffEvent***; i++){  
 socket.emit(**'testEvent'**);  
 }  
 **console**.log(***totalUserCnt***);  
 setTimeout(*newConnection*, 100);  
 }  
}  
  
*newConnection*();

Server-side code:

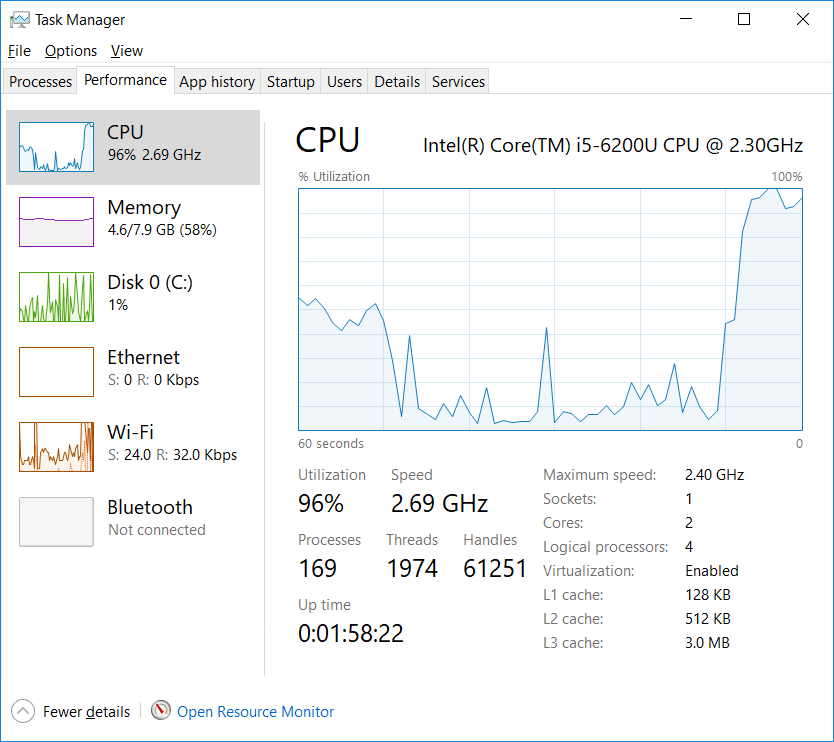
$io->on(**'connection'**, **function**($socket)**use**($io, $userCnt, $maxUserCnt, $cutOffEvent, $time\_pre, &$eventCnt){  
 $userCnt++;  
 **if**($userCnt == $maxUserCnt){  
 **echo "Reached max user count!\n"**;  
 }  
 $socket->on(**'testEvent'**, **function**()**use**($socket, $cutOffEvent, &$eventCnt, $time\_pre){  
 $eventCnt++;  
 **if**($eventCnt == $cutOffEvent){  
 $time\_post = *microtime*(**true**);  
 **echo** $time\_post - $time\_pre;  
 }  
 });  
});

A new connection is created on client-side every 100 ms, each connection emitting ‘cutOffEvent’ amounts of events. When server-side finishes receiving all these event, total response time is calculated and outputted.

**100 connections:**

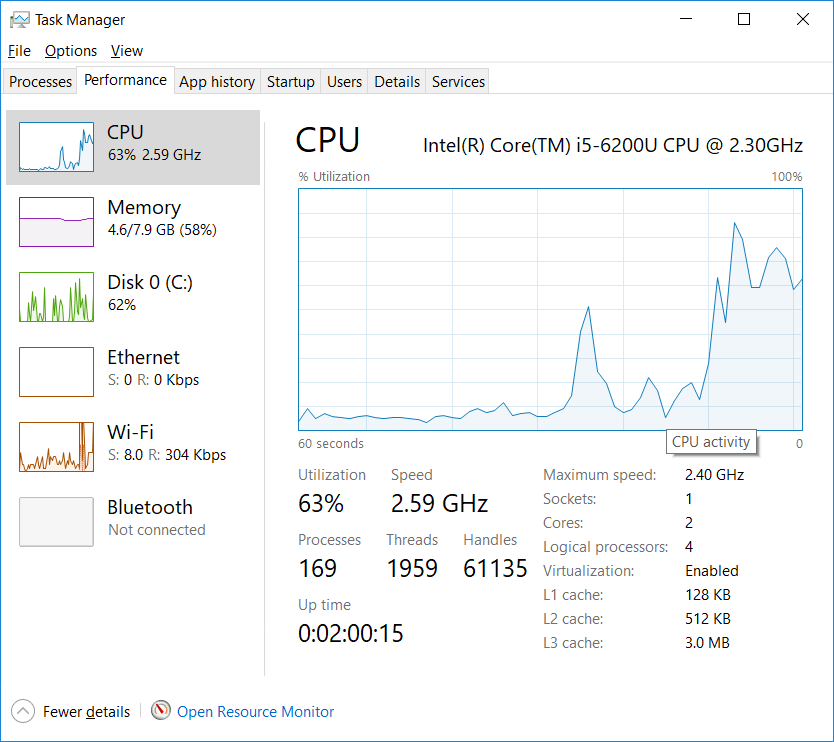
Total time for server to receive 100,000 events (100 per connection): 4.04 seconds

CPU and memory status:



Total time for server to receive 500,000 events (500 per connection): Server failed and never completed the task.

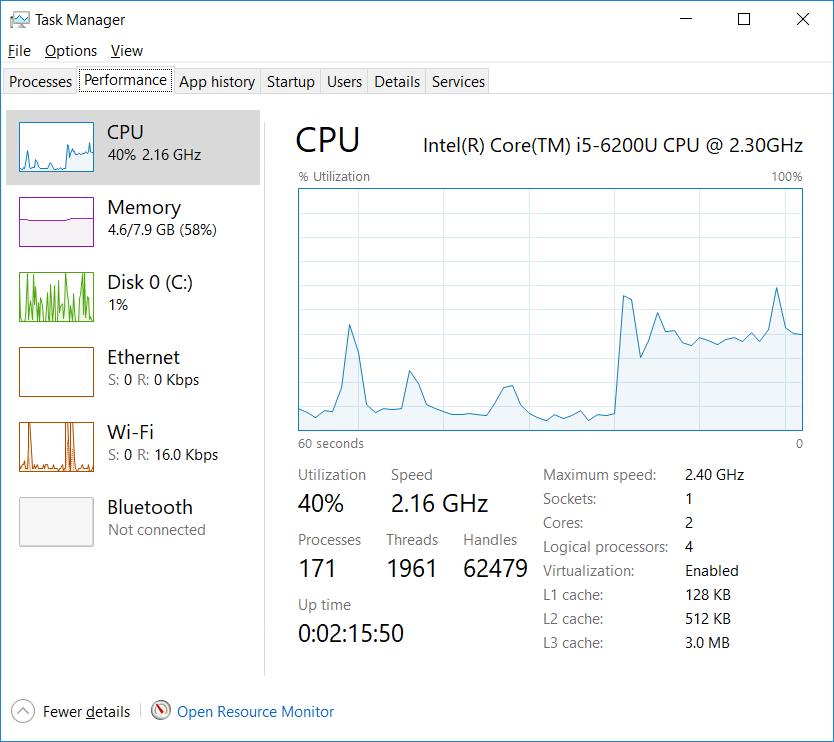
CPU and memory status:



**200 connections:**

Total time for server to receive 200,000 events (100 per connection): 28.287 seconds

CPU and memory status:



**Socket.IO on Javascript**

Client to server data transmission time:

Client-side code:

var socket = io();

for(var i=0; i<=220000; i++){

socket.emit('testEvent', i);

}

Similar to php’s client-side, a for loop emitting 220,000 events is used.

Server-side code:

io.on('connection', function(socket){

var time\_pre = Date.now();

socket.on('testEvent', function(data){

if(data == 220000){

var time\_post = Date.now();

console.log((time\_post - time\_pre) \* 0.001);

}

})

});

**Data:**

|  |  |
| --- | --- |
| Trial number | Seconds |
| 1 | 5.536 |
| 2 | 4.73 |
| 3 | 5.229 |
| 4 | 4.584 |
| 5 | 7.07 |

Average time cost: 5.4298 seconds

Server to client data transmission time cost:

Client-side code:

var time\_pre = Date.now();

socket.on('testEvent', function(data){

if(data == 220000){

var time\_post = Date.now();

console.log((time\_post - time\_pre) \* 0.001);

}

})

Client-side code:

$(function(){

var socket = io();

var time\_pre = Date.now();

socket.on('testEvent', function(data){

if(data == 220000){

var time\_post = Date.now();

console.log((time\_post - time\_pre) \* 0.001);

}

})

})

Surprisingly, Javascript was able to handle much larger data even when receiving events from server side. Hence, it is possible to transmit up to 220,000 events from server-side to clients.

**Data:**

|  |  |
| --- | --- |
| Trial number | Seconds |
| 1 | 7.205 |
| 2 | 6.818 |
| 3 | 4.513 |
| 4 | 5.078 |
| 5 | 4.776 |

Average time cost: 5.6778 seconds

**Client-side Load Testing:**

Client-side code:

**function** *newConnection*(){  
 **if**(***totalUserCnt*** < ***maxUsers***){  
 ***totalUserCnt***++;  
 **var** socket = io(**"http://localhost:3000"**, {**multipliex**: **false**});  
 socket.emit(**'testEvent'**, ***totalUserCnt***);  
 **var** time\_pre = Date.now();  
 socket.on(**'testReceiveEvent'**, **function**(data){  
 **if**(data == ***cutOffEvent***){  
 ***avgTime***.push(((Date.now() - time\_pre) \* 0.001));  
 **if**(***avgTime***.**length** >= ***maxUsers***){  
 **var** time = 0;  
 ***avgTime***.forEach(**function**(t){time += t;});  
 **console**.log(time / ***avgTime***.**length**);  
 }  
 }  
 });  
 setTimeout(*newConnection*, 100);  
}  
 }  
  
 *newConnection*();

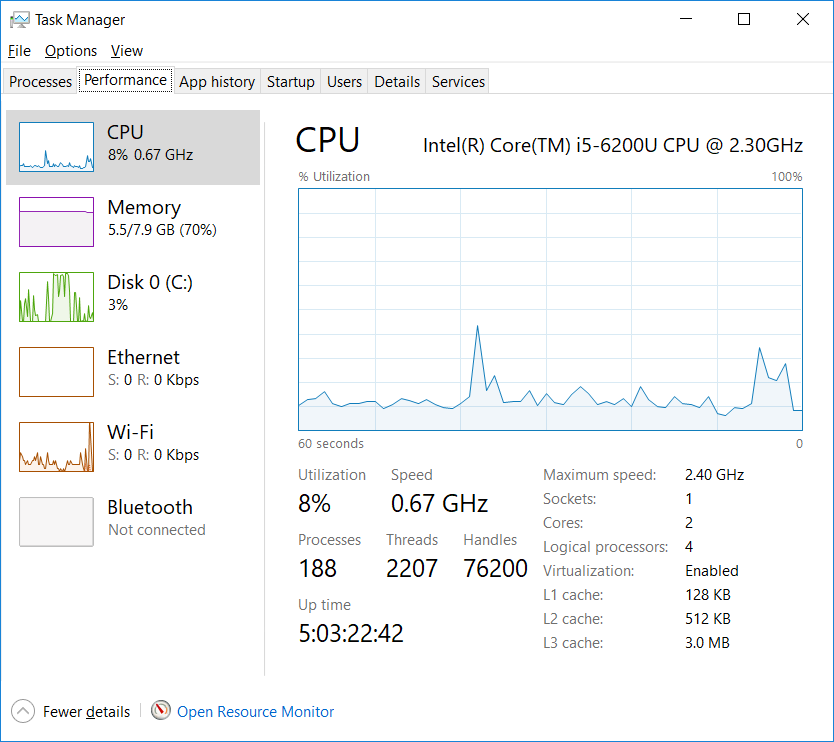
//avgTime is an array that keeps track of the response time for all the connections, when all the connections are done receiving server emitted events, the average of avgTime is calculated to obtain the average response time for all the connections.

Server-side code:

io.on(**'connection'**, **function**(socket){  
 userCnt++;  
 **if**(userCnt == maxUserCnt){  
 **console**.log(**"Reached max user count!"**);  
 }  
 socket.on(**'testEvent'**, **function**(data){  
 **for**(**var** i=0; i<=cutOffEvent; i++){  
 socket.emit(**'testReceiveEvent'**, i);  
 }  
 })  
});

A new connection is created every 100 ms and emits an event to server, upon receiving this event on server, server-side starts testing maximum events can be emitted to these connections and the response time by emitting “cutOffEvent” times of events.

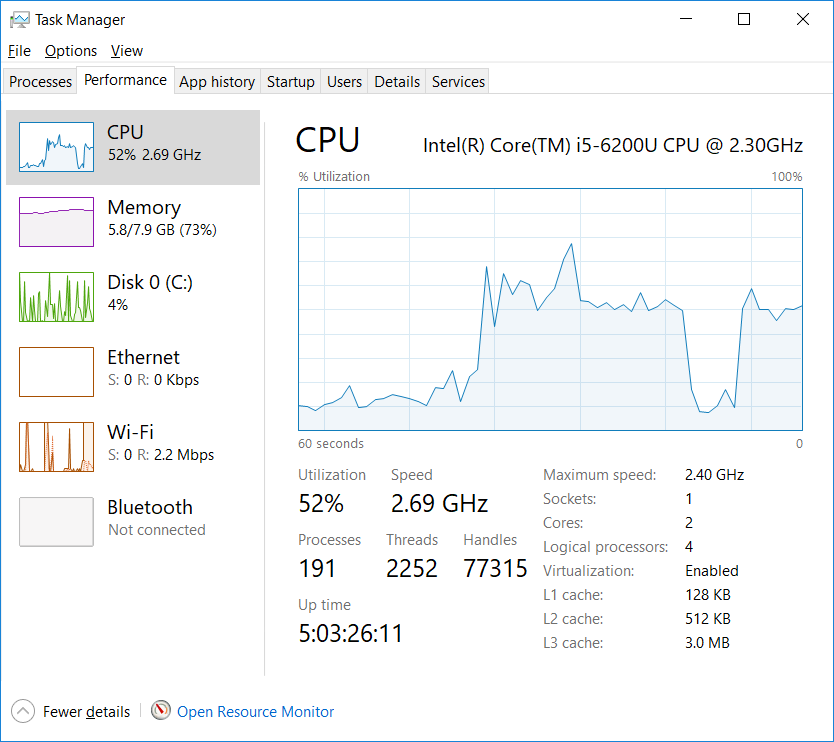
CPU and memory status before running:



**100 connections:**

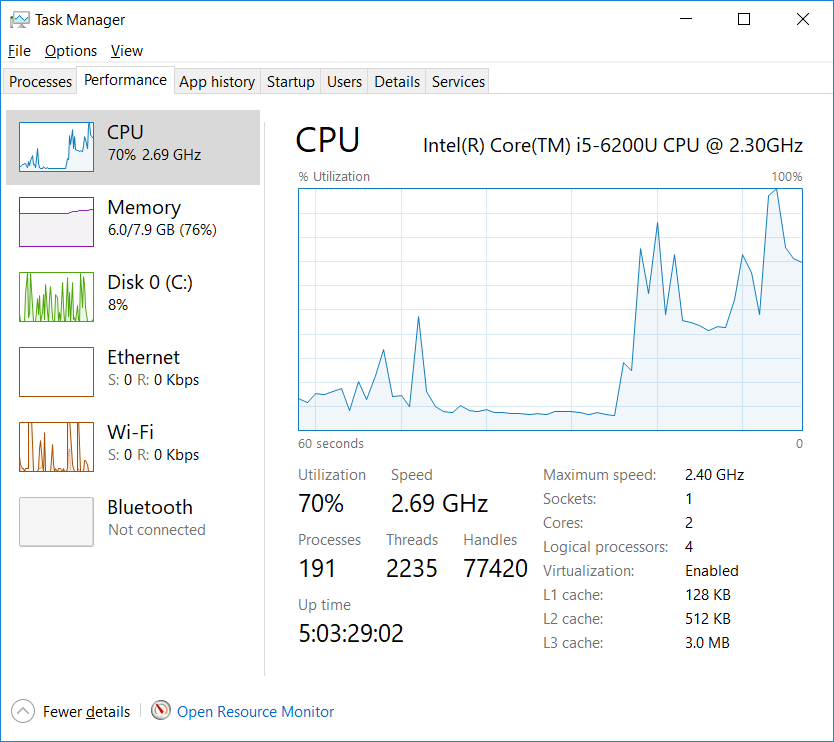
Average time for each user to receive 10,000 events emitted from server: 1.244 seconds

CPU and memory status while running:



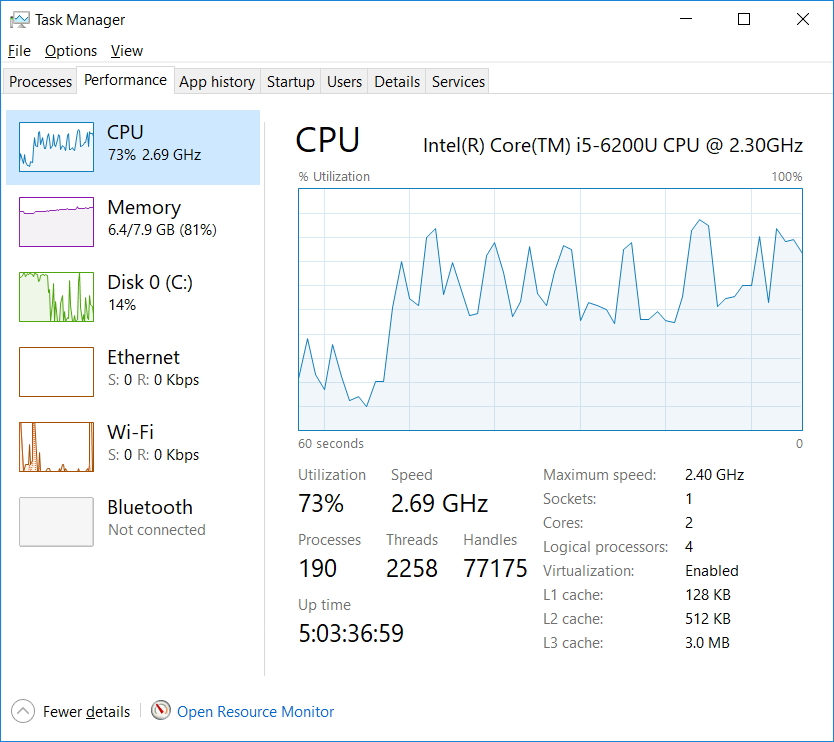
Average time for 50,000 events emitted from server: 8.09878 seconds

CPU and memory status while running:



Average time for 100,000 events emitted from server: 20.375 seconds

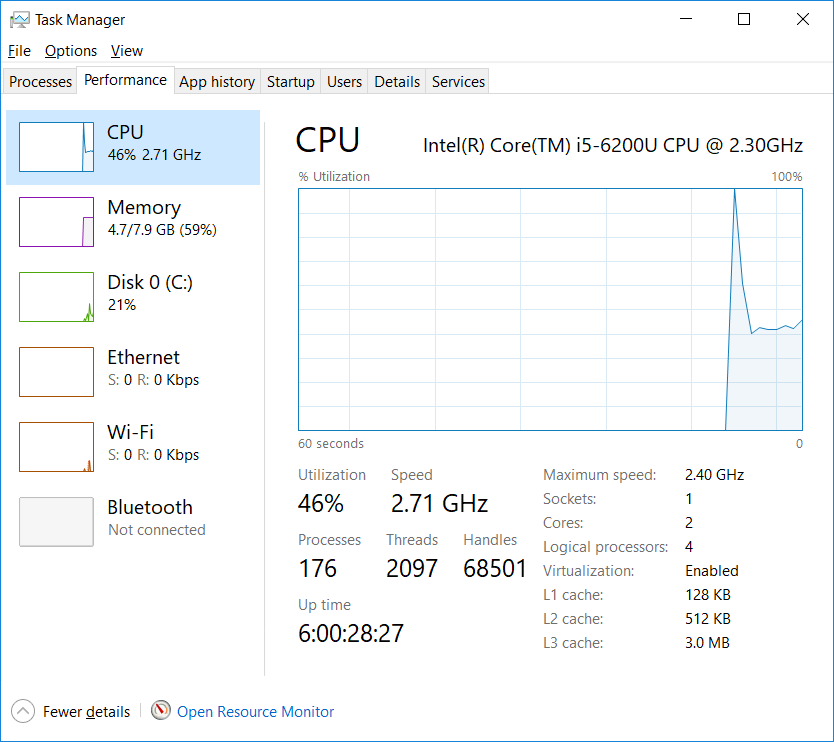
CPU and memory status while running:



**200 Connections:**

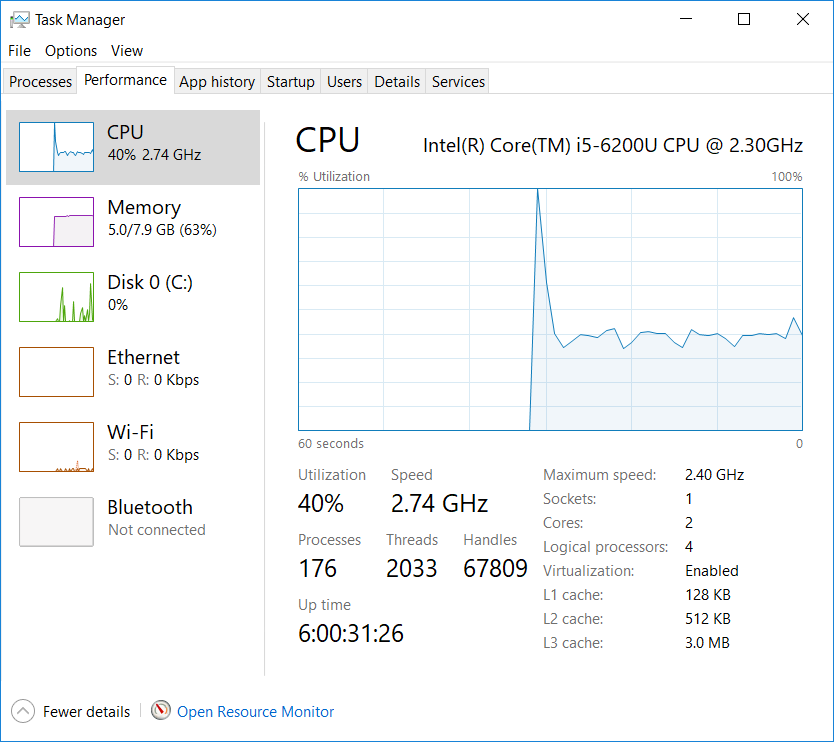
Average time for each user to receive 10,000 events emitted from server: 0.45 seconds

CPU and memory status:



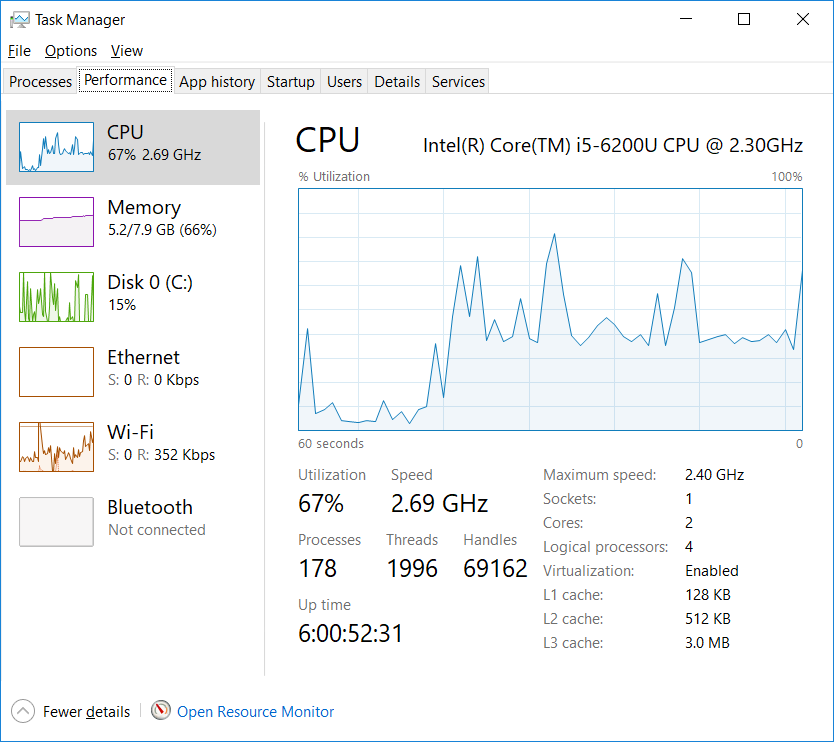
Average time for each user to receive 50,000 events emitted from server: 5.2239 seconds

CPU and memory status:



Average time for each user to receive 100,000 events emitted from server: 15.4798 seconds

CPU and memory status:

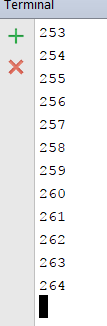


**300 connections:**

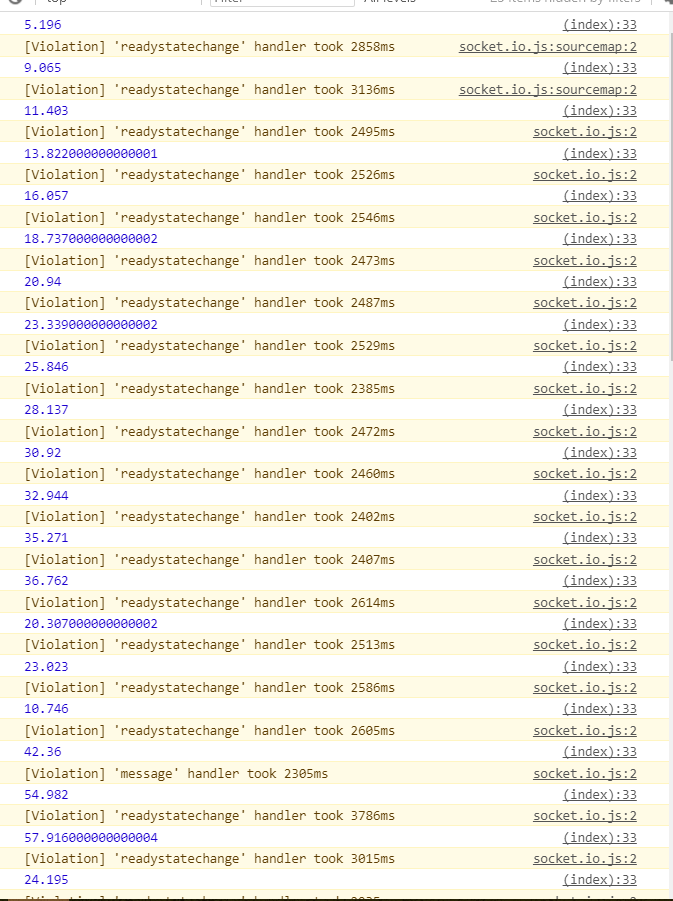
It appears that the maximum number of connections for Socket.IO is 263.



As the client-side console log stops outputting at 263 connections, and for server-side user count stops incrementing at 264.



**Observations:**

 I observed that the response time has a sudden jump from 10 seconds to 58 seconds in the beginning, then stabilizes at around 8 seconds. I’m willing to guess that the CPU has no cache of the task initially, so the response time was slow and long, as the CPU performs the same task repeatedly, more data gets cached and finally the response time gets stabilized at 8 seconds.

Presumably, this is also the cause for 200 connections having better performance than 100 connections, as the number of connections increase, the average response time converges to a more stable time.

**Server-side Load Testing:**

Client-side code:

**function** *newConnection*(){  
 **if**(***totalUserCnt*** < ***maxUsers***){  
 ***totalUserCnt***++;  
 **var** socket = io(**"http://localhost:3000"**, {**multipliex**: **false**});  
 **for**(**var** i=0; i<***cutOffEvent***; i++){  
 socket.emit(**'testEvent'**);  
 }  
 **console**.log(***totalUserCnt***);  
 setTimeout(*newConnection*, 100);  
}  
 }  
  
 *newConnection*();

Server-side code:

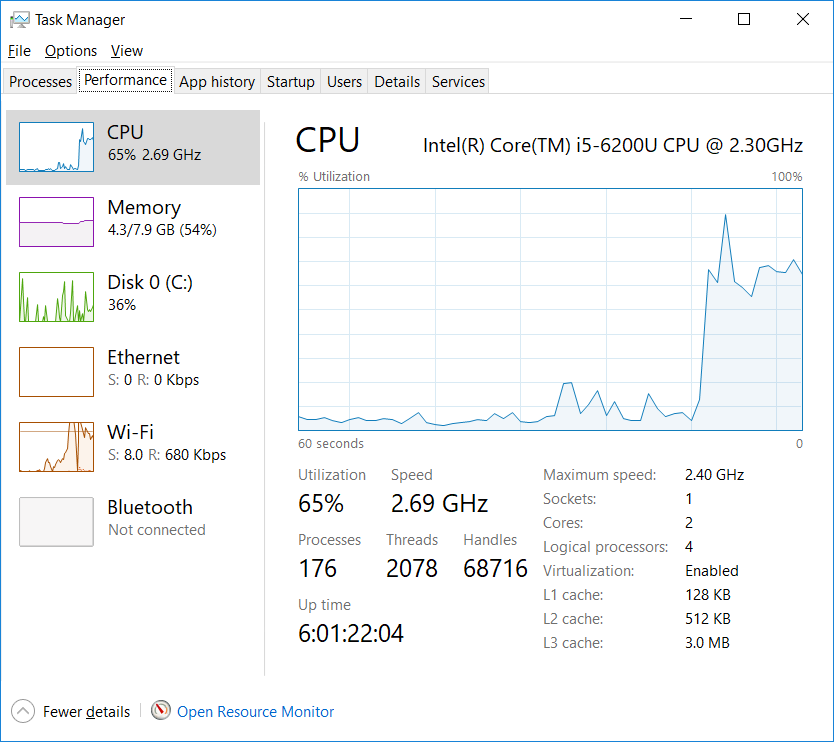
**var** userCnt = 0;  
**var** maxUserCnt = 100;  
**var** time\_pre = Date.now();  
**var** cutOffEventClient = 10000;  
**var** cutOffEvent = maxUserCnt \* cutOffEventClient;  
**var** eventCnt = 0;  
io.on(**'connection'**, **function**(socket){  
 userCnt++;  
 **if**(userCnt == maxUserCnt){  
 **console**.log(**"Reached max user count!"**);  
 }  
 socket.on(**'testEvent'**, **function**(){  
 eventCnt++;  
 **if**(eventCnt == cutOffEvent){  
 **var** total\_time = ((Date.now() - time\_pre) \* 0.001);  
 **console**.log(total\_time);  
 }  
 })  
});

A new connection is created on client-side every 100 ms, each connection emitting ‘cutOffEvent’ amounts of events. When server-side finishes receiving all these event, total response time is calculated and outputted.

**100 connections:**

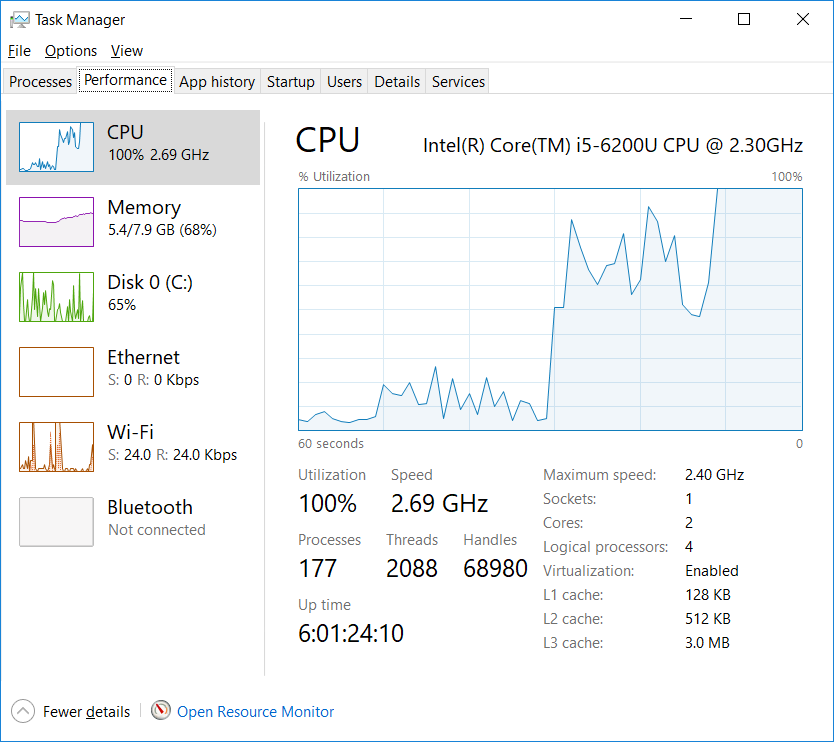
Total time for server to receive 1,000,000 events (10,000 events per connection): 21.792 seconds

CPU and memory status:



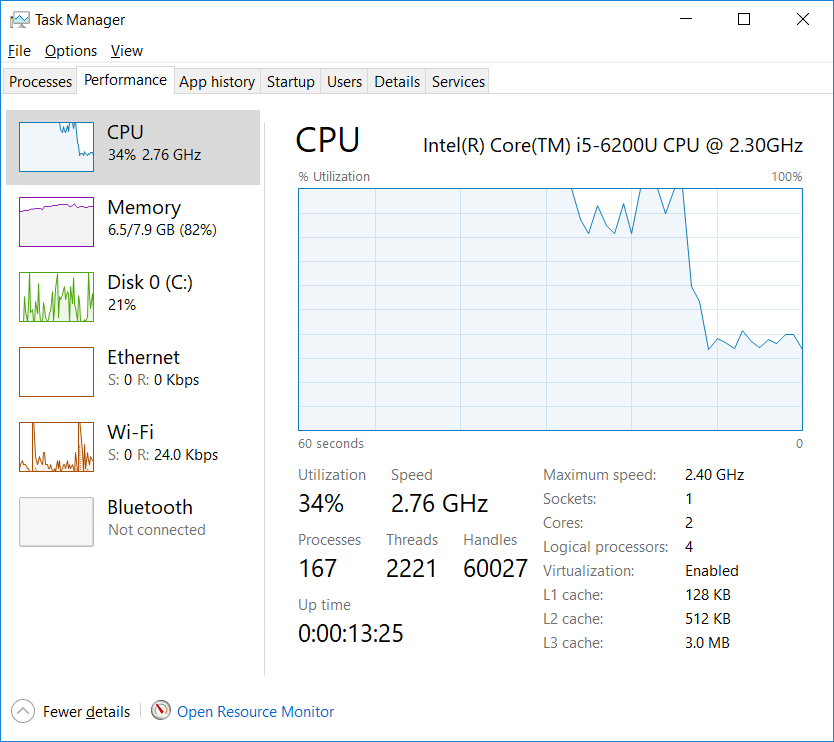
Total time for server to receive 2,000,000 events (20,000 each connection): 62.966 seconds

CPU and memory status:



Total time for server to receive 3,000,000 events (30,000 each connection): 79.015 seconds

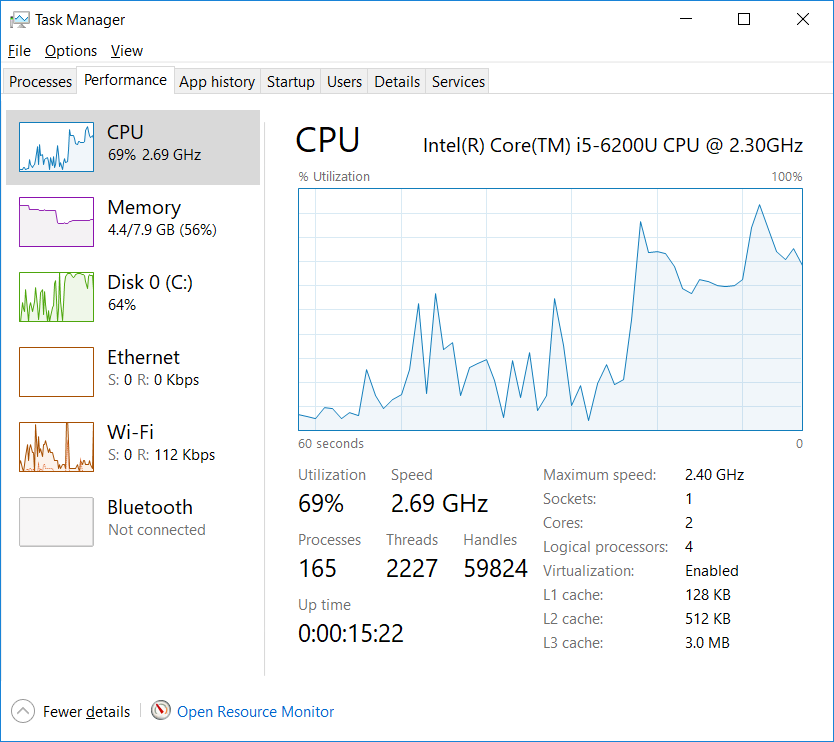
CPU and memory status:



**200 connections:**

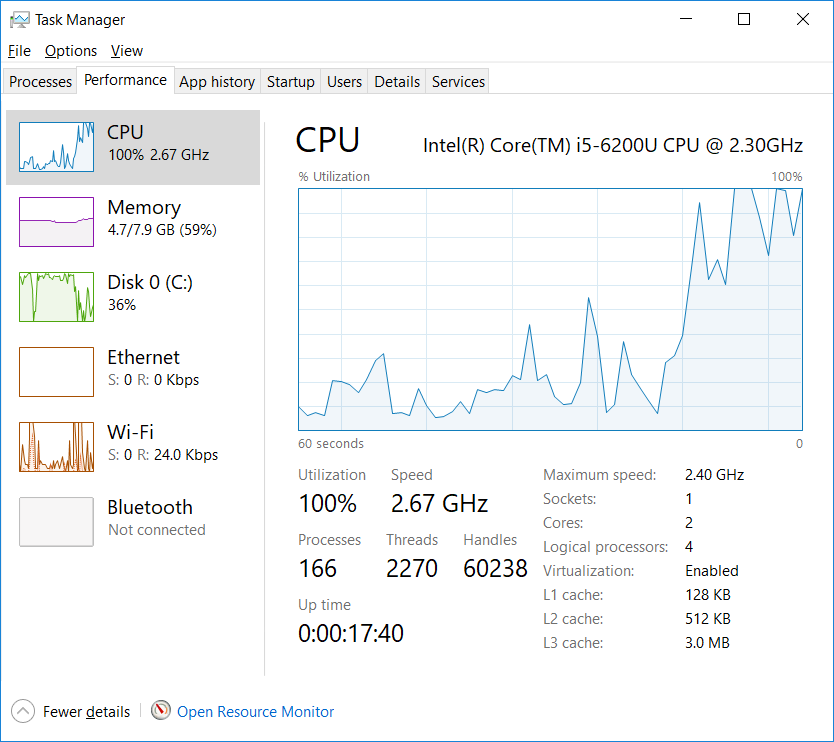
Total time for server to receive 2,000,000 events (10,000 each connection): 28.985 seconds

CPU and memory status:



Total time for server to receive 4,000,000 events (20,000 each connection): Server-side failed and never finished receiving all the events.

CPU and memory status:



**Conclusion:**

Time performance – How much time a socket took to finish receiving all events.

Load performance – The maximum number of events a socket able to receive.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Single connection performance (Client to server) | 100 connections time performance (Client) | 100 connections load performance (Client) | 200 connections time performance (Client) | 200 connections load performance (Client) | Maximum allowed connections |
| Php Socket.IO | Lose  (220,000 events: 17.621 seconds) | Lose  (5,000 events: 2.67 seconds,  8,000 events:  7.6858 seconds) | Lose | Lose  (5,000 events: 2.304 seconds) | Lose | 255 |
| Node.js Socket.io | Win  (220,000 events:  5.4298 seconds) | Win  (10,000 events: 1.244 seconds,  50,000 events: 8.0878 seconds,  100,000 events: 20.375 seconds) | Win | Win  (10,000 events: 0.45 seconds,  50,000 events: 5.2239 seconds,  100,000 events: 15.4798 seconds) | Win | 263 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Single connection performance (Server to client) | 100 connections time performance (Server) | 100 connections load performance (Server) | 200 connections time performance (Server) | 200 connections load performance (Server) |
| Php Socket.IO | Lose  (10,000 events: 2.0194 seconds) | Lose  (100,000 events: 4.04 seconds) | Lose | Lose  (200,000 events: 28.287 seconds) | Lose |
| Node.js Socket.io | Win  (220,000 events: 5.6778 seconds) | Win  (1,000,000 events: 21.792 seconds,  2,000,000 events: 62.966 seconds,  3,000,000 events: 79.015 seconds) | Win | Win  (2,000,000 events: 28.985 seconds) | Win |

Socket.IO on Node.js has out performed PHP Socket.IO in all aspects, in terms of speed performance, load performance and maximum connections supported. I believe one of the main factor is that Javascript has better performance than PHP in general, so it would be reasonable to choose Socket.IO over phpSocket.IO.