TECHNICAL CHALLENGE

DOCUMENT TITLE

Technical Challenge Report: Answers & Analysis

Revision no.: 01

01	20-02-20	
Rev. no.	Issue Date	Ademola **MACGREGOR**
		Prepared by

CHANGE RECORD

Date	Author	Version	Change Note
2020-02-20	Ademola **MACGREGOR**	01	Document created

Table of content

1. SECTION ONE

1.1. REQUIREMENT CHECKLIST – SCRIPT OVERVIEW OF TECHNICAL REQUIREMENT

TASK BREAKDOWN	STATUS
Write a script/program that periodically calls rippled's server_info command and records the sequence number of the latest validated ledger along with the current time.	Completed
Record this data in a file.	Completed
Then, use this data to construct a plot (time on the x-axis, sequence number on the y-axis) that visualizes how frequently the ledger sequence is incremented over time (i.e. how often new ledgers are validated).	Completed
Choose a time span and polling interval that can effectively capture and depict this information.	Completed

1.2. REQUIREMENT CHECKLIST – QUESTIONS AND ANSWERS

TASK BREAKDOWN	STATUS
How does your script work?	Completed
How did you decide on your polling interval?	Completed
What do the results tell you?	Completed
What might explain the variation in time between new ledgers? (this description of the consensus algorithm may help you: https://developers.ripple.com/consensus-principles- and-rules.html)	Completed
There are some other (better) ways that you could use the rippled API to find how long each ledger took to close/validate. Using the API documentation, find and describe one of these methods (you don't need to actually implement it).	Completed

2. SECTION TWO

2.1. HOW DOES THE SCRIPT WORK

Below are some background information about the script:

Scripting Language: Shell
Platform/Environment: Amazon Linux AMI (EC2 Cloud Instance)
Tools & Dependencies:

curL, awk, cat, sed, cut etc..
gnuplot software
jq (json filter)
xauth
Postman (for testing API before scripting)

The step by step actions carried out when the script are detailed below:

i. Backup and deletion of result from previous API request from output directory.

ii. Declaration of all required input variables.

iii. Parameterized User Entry with input validation for
   a. Desired Polling interval in seconds (positive integer) for posting requests to http://s1.ripple.com:51234/.
   b. Desired Count of Polls (positive integer) for posting requests to http://s1.ripple.com:51234/.

iv. API Request to http://s1.ripple.com:51234/ and filtering of json response for desired data (result.info.time & result.info.validated_ledger.seq). The request is in a "for loop" with respect to values entered for the Polling interval and Polling Count. The resulting data is saved to a file.

   i. API Request Operation below:

```
#Ripple API Call
for ((i=1; i<=$count; i++))
do
        curl -s -H "Content-type: application/json" -d '{"method":
        "server_info","params": [{}]}' 'http://s1.ripple.com:51234/' | jq -r '[.result |
        .info.time, .info.validated_ledger.seq] | @tsv' >> $output
        sleep $interval
done
```

v. Graph is plotted with gnuplot with axis set to auto. All required formats for datetime are set with reference to the exported data.

i. Graph Parameters Below:

```
gnuplot -persist <<-EOFMarker
        # grid
        set grid
        #ranges
        set autoscale x
        set autoscale y
        #title and labels
        set title 'VALIDATED LEDGERS TIME - SEQ GRAPH'        # plot title
        set xlabel 'Time'                                     # x-axis label
        set ylabel 'SequenceNumber'                           # y-axis label
        #datetime formats
        set xdata time
        set timefmt "%Y-%b-%d %H:%M"
        set format x "%m/%d"
        set timefmt "%Y-%b-%d %H:%M:%S"
        plot "/tmp/xrpledger/result/output.csv" using 1:3 with linespoint
EOFMarker
```

vi. Average, Minimum and Maximum time differences between API calls are calculated from the response dataset after removal of records with duplicate sequence numbers and conversion of the timestamp to UNIX Epoch time. The result is then printed on the screen for convenience.

2.2. HOW DID YOU DECIDE ON YOUR POLLING INTERVAL

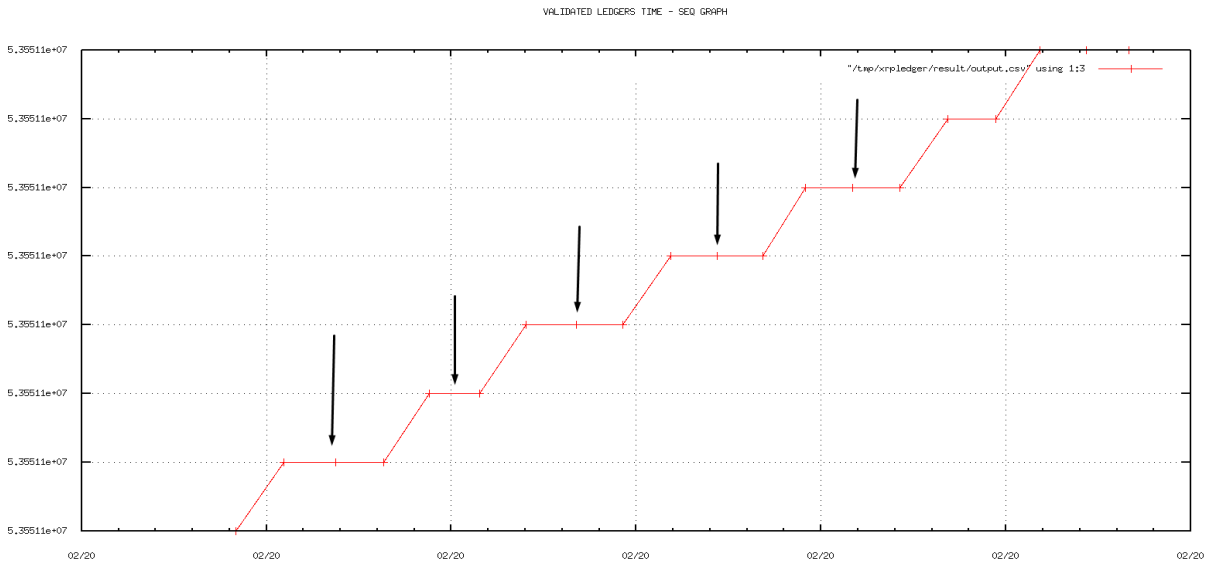
In order to determine the polling interval most suitable for the test, I carried out four rounds of tests with different polling intervals and made a comparison. The script allows for a user to input his desired polling interval. The test scenarios are below:

- Test 1: Polling Interval = 1 second; Polling Count: 20
- Test 2: Polling Interval = 2 seconds; Polling Count: 20
- Test 3: Polling Interval = 3 seconds; Polling Count: 20
- Test 4: Polling Interval = 4 seconds; Polling Count: 20

Find below my observations

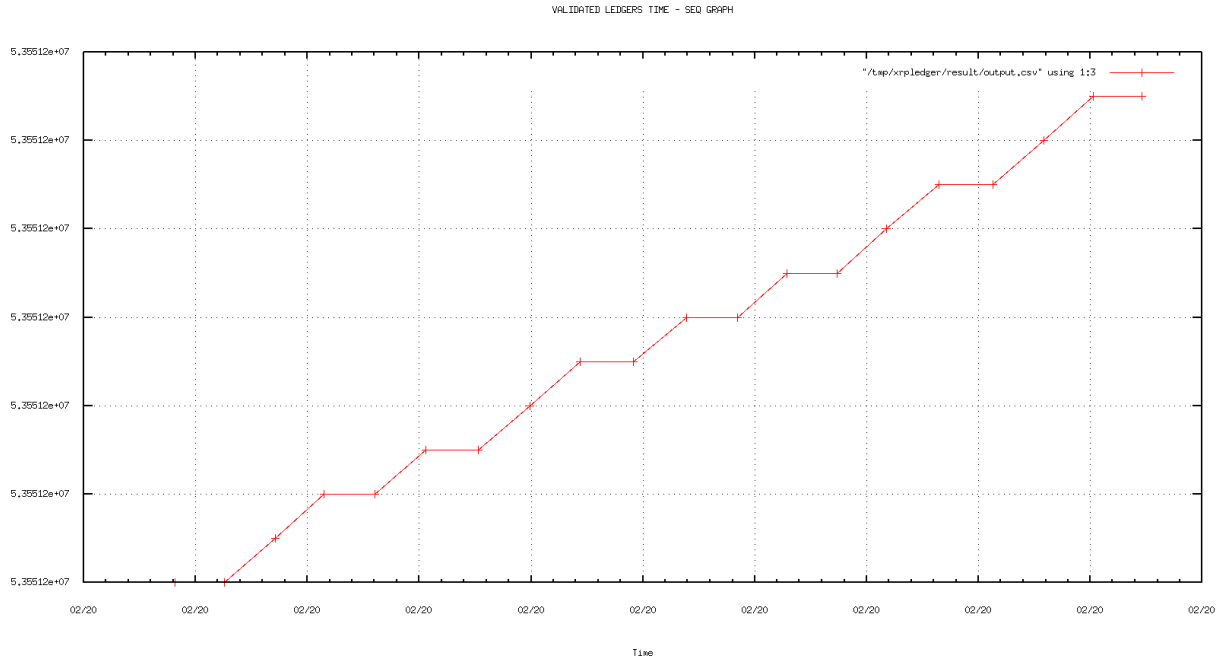
Test 1: Polling Interval = 1 second; Polling Count: 20

- The sequence number increases with respect to time in a linear progression.
- In all cases, each sequence number is repeated 2 to 3 times during the query cycle.
- The Average, Maximum and Minimum time taken for new ledger to be validated from the extracted data was 3, 4 and 1 respectively.



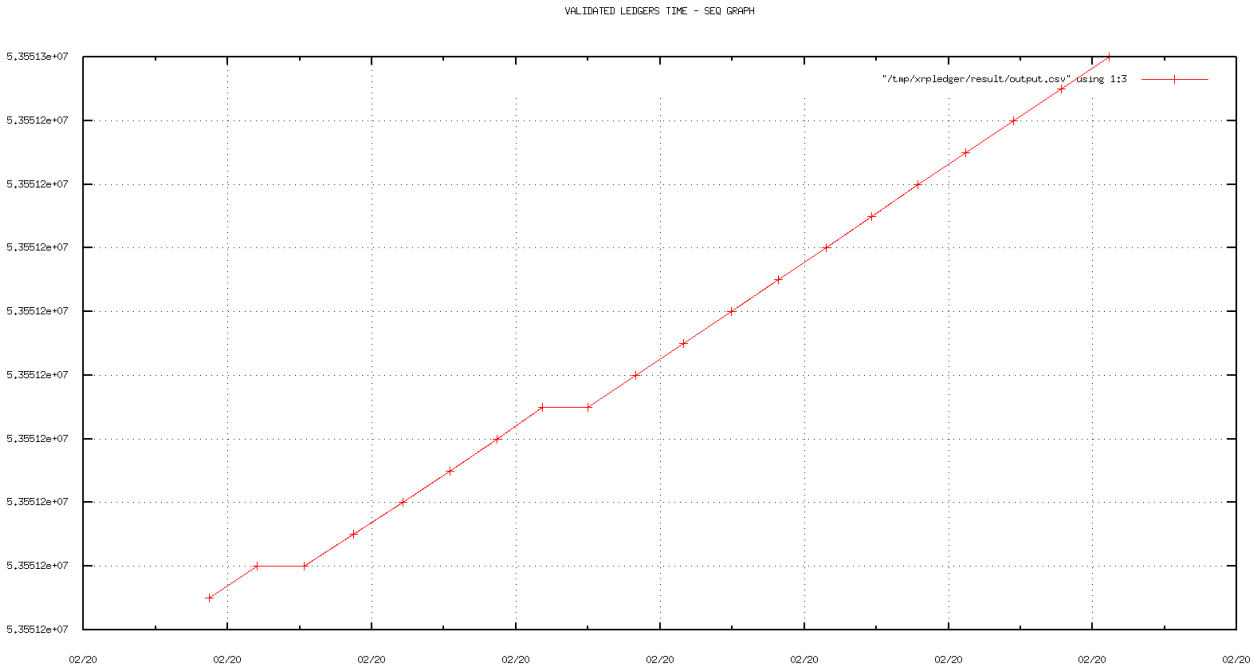
Test 2: Polling Interval = 2 second; Polling Count: 20

- The sequence number increases with respect to time in a linear progression.
- In most cases, the sequence number is repeated twice during the query cycle.
- The Average, Maximum and Minimum time taken for new ledger to be validated from the extracted data was 3.72727, 5 and 2 respectively.



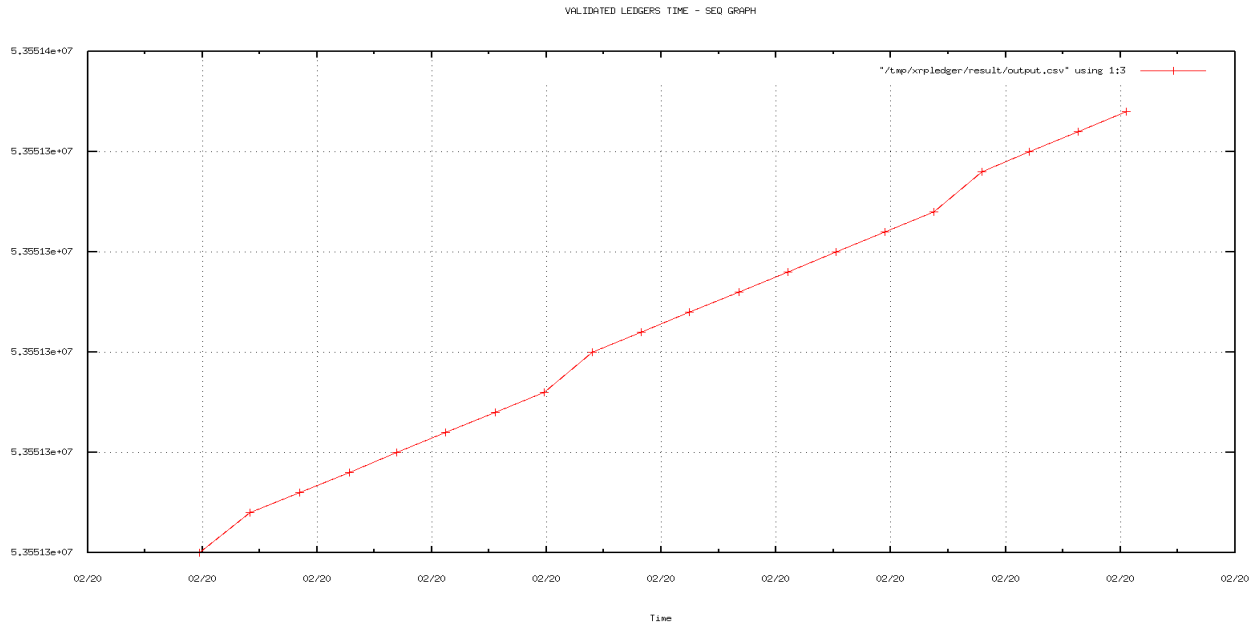
Test 3: Polling Interval = 3 seconds; Polling Count: 20

- The sequence number increases with respect to time in a linear progression.
- In very few cases (2 out of 20 responses), the sequence number is repeated twice during the query cycle.
- The Average, Maximum and Minimum time taken for new ledger to be validated from the extracted data was 3.70588, 7 and 3 respectively.



Test 4: Polling Interval = 4 seconds; Polling Count: 20

- The sequence number increases with respect to time in a linear progression.
- In no cases was the sequence number in the response repeated.
- The Average, Maximum and Minimum time taken for new ledger to be validated from the extracted data was 4.26316, 5 and 4 respectively.



2.3. WHAT DO THE RESULTS TELL YOU?

The results have led me to conclude the following:

- An increase in polling Interval leads to a more linear graph without repetition of sequence numbers. This shows that it mostly takes **three seconds or more** for a new ledger to be validated.
- The time it takes for a new ledger to be validated is not constant. The time varies.

2.4. HOW DID YOU DECIDE ON THE POLLING INTERVAL?

After reviewing the output data, I have decided the following:

- Polling interval of 3 seconds or more for graphical illustration/display as it shows a more accurate depiction of the progression of sequence numbers with respect to time.
- Polling interval of 1 second for calculation of “Mean, Maximum and Minimum Time Taken for New Ledger to Be Validated”, however I have applied a filter to remove records with duplicate sequence numbers to allow for accuracy.

2.5. WHAT MIGHT EXPLAIN THE VARIATION IN TIME BETWEEN NEW LEDGERS?

The variation in time can be as a result of the following scenarios:

- Difference in time taken to achieve reliable convergence for each consensus round. If all participants agree to include a transaction if 50% or more of other participants agree, then this

will lead to a quicker convergence time. However in the case of disputes, the threshold keeps increasing which will lead to a delay.

- If a new consensus round must occur because no supermajority is clear from received validations. Although extremely rare, this could lead to a network losing a few seconds.

2.6. THERE ARE SOME OTHER BETTER WAYS YOU COULD USE RIPPLE API TO FIND HOW LONG EACH LEDGER TOOK TO CLOSE/VALIDATE. USING API DOCUMENTATION, FIND AND DESCRIBE ONE OF THESE METHODS.

One of the other ways the Ripple API can be used to find closure and validation times is by using the "ledger methods" to retrieve information from the public ledger.

A Web Socket Request, the ID of a Ledger can be passed in a request:

```
{
  "id": 100000,
  "command": "ledger",
  "ledger_index": "validated",
  "full": false,
  "accounts": false,
  "transactions": false,
  "expand": false,
  "owner_funds": false
}
```

The response includes the following which can be useful in finding out how long each ledger took to close/validate.

- `ledger.close_time` - Integer - The time this ledger was closed, in
- `ledger.parent_close_time` - Integer - The time at which the previous ledger was closed.

If we subtract ledger.parent_close_time from ledger.close_time, this will provide how long it took the current ledger to close

2.7. LIST OF SOME CREATIVE IDEAS INCLUDED IN MY SCRIPT

- Archive of previous output data (backup and timestamp) to allow for safekeeping of historical data.
- Addition of user entry with input validation for Polling interval and Polling Count.
- Addition of status bar to indicate the status of the data request/response.
- Improvement of calculation of min, max, and average time that it took for a new ledger to be validated during the span of time captured by filtering out records with the same sequence number.
- Addition of grids and other enhancements to gnuplot graph.

2.8. SCREENSHOT OF SCRIPT AND RESULT