

COMP5311

Internet Infrastructure and Protocols



2010-2011 Group Project

SPDY: An Experimental Protocol for a Faster Web

Name	Email	Student no
Lui Yiu Sum - Ricky		
Ho Kai Man		
Po Yu Paulman CHAN		

Team 14

Date: 20 Dec 2010

Lecturer: Dr. Rocky K. C. Chang

Project tutor: Daniel Luo

Table of Contents

1.	Introduction	3
2.	Objective.....	3
3.	Team Background	4
4.	Overview SPDY	4
5.	Analysis [analysis the implementation on JAVA/Python]	7
	5.1 Compressed header	7
	5.2 Prioritized requests.....	8
	5.3 Multiplexed Requests.....	9
6.	Problems [Implementation SDPY Client/Server].....	9
	6.1. Java implementation of SPDY client/server	9
	6.2. Python implementation	11
7.	Academic result [Measurement Data].....	15
8.	Conclusion	21
9.	Advance.....	21
10.	Reference	22

1. Introduction

Now a day, the most common medium for exchange information through the internet are web (http). Consider the data size rapidly increased in these 5 to 10 years. From earlier simple text, small photo (gif, jpeg) until now rich text, web application (php, jsp.etc..) graphic(high resolution), flash, anime, video(YouTube). Data increased from hundred to thousand and million of bytes and becomes more on the future. We know that hardware resource no big exchange/enhanced. Thus, efficient on transporting data through the network are become more importance and concerned.

The web browsers (IE, Firefox, Google) we are currently use basically using http application-layer protocol for transporting contents over the web. We may need to think about the current http protocol used to transport information are still good enough? Can it's fulfill web evolution and rapidly growth? Http may not be the best protocol but it can adopt different kinds of web services, so it still in widely use today. "SPDY" a new application - layer protocol define by Google. The purpose of "SPDY" is not to replace "HTTP", it just to provide an alternative ways for improving the data exchange through the network in better performance in process and time.

2. Objective

From the SPDY official pages[<http://www.chromium.org/spdy>], it mentioned the performance compare with using SPDY and HTTP Protocol (have observed up to 64% reductions in page load times in SPDY compare with HTTP.). This report is trying to verify the performance of SPDY. By analysis how it works (how it implementation on programming language Python and Java) and test it (build a custom SPDY server to test the performance), collect academic result from measuring the load times using SPDY and HTTP. Finally make conclusion from the finding data.

3. Team Background

SPDY is in an experiment phase, the code from Chromium may not up to date. We cannot know the current code support which version of SPDY protocol and we have to compile the code by our own before to do the measurement. Chromium provides several code implementations for experience. We chose the java implementation for measurement as java is very popular, but after testing for two week we found some problem of this implementation (mention in 6. Problem [Implementation]). We then change to test the python implementation. Due to we were running out of time, we can only test the SPDY server and SPDY client but not the SPDY proxy.

We may not provide much of academic result and correct assumption in our answer. Consider with our members different background, familiar in programming language, timing and workloads. We try to work hard and present what we have done from the project. Although we have quite a lot of difficulty during the project, we are enjoying exploring on this new technology.

For the first two weeks Man and Ricky study the program implementation and summarize founding. Then, Man focus on the code and Ricky doing the report. For last two week, we integrated our work and complete the remaining report.

Our teams have three members Ricky, Man and Paulman, but one of our team members – Paulman contributes nothing to this project. Paulman have been absent or left early for several lesson after the group project released. We cannot contact him to discuss the project and he did not reply our email. He only appears on the date of presentation and explains that he was getting sick and hopes that we can help him for the project. Due to we cannot know he is telling the true or not and we know Paulman is taking THREE subject selected in this seminar. We decided to report the problem in this report.

4. Overview SPDY

On SPDY, it mentioned several points of HTTP inhibit optimal performance as below:

- Single request per connection.
- Exclusively client-initiated requests.
- Uncompressed request and response headers.
- Redundant headers.
- Optional data compression. HTTP uses optional compression encodings for data. Content should always be sent in a compressed format.

(Reference: <http://dev.chromium.org/spdy/spdy-whitepaper>)

I think we did not talk too much about the detail of the points, it because we can easily access the SPDY project webpage to get the detail information

Flow control, we have summarized between TCP and SPDY

TCP	SPDY
<p>TCP uses an end-to-end flow control protocol to avoid having the sender send data too fast for the TCP receiver to receive and process it reliably. TCP uses a sliding window flow control protocol. In each TCP segment, the receiver specifies in the receive window field the amount of additional received data (in bytes) that it is willing to buffer for the connection. The sending host can send only up to that amount of data before it must wait for an acknowledgment and window update from the receiving host.</p> <p>When a receiver advertises a window size of 0, the sender stops sending data and starts the persist timer. The persist timer is used to protect TCP from a deadlock situation that could arise if a subsequent window size update from the receiver is lost, and the sender cannot send more data until receiving a new window size update from the receiver. When the persist timer expires, the TCP sender attempts recovery by sending a small packet so that the receiver responds by sending another acknowledgement containing the new window size.</p> <p>If a receiver is processing incoming data in small increments, it may repeatedly advertise a small receive window. This is referred to as the silly window syndrome, since it is inefficient to send only a few bytes of data in a TCP segment, given the relatively large overhead of the TCP header. TCP senders and receivers typically employ flow control logic to specifically avoid repeatedly sending small segments. The sender-side silly window syndrome avoidance logic is referred to as Nagle's algorithm.</p> <p>Reference: http://en.wikipedia.org/wiki/Transmission_Control_Pr</p>	<p>Each side can announce how much data or bandwidth it will accept for each class of streams. If this is done, then speculative operations such as server push can soak up a limited amount of the pipe (especially important if the pipe is long and thin). This may allow for the elimination of more complex "is this already in the cache" or "announce what I have in my cache" schemes which are likely costly and complex.</p> <p>The WINDOW_UPDATE control frame is used to implement per stream flow control in SPDY. Flow control in SPDY is a per hop, that is, only between the two endpoints of a SPDY connection. If there are one or more intermediaries between the client and the origin server, flow control signals are not explicitly forwarded by the intermediaries. (However, throttling of data transfer by any receiver may have the effect of indirectly propagating flow control information upstream back to the original sender.)</p> <p>Flow control only applies to the data portion of data frames. Receivers must buffer all control frames. If a receiver fails to buffer an entire control frame, it must send a RST_STREAM with FLOW_CONTROL_ERROR to terminate the stream.</p> <p>Flow control in SPDY is implemented by a data transfer window kept by the sender of each stream. The data transfer window is a simple int32 that indicates how many byte of data the sender can transmit. After a stream is created but before any data frame is transmitted, both the sender starts with the initial window size. This initial window size is a measure of the buffering capability of the receiver. The sender must not send a data frame with data length greater than the transfer window</p>

otocol#Flow_control	<p>size. After sending each data frame, the sender decrements its transfer window size by the amount of data transmitted. When the window size becomes less than or equal to 0, the sender must pause transmitting data frame. At the other end of the stream, the receiver sends a WINDOW_UPDATE control back to notify the sender that it has consumed some data and freed up buffer space to receive more data.</p> <p>Reference: http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft2#TOC-Streams</p>
---------------------	---

We focus on the SPDY mentioned offers basic improvements over HTTP

- Multiplexed requests. There is no limit to the number of requests that can be issued concurrently over a single SPDY connection. Because requests are interleaved on a single channel, the protocol is more efficient over TCP.
- Prioritized requests. Clients can request certain resources to be delivered first. This avoids the problem of congesting the network channel with non-critical resources when a high-priority request is pending.
- Compressed headers. Clients today send a significant amount of redundant data in the form of HTTP headers. Because a single web page may require 50 or 100 subrequests, this data is significant. Compressing the headers saves a significant amount of latency and bandwidth compared to HTTP.
- Server pushed streams. This enables content to be pushed from servers to clients without a request.

[<http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft2>].

From the documentation provided on “spdy-protocol-draft2” we know that how does it work in logically. We should in dept analysis how does it implemented in real (coding). From the SPDY Project pages, it releases External work (experimental) on Python implementation of a SPDY server and Java implementations of SPDY client/server. It gives us an opportunity to see how it implemented in the above programming language in later section (some of basic improvement point).

5. Analysis [analysis the implementation on JAVA/Python]

5.1 Compressed header

SPDY explored that HTTP header which have amount of redundant data. Single web page may require 50 or 100 sub requests, this data is significant. Compressing the headers saves a significant amount of latency and bandwidth compared to HTTP. We found that Java and Python both using “zlib” library for the header compression.

The Python implemented SPDY project is named “nbhttp”

[Python] nbhttp : spdy_common.py

Line97:

```
    if compressed_hdrs:
        self._compress = c_zlib.Compressor(-1, dictionary)
        self._decompress = c_zlib.Decompressor(dictionary)
    else:
        self._compress = dummy
        self._decompress = dummy
```

[Java] SpdyConnection.java

Line 498:

```
    if (headerCompression) {
        headerCompressBuffer.recycle();
        headCompressOut.compress(headers, headerCompressBuffer, false);
        headerCompressBuffer.copyAll(headBuf);
        headerCompressBuffer.recycle();
    }
```

[Java] CompressFilter.java

Line 61:

```
    cStream.deflateInit(JZlib.Z_BEST_SPEED, 10);
```

They both using zlib (JZlib, java version zlib) for header compression. zlib supported algorithm same as zip which is widely used in data compression. [<http://www.zlib.net/>]

[<http://en.wikipedia.org/wiki/Zlib>]

All HTTP headers can be compressed; TCP header stored the sender and destination information

required for communicates with client/server. The compressed HTTP headers will decompress on the application layer of the receiver. On the transportation of packet, HTTP header information not required. We need to consider compress and decompress headers need extra works for calculation.

5.2 Prioritized requests

SPDY defined SYN_STREAM a control frames. In SYN_STREAM frame structure the priority field handle prioritized requests.

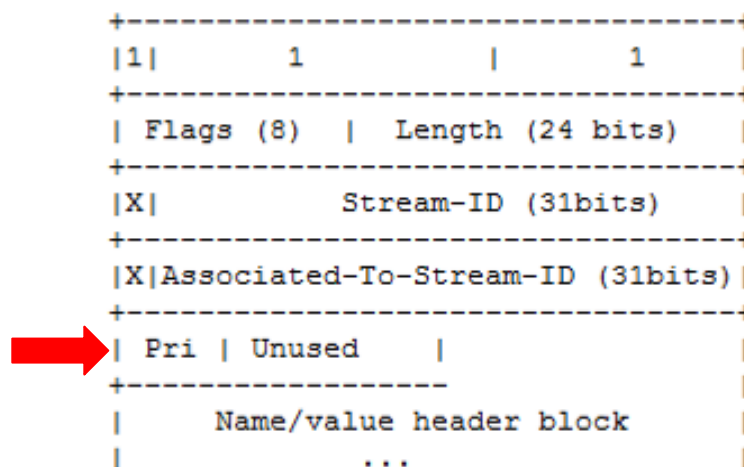


Figure. SYN_STREAMA Frame structure

Priority: A 2-bit priority field. If an endpoint has initiated multiple streams, the priority field represents which streams should be given first precedence. Servers are not required to strictly enforce the priority field, although best-effort is assumed. 0 represents the lowest priority and 3 represents the highest priority. The highest-priority data is that which is most desired by the client.

Seem both Java and Python not implemented the priority function yet, SPDY server in Java and Python do regular transmission without considering if the Priority set to 0 values. Both Java and Python set the Priority field in 0 values.

Good prioritization management is difficult to design. Some webpage more concern in graphic, some are not. What resources are critical to be delivered first and what are non-critical. I think server need to take the most response. Server should provide general recourses priority configuration (focus on all webpage) on text, graphic (gif, bmp, jpeg), multimedia (sound, video) to identify critical resources. Also specific recourses priority configuration (focus on individual webpage) is a must. Like root and user relationship.

5.3 Multiplexed Requests

We have not analysis on this field from the Java and Python due to time limitation. We need to think about if multiplexed requests allowed. There is no limit to the number of requests that can be issued concurrently over a single SPDY connection. The number of request can be numerous that maybe cause flooding. How to control a reasonable request limitation is are must.

6. Problems [Implementation SDPY Client/Server]

We use the java implementation of a SPDY client/server to analysis and do the measurement. But we found that the java implementation code is not working, we then change to test the Python implementation.

6.1. Java implementation of SPDY client/server

After downloaded all the source code, we run the test case to ensure the code is completed and work. These test cases are provided with SPDY with the java source code. The following image show the test report, most of the test case can be run successfully. Such as HTTPS, SPDY, Live HTTP and Proxy etc.

```

<?xml version="1.0" encoding="UTF-8"?>
- <testrun name="src: project: testspdy4" tasks="133" started="0" failures="0" errors="2" ignored="0">
+ <testsuite name="org.apache.tomcat.lite.io.CBufferTest" time="0.0">
+ <testsuite name="org.apache.tomcat.lite.http.DispatcherTest" time="0.0">
- <testsuite name="org.apache.tomcat.lite.http.HttpsTest" time="0.469">
+ <testcase name="testSimpleClient" classname="org.apache.tomcat.lite.http.HttpsTest" time="0.36" />
+ <testcase name="testSimpleServer" classname="org.apache.tomcat.lite.http.HttpsTest" time="0.015" />
+ <testcase name="testSimpleClient20" classname="org.apache.tomcat.lite.http.HttpsTest" time="0.094" />
+ <testcase name="testSimpleRequestGoogle" classname="org.apache.tomcat.lite.http.HttpsTest" time="0.0" />
</testsuite>
+ <testsuite name="org.apache.tomcat.test.watchdog.WatchdogTestCase" time="0.0">
+ <testsuite name="org.apache.tomcat.lite.http.HttpChannelInMemoryTest" time="0.016">
+ <testsuite name="org.apache.tomcat.lite.util.UEncoderTest" time="0.094">
+ <testcase name="ServletTests" classname="ServletTests" incomplete="true" />
+ <testsuite name="org.apache.tomcat.lite.io.OneTest" time="0.0">
+ <testsuite name="org.apache.tomcat.lite.io.BBufferTest" time="0.0">
+ <testsuite name="org.apache.tomcat.lite.io.UEncoderTest" time="0.0">
- <testsuite name="org.apache.tomcat.lite.http.SpdyTest" time="0.094">
+ <testcase name="testClient" classname="org.apache.tomcat.lite.http.SpdyTest" time="0.016" />
+ <testcase name="testServer" classname="org.apache.tomcat.lite.http.SpdyTest" time="0.016" />
</testsuite>
+ <testsuite name="org.apache.tomcat.lite.http.LiveHttpTest" incomplete="true">
+ <testsuite name="org.apache.tomcat.lite.http.HttpChannelTest" time="0.0">
+ <testsuite name="org.apache.tomcat.lite.load.MicroTest" time="0.0">
+ <testsuite name="org.apache.tomcat.lite.proxy.ProxyTest" time="2.0">
+ <testcase name="testSingleRequest" classname="org.apache.tomcat.lite.proxy.ProxyTest" time="0.016" />
+ <testcase name="test2Requests" classname="org.apache.tomcat.lite.proxy.ProxyTest" time="0.062" />
+ <testcase name="testRequestSimple" classname="org.apache.tomcat.lite.proxy.ProxyTest" time="0.031" />
+ <testcase name="testExtAdapter" classname="org.apache.tomcat.lite.proxy.ProxyTest" time="1.044" />
+ <testcase name="testStaticAdapter" classname="org.apache.tomcat.lite.proxy.ProxyTest" time="0.0" />
+ <testcase name="testRequestParams" classname="org.apache.tomcat.lite.proxy.ProxyTest" time="0.016" />
+ <testcase name="testRequestChunked" classname="org.apache.tomcat.lite.proxy.ProxyTest" time="0.015" />
+ <testcase name="testRequestSlow" classname="org.apache.tomcat.lite.proxy.ProxyTest" time="0.016" />
</testsuite>
+ <testsuite name="org.apache.tomcat.lite.load.LiveHttpThreadedTest" time="0.0">
+ <testsuite name="org.apache.tomcat.lite.proxy.SmallProxyTest" time="0.0">
+ <testcase name="testProxy" classname="org.apache.tomcat.lite.proxy.SmallProxyTest" time="0.0" />
</testsuite>
- <testsuite name="org.apache.tomcat.lite.io.SocksTest" time="0.016">
+ <testcase name="testSocks" classname="org.apache.tomcat.lite.io.SocksTest" time="0.016" />
</testsuite>
- <testsuite name="TestSuite with 40 tests [example: org.apache.tomcat.lite.http.LiveHttpTest]">
- <testsuite name="org.apache.tomcat.lite.http.LiveHttpTest" incomplete="true">
+ <testcase name="testSimpleRequest" classname="org.apache.tomcat.lite.http.LiveHttpTest" incomplete="true" />

```

Figure 1 Testing report – These test cases is provided with the SPDY source code.

After testing the SPDY source code, we setup two pc. One will be the Proxy server and the other will be the Client. Firstly, we use the client to request a webpage from hk.yahoo.com through the proxy server to ensure all the setting is working. Secondly, we will try to enable SPDY prototype. Finally, due to the second test case fail, we write the third test case base on the founding of second test. During the testing, we also use Wireshark to record the network transaction.

When the Test Case 2 failed, we check the record from Wireshark. We found that the server sends the data to the client when received the client request, but not all data were sent completely. So, we wrote the Test Case 3. This test case will just return a String and the size is less then 8KB.

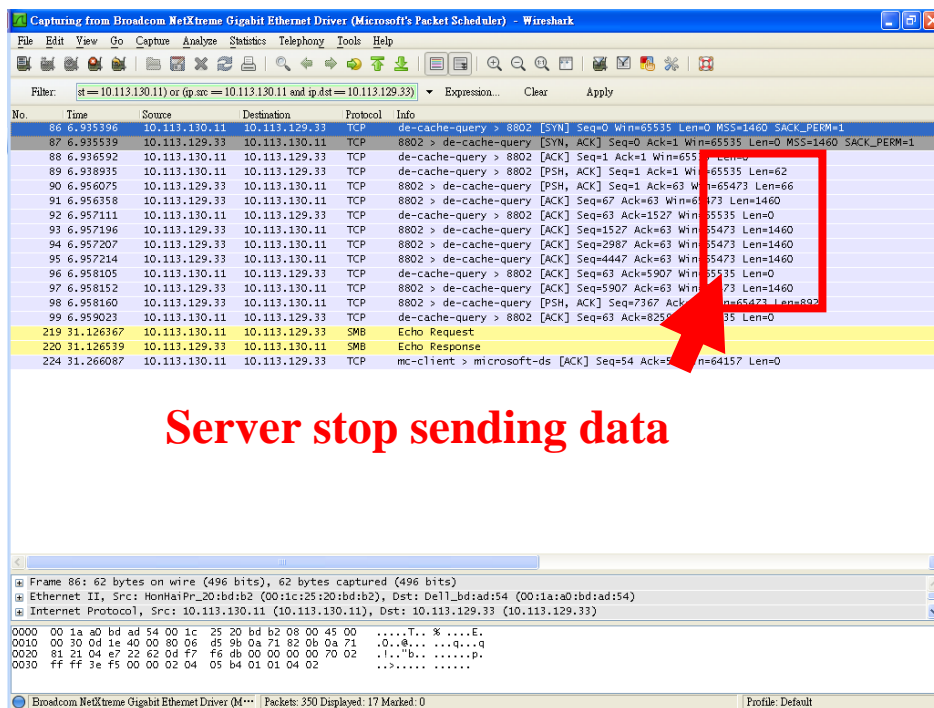


Figure 2 Test Case 2 – Wireshark record

Finally we summarized what the Java SPDY Server/client have done.

Testing result

Test Case No	Test Case	SPDY	Result
1	Client(HTTP) -> Proxy Server(HTTP) -> hk.yahoo.com	Disable	Success
2	Client(SPDY) -> Proxy Server(SPDY) -> hk.yahoo.com	Enable	Fail
3	Client(SPDY) -> Proxy Server(SPDY) -> return a string less then 8KB	Enable	Success

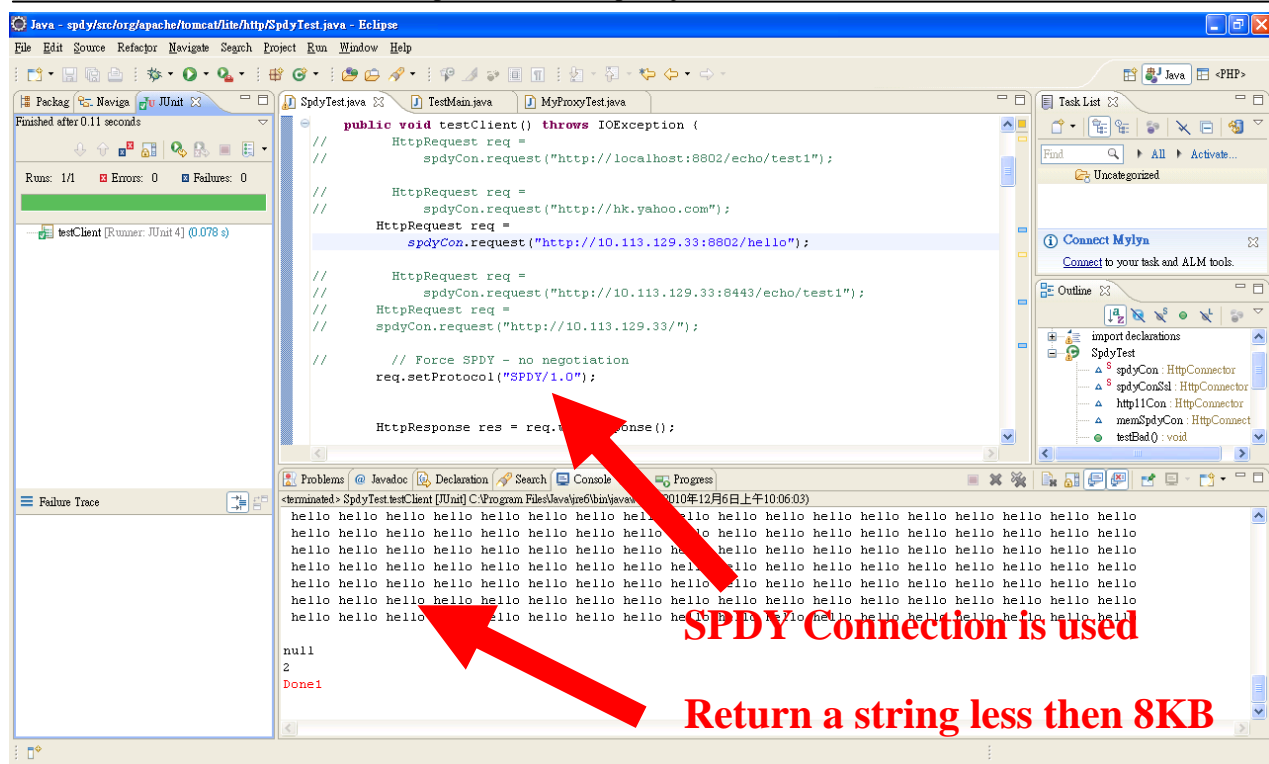


Figure 3 Test case 3- enable SPDY, but only work by return a string less then 8KB

The response time slightly more then not enable SPDY. As the response file size is too small and only one test case can be use, we decided change to test the Python implementation.

6.2. Python implementation

Due to we spent a lot of time on testing java implementation, we do not have enough time to setup the python proxy server. In the python testing, we can only setup the SPDY server and return a text file around 2100 KB to the SPDY client.

Testing result

Test Case No	Test Case	SPDY	Result	Time completion
1	Client(HTTP) -> Server(HTTP) -> return a text file around 2100KB	Disable	Success	15.64s
2	Client(SPDY) -> SPDY_Server(SPDY) -> return a text file around 2100KB	Enable	Success	22.891s

Even through the return file were increased, the result is same as the java implement. SPDY protocol is required more time then HTTP. We are not sure Python SPDY implementation are well-development because on this result.

```

C:\python_spdy\scripts>python server.py 1556

push_tcp.run()
File "C:\python_spdy\scripts\push_tcp.py", line 417, in run
    asyncore.poll(self.timeout)
File "C:\Python27\lib\asyncore.py", line 140, in poll
    r, w, e = select.select(r, w, e, timeout)
KeyboardInterrupt

C:\python_spdy\scripts>python spdy_server.py 1555
INFO:server:PID: 2748

Traceback (most recent call last):
  File "spdy_server.py", line 247, in <module>
    push_tcp.run()
  File "C:\python_spdy\scripts\push_tcp.py", line 417, in run
    asyncore.poll(self.timeout)
  File "C:\Python27\lib\asyncore.py", line 140, in poll
    r, w, e = select.select(r, w, e, timeout)
KeyboardInterrupt

```

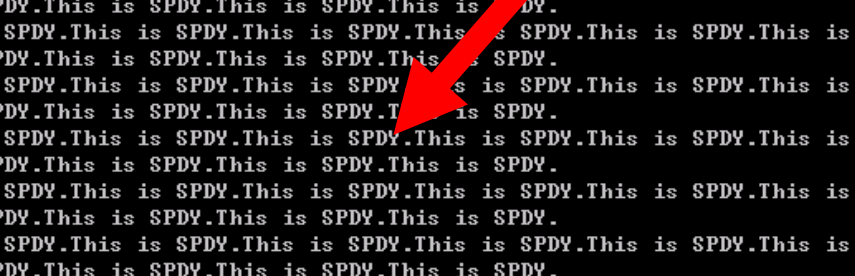
Scripts to run Server

```

C:\python_spdy\scripts>
C:\python_spdy\scripts>
C:\python_spdy\scripts>
C:\python_spdy\scripts>python server.py 1556
PID: 3740

```

Figure 4 Start up python server (HTTP)



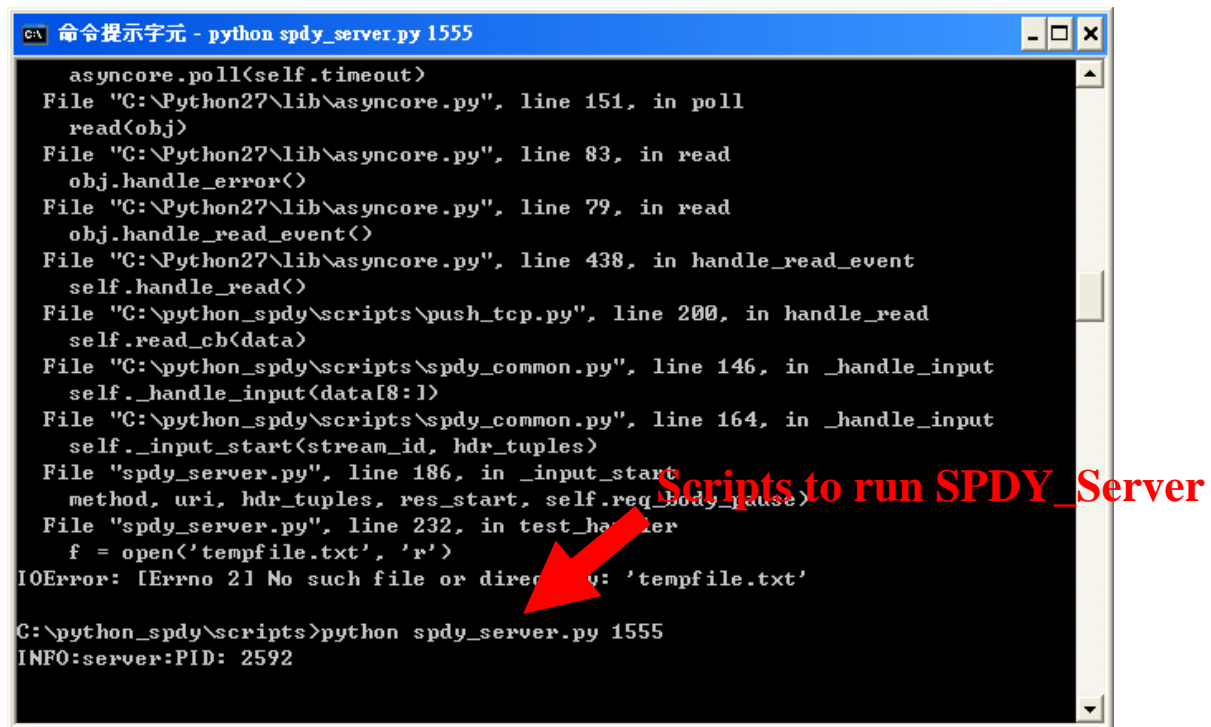
```
C:\> 命令提示字元

is is SPDY.This is SPDY.This is SPDY.This is SPDY.
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.

Time:
0:00:15.640000

C:\python_spdy_Server\scripts>
```

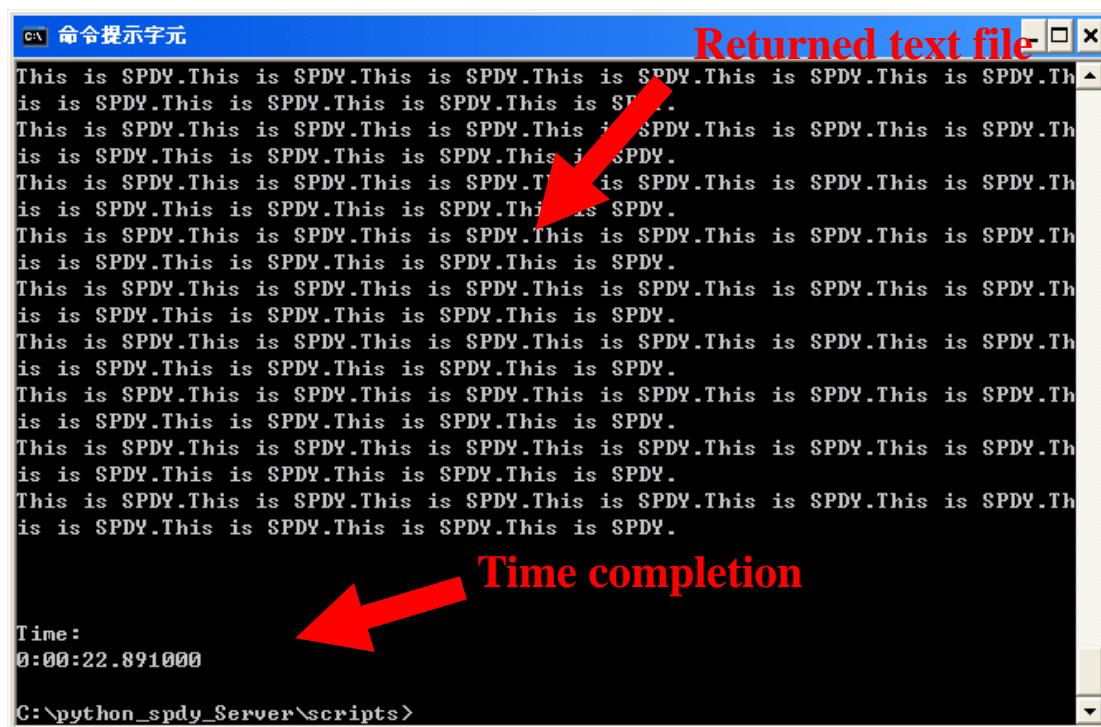
Figure 5 Run the client(HTTP) and call the server(HTTP)



```
命令提示符 - python spdy_server.py 1555
asyncore.poll(self.timeout)
File "C:\Python27\lib\asyncore.py", line 151, in poll
  read(obj)
File "C:\Python27\lib\asyncore.py", line 83, in read
  obj.handle_error()
File "C:\Python27\lib\asyncore.py", line 79, in read
  obj.handle_read_event()
File "C:\Python27\lib\asyncore.py", line 438, in handle_read_event
  self.handle_read()
File "C:\python_spdy\scripts\push_tcp.py", line 200, in handle_read
  self.read_cb(data)
File "C:\python_spdy\scripts\spdy_common.py", line 146, in _handle_input
  self._handle_input(data[8:])
File "C:\python_spdy\scripts\spdy_common.py", line 164, in _handle_input
  self._input_start(stream_id, hdr_tuples)
File "spdy_server.py", line 186, in _input_start
  method, uri, hdr_tuples, res_start, self.req_body_pause)
File "spdy_server.py", line 232, in test_handler
  f = open('tempfile.txt', 'r')
IOError: [Errno 2] No such file or directory: 'tempfile.txt'

C:\python_spdy\scripts>python spdy_server.py 1555
INFO:server:PID: 2592
```

Figure 6 Start up python (SPDY) server



```
命令提示符
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.
This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.This is SPDY.Th
is is SPDY.This is SPDY.This is SPDY.This is SPDY.
Time:
0:00:22.891000

C:\python_spdy_Server\scripts>
```

Figure 7 Run the (SPDY) client and call the (SPDY) server

Wireshark record (return a mall file for easy comparison).

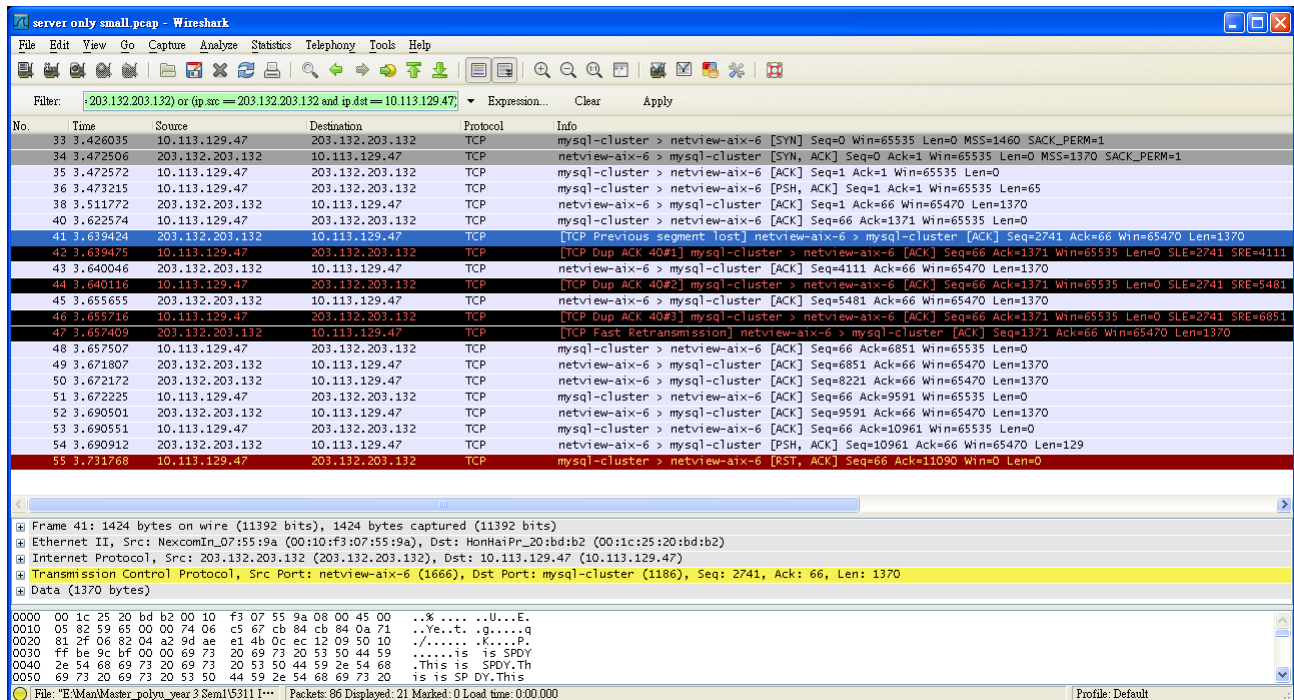


Figure 8 captured in HTTP Client/Server

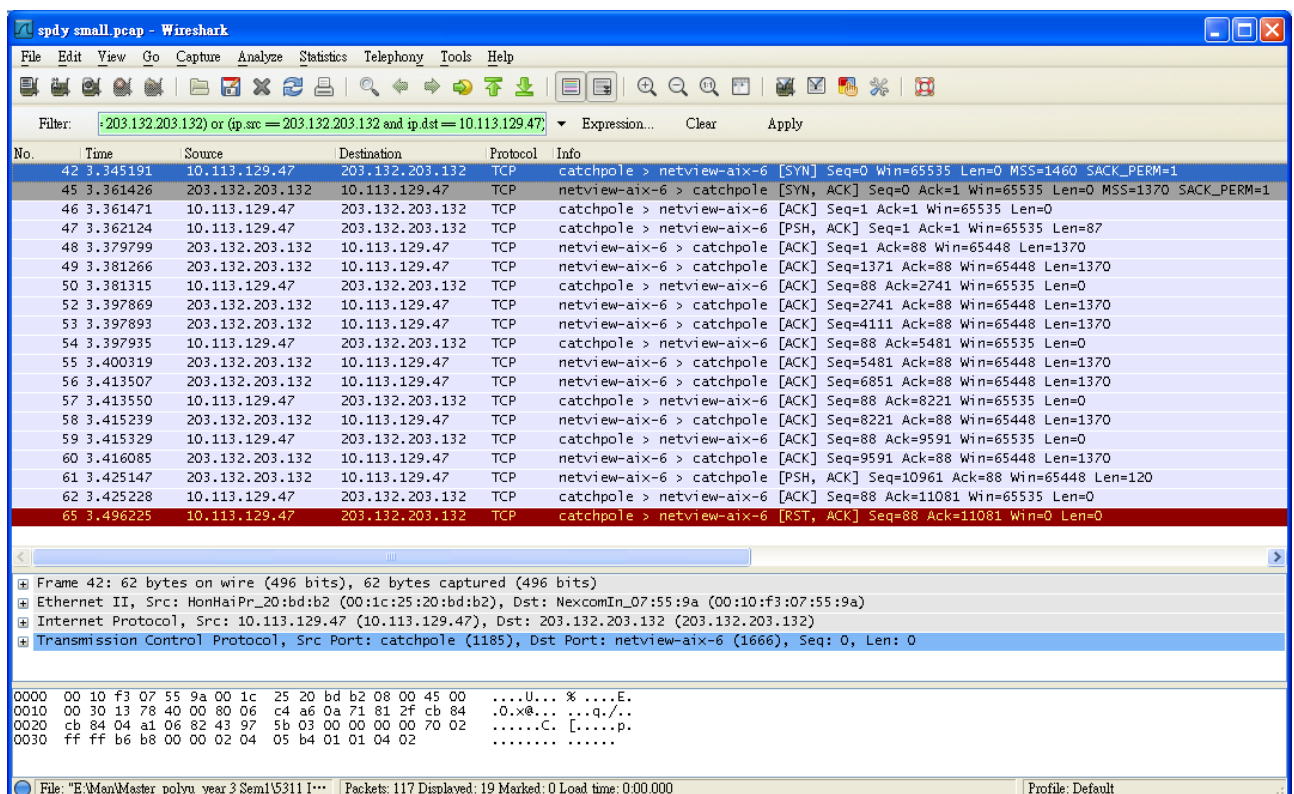


Figure 9 captured in SPDY Client/Server

SPDY make the web faster, but on the other side, SPDY did take other cost to make the web faster.

Every time a packet is send, the header is compress. The server and client have to spent time to compress

and decompress, which cause more CPU, memory usage and additional delay. The new framing control and prioritized request is also not free. SPDY is changing the cost to other from network side.

7. Academic result [Measurement Data]

From the previous section “Problem [Implementation SPDY client/Server]”, although we can run the Python SPDY client and server. Data are not strong enough to support as SPDY project say. On Chromium’s Google browser, it provided a benchmark plug-in and supported using SPDY. [http://www.chromium.org/chrome-benchmarking-extension]

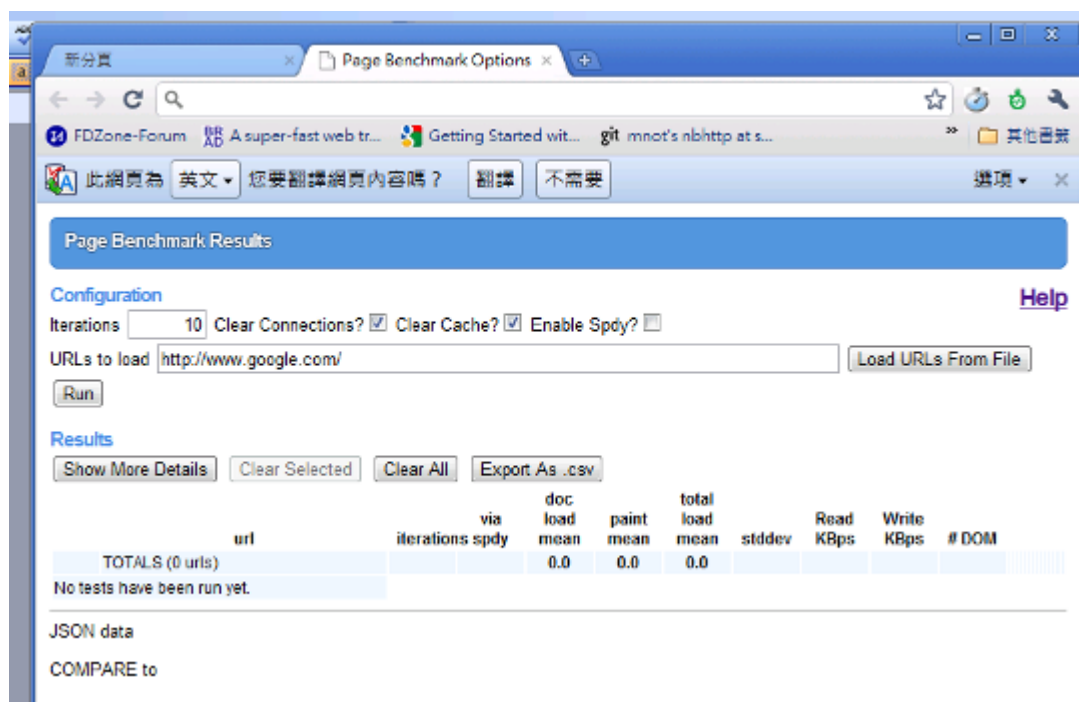


Figure 7.1. Google benchmark plug-in pages

At early of November, we have tried “Enable Spdy” for testing different webpage, it seem that all the web server not support SPDY yet including www.google.com.hk. Until end of November, we found that www.google.com.hk supported using SPDY. (“Enable Spdy” options is direct use SPDY for communicate to the destination web server). We can use the benchmark plug-in testing SPDY connection with www.google.com.hk. Only www.google.com.hk supported SPDY. So we can only test with google.

See the result below:

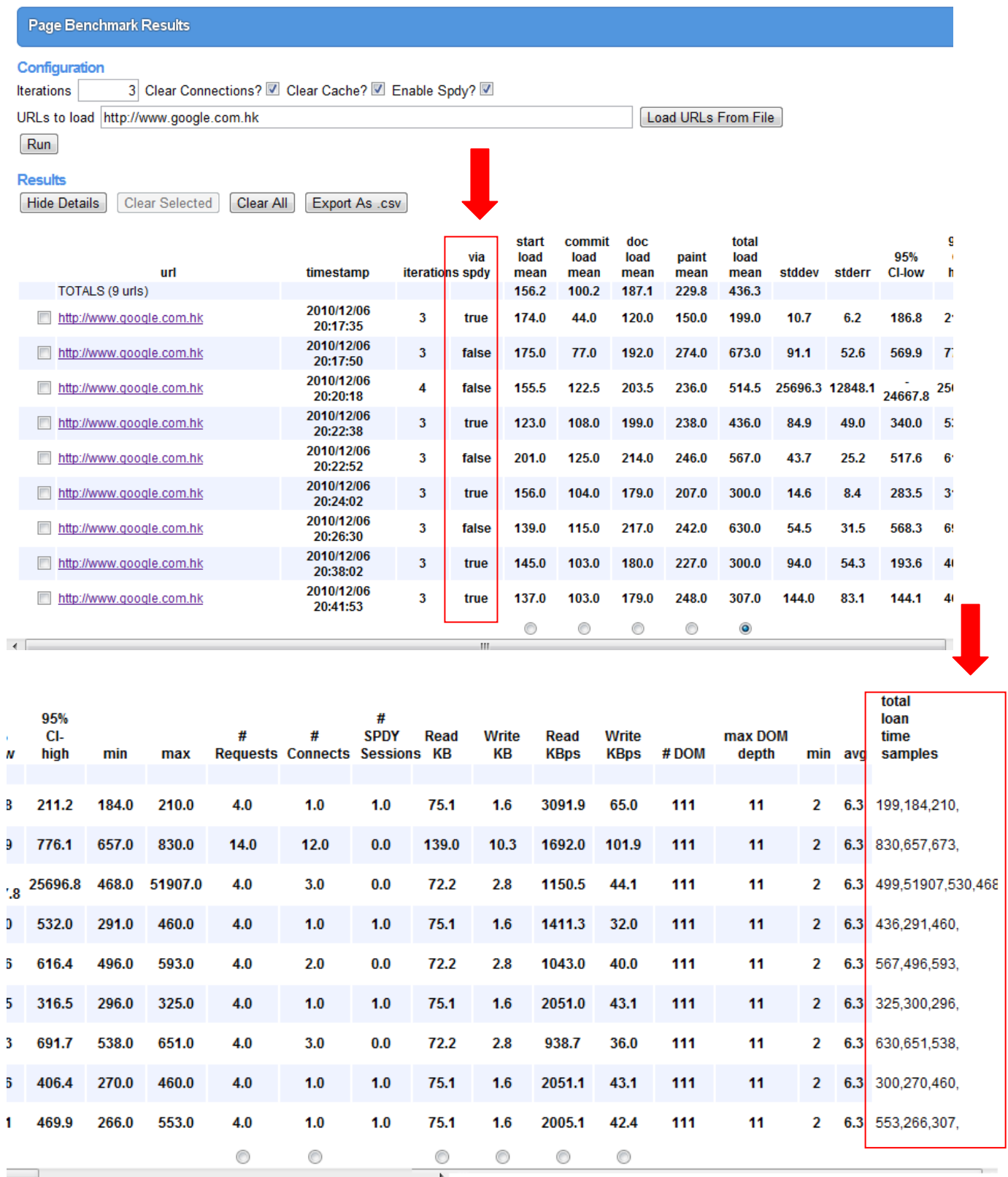


Figure 7.2. Google benchmark result on www.google.com.hk via SPDY/HTTP

Also we try to test a www.google.com.hk with searching requirement and show in picture to increase the page loading

v	95% CI- high	min	max	# Requests	# Connects	# SPDY Sessions	Read KB	Write KB	Read KBps	Write KBps	# DOM	max DOM depth	min	avg	total loan time samples
.2	3500.8	432.0	7787.0	3.0	2.7	2.7	256.8	2.4	3193.0	25.9	1362	19	2	14.3	432,766,7787,964,452,
.3	4641.7	548.0	10475.0	3.0	2.0	0.0	249.7	2.4	2661.0	25.2	1362	19	2	14.3	675,548,600,10475,1325,
				○	○		○	○	○	○					

Figure 7.4 Google benchmark result on www.google.com.hk “Disney car” visa SPDY/HTTP

Page Benchmark Results

Configuration

Iterations Clear Connections? ☒ Clear Cache? ☒ Enable Spdy? ☐

URLs to load

Results

	url	timestamp	iterations	via spdy	start load mean	commit load mean	doc load mean	paint mean	total load mean	stddev	stderr	95% CI-low	95% CI-high	min
TOTALS (4 uris)														
<input type="checkbox"/>	http://www.google.com.hk/lim_80&bih=639	2010/12/19 20:32:10	4	true	48.5	98.0	296.0	356.0	356.0	367.9	184.0	4.6	716.6	294.0
<input type="checkbox"/>	http://www.google.com.hk/lim_80&bih=639	2010/12/19 20:32:19	4	false	46.5	96.0	487.5	275.5	546.5	27.6	13.8	519.5	573.5	498.0
<input type="checkbox"/>	http://www.google.com.hk/lim_80&bih=639	2010/12/19 20:32:43	4	true	45.0	87.0	307.0	356.5	356.5	13.8	6.9	343.0	370.0	344.0
<input type="checkbox"/>	http://www.google.com.hk/lim_80&bih=639	2010/12/19 20:32:50	4	false	46.0	99.5	485.0	291.5	517.0	69.1	34.6	449.3	584.7	504.0
<input type="button" value="Compare"/>														

max	# Requests	# Connects	# SPDY Sessions	Read KB	Write KB	Read KBps	Write KBps	# DOM	max DOM depth	min	avg	total loan time samples
1089.0	3.0	5.0	5.0	218.7	3.1	5037.7	72.0	1349	19	2	14.4	1089,370,294,342,
567.0	3.0	2.0	0.0	249.5	2.4	3741.3	35.5	1358	19	2	14.3	535,558,498,567,
375.0	3.0	1.0	1.0	217.9	1.6	5013.3	36.2	1349	19	2	14.4	368,345,375,344,
654.0	3.0	2.0	0.0	249.5	2.4	3954.2	37.5	1358	19	2	14.3	508,526,504,654,
	○	○		○	○	○	○					

Figure 7.5 different time result on Google benchmark on www.google.com.hk “Disney car” visa SPDY/HTTP

We have summarized the result on figure 7.2, 7.4, 7.5 as follow table and more on different time benchmark result

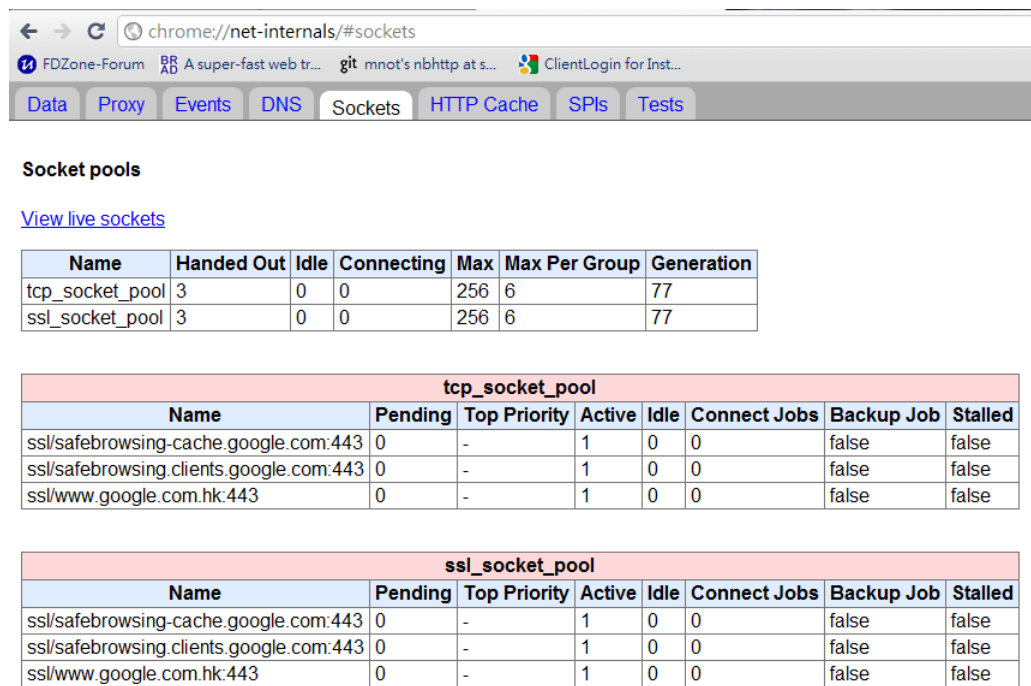
www.google.com.hk

Test Case	Iterations	via Spdy	min load time	max load time	Total load means	loan times on each iterations	Compare Group
1	3	Y	184	210	199	199, 184, 210	1
2	3	N	657	830	673	830, 657, 673	
3	4	N	468	51907	514.5	499, 51907, 530,468	2
4	3	Y	291	460	436	436, 291,460	
5	3	Y	496	593	567	567, 496, 593	
6	3	Y	296	325	300	325, 300, 296	3
7	3	N	538	651	630	630, 651, 538	
8	3	Y	270	460	300	300, 270, 460	4
9	3	Y	266	553	307	553, 266, 307	
10	5	N	299	284	235.7	235, 229, 240, 232, 284	5
11	5	Y	181	199	194.7	196, 195, 193, 181, 199	
12	5	N	172	238	224.3	172, 238, 226, 221, 226	
13	5	Y	129	306	167.3	174, 156, 129, 306, 172	
http://www.google.com.hk/images?hl=zh-tw&q=disney%20car&um=1&ie=UTF-8&source=og&sa=N&tab=wi&biw=1280&bih=639							
Test Case	Iterations	via Spdy	min load time	max load time	Total load means	loan times on each iterations	Compare Group
1	5	Y	432	7787	727.3	432, 766, 7787, 964, 452	1
2	5	N	548	10475	866.7	675, 548, 600, 10475, 1325	
3	4	Y	294	1089	356	1089, 370, 294, 342	2
4	4	N	498	567	546.5	535, 558, 498, 567	
5	4	Y	344	375	356.5	368, 345, 375, 344	
6	4	N	504	654	517	508, 526, 504, 654	

Table 7.6 Benchmark Result table

From the benchmark result, via SPDY have the better performance on each compare group (consider different time). But test set only focus on google SPDY supported server. We have no other web server like yahoo, Microsoft supported SPDY for comparison. We can only say that using SPDY in google server have better performance with HTTP.

The Chrome browser plug-in also shows the Socket information while SPDY is enabling. From the figure 10, we can see SPDY is running over SSL.



The screenshot shows the Chrome DevTools 'net-internals' page with the 'Sockets' tab selected. It displays two summary tables for 'tcp_socket_pool' and 'ssl_socket_pool'. Below each summary table is a detailed table showing the state of individual connections to Google services.

Socket pools

[View live sockets](#)

Name	Handed Out	Idle	Connecting	Max	Max Per Group	Generation
tcp_socket_pool	3	0	0	256	6	77
ssl_socket_pool	3	0	0	256	6	77

Name	Pending	Top Priority	Active	Idle	Connect Jobs	Backup Job	Stalled
ssl/safebrowsing-cache.google.com:443	0	-	1	0	0	false	false
ssl/safebrowsing.clients.google.com:443	0	-	1	0	0	false	false
ssl/www.google.com.hk:443	0	-	1	0	0	false	false

Name	Pending	Top Priority	Active	Idle	Connect Jobs	Backup Job	Stalled
ssl/safebrowsing-cache.google.com:443	0	-	1	0	0	false	false
ssl/safebrowsing.clients.google.com:443	0	-	1	0	0	false	false
ssl/www.google.com.hk:443	0	-	1	0	0	false	false

Figure 10 SPDY test over google – Socket information

8. Conclusion

In this project, we cannot see performance is better when SPDY is enabling. The reason may be the SPDY protocol Draft3 still not implement yet. The current implementation cannot utilize all the benefits proposed by the SPDY protocol Draft3.

There are several proposed for web latency, SPDY is not the only one. Some of these are trying to replace TCP, and some is runs on top of UDP. Since TCP is the most popular protocol. It has been used for many years, a lot of hardware device support well for TCP. It is sure that changing TCP or replace it will face a lot of difficulty. One of the goals of SPDY is to minimize deployment complexity by using the TCP as the underlying transport layer, so requires no changes to existing networking infrastructure. The goals can definitely make SPDY more reality.

Although in our experiment measurement can not prove the benefit of SPDY, we can see the SPDY group is making process. It is not an easy task to develop a new network protocol, beside the problem of the new protocol, there are too many other factor have to consider.

9. Advance

During the project study, we have more familiar with flow control, coding implementation and improved network knowledge, from design (documentation) to coding and implement.

New Protocol is not easy to design. It includes a lot of time, effort, technology and knowledge. It also faces to different physical and logical problems, defining structure, rules and algorithm. The flow control similar in our COMP5311 teaching material, the course is helping us easy to pickup this knowledge.

Both of us are interested in this project. This project topic is much related to us (Internet access). In real working environment, our job nature is seldom to keep in touch on new technology and knowledge. Study on this project is challenging. We found that we are weak in much of networking advance knowledge. COMP5311 course has helped us a lot.

For continuous study on this project, we build up a SPDY server and SPDY proxy server. We have improved our knowledge related in this project like Python programming language and networking concept. We also do testing and measurement in SPDY, but it would be better if we can measure and compare the number of packet flow of enable/disable SPDY, which can help us to more understand how SPDY can send few packets than HTTP.

10. Reference

<http://dev.chromium.org/spdy/spdy-whitepaper>

<http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft2#TOC-Streams>

<http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft2>

http://en.wikipedia.org/wiki/Transmission_Control_Protocol#Flow_control