EPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**		
1. REPORT DATE *(DD-MM-YYYY)* 16-02-2017	**2. REPORT TYPE** Technical	**3. DATES COVERED *(From - To)*** 31/05/2011 – 16/02/2017
4. TITLE AND SUBTITLE Real Time Clock Difference Approximation Method for Two Computers over a Variable Latency Network		**5a. CONTRACT NUMBER** **5b. GRANT NUMBER** **5c. PROGRAM ELEMENT NUMBER**
6. AUTHOR(S) Ty Valascho		**5d. PROJECT NUMBER** **5e. TASK NUMBER** **5f. WORK UNIT NUMBER**
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army, Tank Automotive Research Development and Engineering Command (TARDEC) Warren, MI 48397		**8. PERFORMING ORGANIZATION REPORT**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) US Army, Tank Automotive Research Development and Engineering Command (TARDEC) Warren, MI 48397		**10. SPONSOR/MONITOR'S ACRONYM(S)** **11. SPONSOR/MONITOR'S REPORT NUMBER(S)**
12. DISTRIBUTION / AVAILABILITY STATEMENT UNCLASSIFIED: Distribution Statement A. Approved for public release; distribution is unlimited.		
13. SUPPLEMENTARY NOTES The views, opinion, and/or findings contained in this report are those of the authors and should not be construed as an Official Department of the Army position, policy, or decision, unless so designated by other documents.		
14. ABSTRACT Describes an algorithm to approximate the difference in internal Real Time Clock (RTC) settings between two computers that are communicating over a network with variable latency. This information can be used to approximate the latency of any message in either direction, not just round trip messages as is commonly done. This method does not rely on a central time server, such as the Network Time Protocol, and only measures the relative clock settings between two computers – not the "true" time.		
15. SUBJECT TERMS Clock synchronization, network latency, robotics		

16. SECURITY CLASSIFICATION OF: **Unclassified / DIST A**			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Ty Valascho
a. REPORT DIST A	**b. ABSTRACT** DIST A	**c. THIS PAGE** DIST A	DIST A	20	**19b. TELEPHONE NUMBER *(include area code)*** (586) 282-4721

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

Real Time Clock Difference Approximation Method for Two Computers over a Variable Latency Network

February 16, 2017

Ty Valascho

US ARMY TARDEC - Ground Vehicle Robotics (GVR)

tyrus.j.valascho.civ@mail.mil

586.282.4721

Executive Summary

Describes an algorithm to approximate the difference in internal Real Time Clock (RTC) settings between two computers that are communicating over a network with variable latency. This information can be used to approximate the latency of any message in either direction, not just round trip messages as is commonly done. This method does not rely on a central time server, such as the Network Time Protocol, and only measures the relative clock settings between two computers – not the "true" time.

Contents

Introduction

This document describes an algorithm to determine the approximate difference in Real Time Clock (RTC) values between two computers communicating over a variable latency network. The RTC of a computer is defined to be its understanding of the current date and time. This is usually represented in computers as the number of occurrences of a specified time increment since a certain specified point in time - for example, the number of milliseconds since midnight on December 31, 1999.

The algorithm is designed to run continuously and accommodate changes to the transit time delay of messages across the network. Knowing the difference in time settings between two computers allows the determination of one-way latency, if the RTC value of the sending computer is included in the message. This is important for robotic control applications and this algorithm was designed to facilitate these latency approximations.

This algorithm was designed to create approximations for RTC difference and should not be used in environments that require high confidence or accuracy less than 100 milliseconds, such as safety critical applications.

Background

Several methods exist to synchronize computer RTCs over a network. One widely used method is the Network Time Protocol (NTP), which relies on very accurate central time servers and a robust method of sending time updates between the time servers and other computers on the network. Once this system is established and working, all clocks can be synchronized accurately.

However, in some system designs it is not desirable to change the clock settings of a computer on a network, or utilize the additional infrastructure required by NTP or similar schemes. Also, if extreme accuracy is not needed, an approximation of time differences as understood by different computing entities on a network will suffice. The algorithm described in this document was designed to the following general requirements. System must:

- Determine approximate RTC differences between two computers connected together on a network / Accurate “real” time – such as Coordinated Universal Time - is not required
- Attain accuracy within 100 milliseconds
- Be robust to variable latency of the network

Description of the Algorithm

In this design, the two computers are called the client and server, with the server determining the time difference based on the sending of short messages to the client and interpreting the responses. Once the time difference is calculated, it could be sent to the client, but the algorithm does not require this.

In this algorithm, the "local time" or RTC settings of the client and server are used as a timestamp when messages are sent and received. This value includes the date and current time of the computer sending the message. The difference between these two local time values is what the algorithm is trying to discern.

best_case_latency

The algorithm relies on the determination and continual measurement of the smallest one-way message latency between the two computers, called *best_case_latency*. The variable *best_case_latency* is an approximation of the shortest transit time a message on this network could experience and is always a positive number. It is calculated by the server sending a message to the client that contains the time the server sent the message, $T_{Msg\_sent\_Server}$. When the client receives this message, it responds to the server with a message that contains the original $T_{Msg\_sent\_Server}$ and the time it sent its response, $T_{Msg\_sent\_Client}$. The time of receipt of this message by the server is then recorded as $T_{Msg\_rcvd\_Server}$.

From this message, *best_case_latency* is calculated as shown in Equation 1.

Equation 1

best_case_latency = MINIMUM[(*best_case_latency* + *best_case_latency_increase_calibration*), *average_roundtrip_latency*]

Where *average_roundtrip_latency* = (($T_{Msg\_rcvd\_Server}$ - $T_{Msg\_sent\_Server}$) / 2)]

The constant *best_case_latency_increase_calibration* is set to a value much less than 1 second (a value of 10 milliseconds was used in testing). The purpose of this calibration is to prevent cases where network conditions degrade and the *best_case_latency* value should increase. While this is not common, it does happen.

The algorithm updates the value of *best_case_latency* every time this message is sent from the server, which is a period of time determined by how quickly the system needs the latency values updated. A period of 5 Hz was used in testing.

delta_time_client_server

The second part of this algorithm computes the difference in clock settings between the two computers, from the perspective of the server and is called *delta_time_client_server*. This is a signed integer, with positive values indicating the client's RTC is less (earlier) than the server's RTC and negative values indicating the client RTC is set to a higher (later) value.

Through testing, it was found that this value should be filtered for stability. The filter used during testing consisted of maintaining a sliding window of the last 100 values of *delta_time_client_server*,

which are averaged every time a new *best_case_latency* is determined and called *averaged_delta_time_client_server.*

This filtered value is compared to the latest determination of the time differences between client and server to get the current value of *delta_time_client_server* as seen in Equation 2.

Equation 2

$$delta\_time\_client\_server = MINIMUM[\ (averaged\_delta\_time\_client\_server + delta\_time\_client\_server\_increase\_calibration),\ (T_{Msg\_rcvd\_Server} - (T_{Msg\_sent\_Client} + best\_case\_latency))\ ]$$

Similar to the *best_case_latency*, a *delta_time_client_server_increase_calibration* of 10 milliseconds was used to prevent this minimum value from getting stuck in "valleys" when underlying conditions change, such as the time setting of either computer or an increase in the physical "best case" time for the message to travel across the communication medium.

The value of the calculation $T_{Msg\_rcvd\_Server} - (T_{Msg\_sent\_Client} + best\_case\_latency)$ is then added to the sliding window and the oldest value removed.

Testing Procedures

Client / Server RTC Verification Test

Synchronized Times

1. Synchronize client and server RTC values to within 50 milliseconds of one another.
2. Start both client and server and connect both to the same network.
3. Wait for the calculated *delta_time_client_server* to stabilize, by noting when changes over time are within a 50 millisecond window of one another.
4. Verify and record RTC values of client and server every 1-2 minutes for 3 total times.

Client 120s Before

1. Synchronize client and server RTC values so that the Client's time setting is 120 seconds before the server's. Record actual values of both client and server simultaneously.
2. Start both client and server and connect both to the same network.
3. Wait for the calculated *delta_time_client_server* to stabilize, by noting when changes over time are within a 50 millisecond window of one another.
4. Verify and record RTC values of client and server every 1-2 minutes for 3 total times.

Client 120s After

1. Synchronize client and server RTC values so that the Client's time setting is 120 seconds after the server's. Record actual values of both client and server taken simultaneously.
2. Start both client and server and connect both to the same network.
3. Wait for the calculated *delta_time_client_server* to stabilize, by noting when changes over time are within a 50 millisecond window of one another.

4. Verify and record RTC values of client and server every 1-2 minutes for 3 total times.

Results

Verification testing was performed in July of 2011 as the algorithm was being developed, but data collected was incomplete and insufficient test conditions were recorded at that time. This testing was performed using the internet as the network, and the results are summarized for reference in Appendix A – July 2011 Test Results. The verification test was performed again in February, 2017, and those results are the ones discussed at length in this document.

Test Configuration

To perform the verification testing, two computers were connected by a 2.4 GHz Wireless link as depicted in Figure 1.

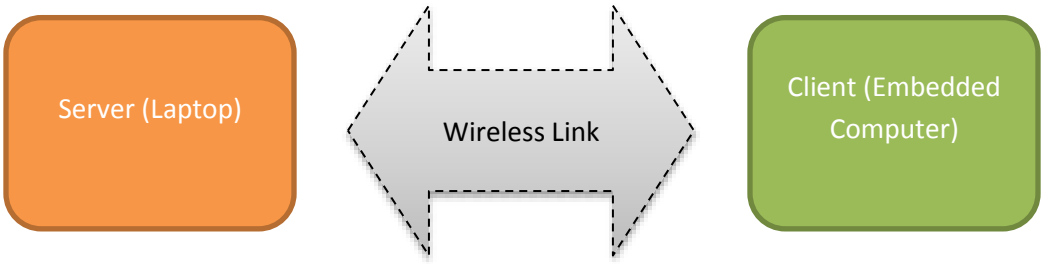

Figure 1 - Testing Configuration

In the test setup, a robotic system was used as the transmission medium but this fact is provided only as supplemental information and is not relevant to the testing that was performed. An image of the actual test setup is provided in Figure 2.

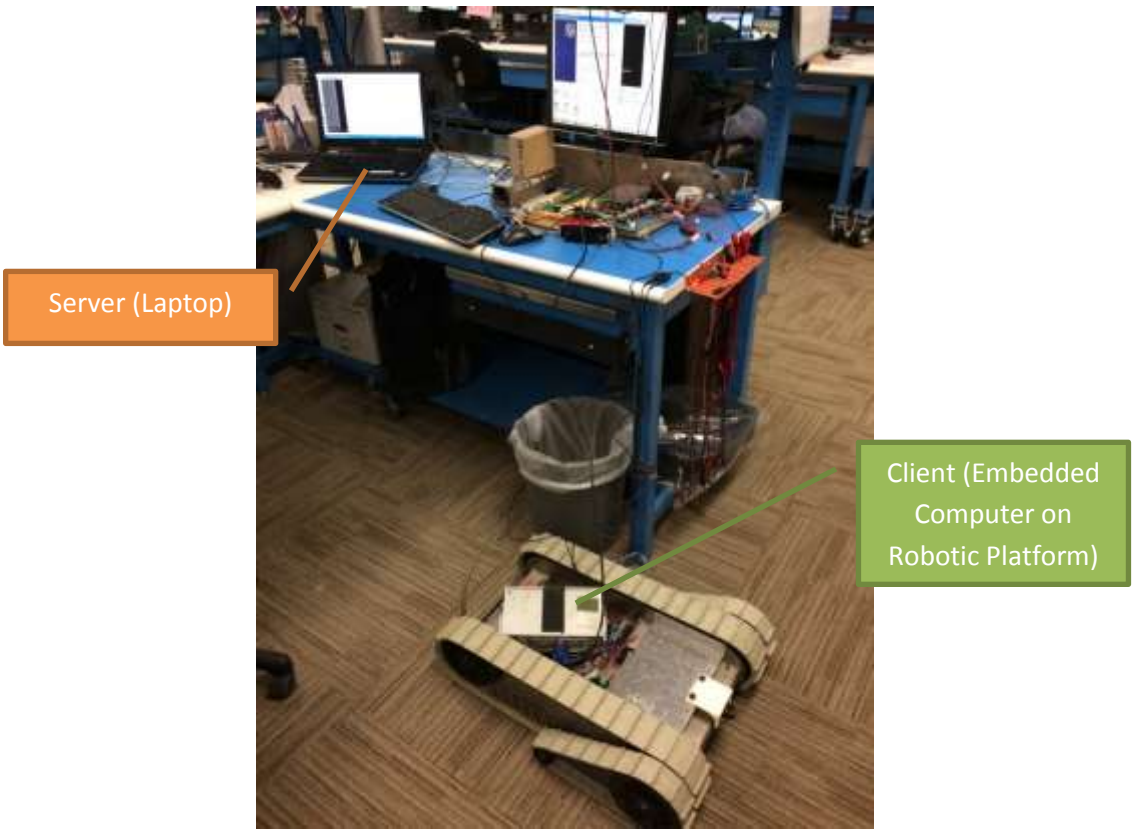

Figure 2 - Test Assets and Configuration

Details about the systems used for testing are provided below.

Server

- **Hardware:** Dell Precision 7510 laptop, service tag #3M6DQ92
- **Operating System:** Windows 8, SP1
- **Software:** LDTO_Client (Release 02.09) and SwitchboardAlpha (Release 02.02)

Client

- **Hardware:** Advantech ARK-3360F, serial number ACA8000069
- **Operating System:** Windows XP, SP3
- **Software:** RobotRedirector (Version 02.03 Beta)

It should be noted that, despite the names of the software packages being used, the SwitchboardAlpha software operated as the server software and the RobotRedirector software acted as the client software during the testing. A more detailed depiction of the test setup is provided in Figure 3.

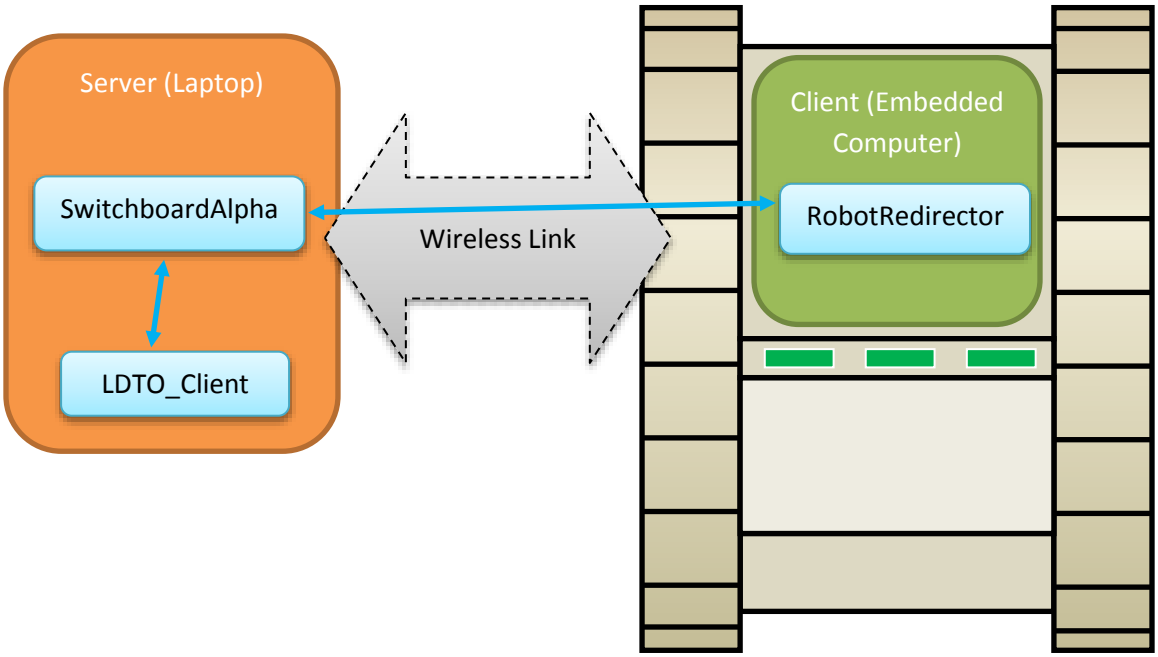

Figure 3 - More Detailed Testing Configuration

The algorithm described by this document was used to derive approximate one-way latency values in near real time between the SwitchboardAlpha / LDTO_Client and also between the SwitchboardAlpha / RobotRedirector. Since the system was configured to have both the SwitchboardAlpha and LDTO_Client software running on the same physical hardware, those results are not interesting when attempting to verify the accuracy of the algorithm under laboratory test conditions. Going forward, the results described refer only to the SwitchboardAlpha / RobotRedirector values, since these messages passed through the Wireless link.

Detailed Test Procedure and Notes

During the test, both computer's RTC values would be set manually by using the Windows command line "time" function. Both values on both computers would be set to the appropriate values for the testing (synchronized / 120 seconds before / 120 seconds after), then both commands executed simultaneously. The two times would then be verified to be within 50 milliseconds of each other before the testing began.

During testing, the SwitchboardAlpha software would be started first, then the RobotRedirector (Client) software, then the LDTO_Client software (server). All messages passed through the switchboard software and the data logging was performed by this software, which recorded several relevant values in a continuously updated csv (comma separated value) file. A description of the values that were recorded is provided in Table 1.

Table 1 - Values Recorded by Server (SwitchboardAlpha)

Data	Description	Source of Info
Ping Sent	RTC Value when the ping msg was sent	Server (SwitchboardAlpha)
Ping Rec'd	RTC Value when the ping msg was rec'd	Client (RobotRedirector)
Pong Sent	RTC Value when the ping response (pong) msg was sent	Client (RobotRedirector)
Pong Rec'd	RTC Value when the ping response (pong) msg was rec'd	Server (SwitchboardAlpha)
Latency Calc	An approximation of latency determined by server	Server (SwitchboardAlpha)
RTC Difference	delta_time_client_server, as calculated by the server	Server (SwitchboardAlpha)
Best Case Latency	best_case_latency, as determined by the server	Server (SwitchboardAlpha)

Client / Server RTC Verification Test Results

3 Trials of each test were performed, and each trial lasted 4-6 minutes with data being recorded continuously. The csv data files were then summarized and graphed. Graphical results for all 9 test runs are provided in Figure 4, Figure 5, and Figure 6.

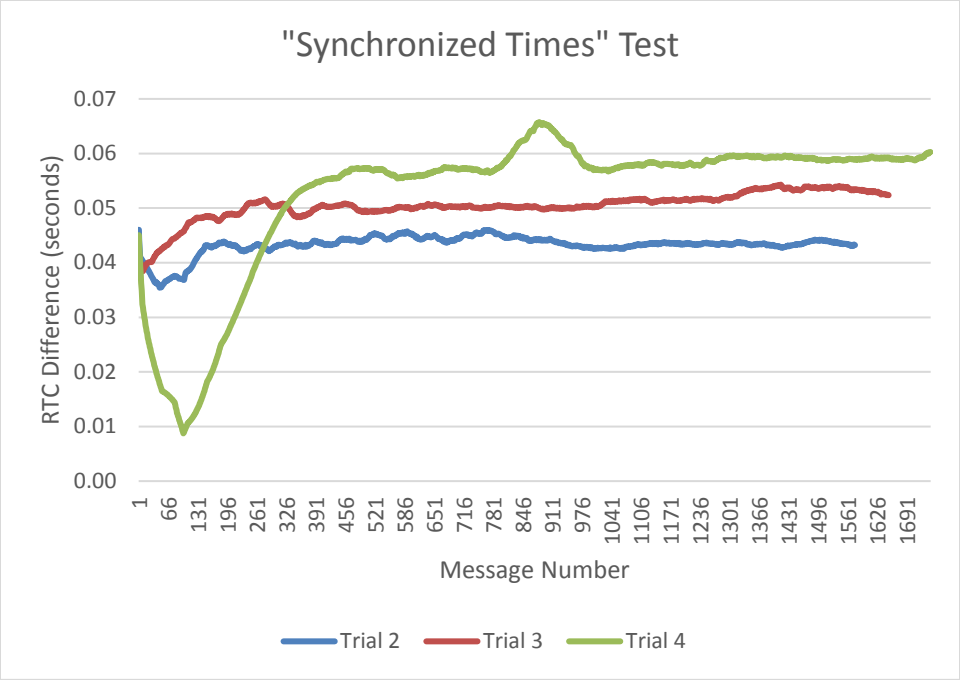

Figure 4 – "Synchronized Times" Graphical Test Results

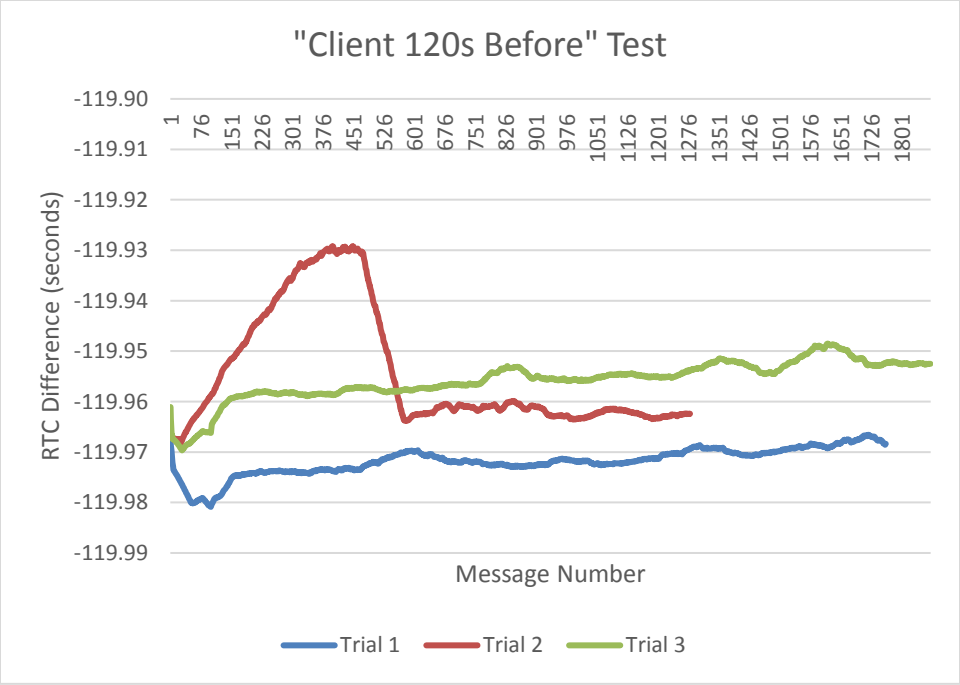

Figure 5 – "Client 120s Before" Graphical Test Results

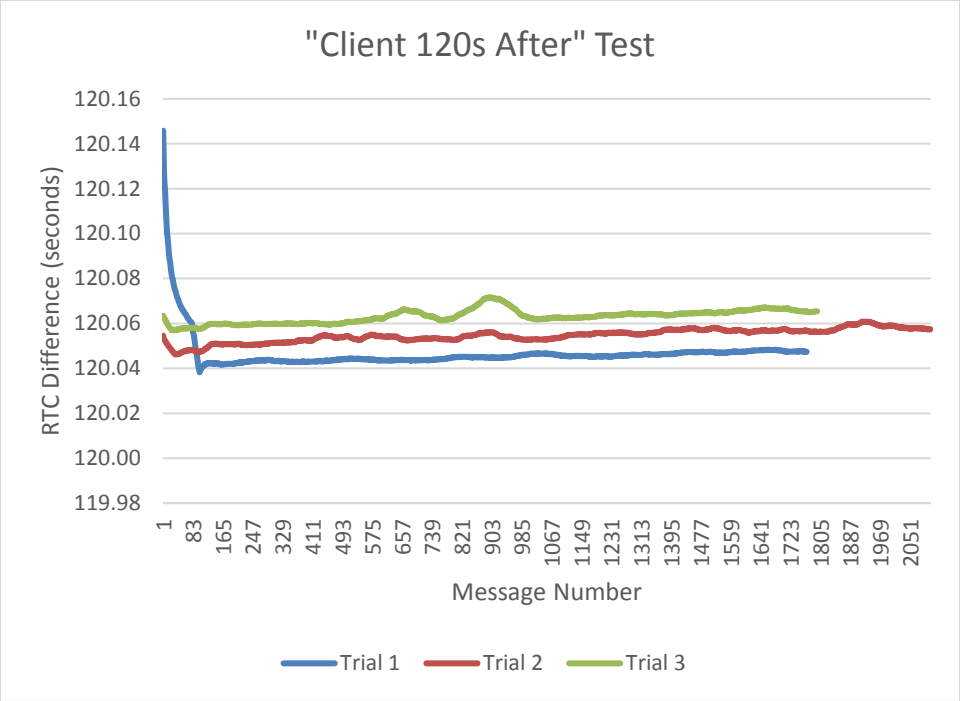

Figure 6 – "Client 120s After" Graphical Test Results

The data for both the calculated values derived by the algorithm and the "manually" measured values has been summarized in Table 2 and Table 3, respectively.

Table 2 - Calculated Values Summarized

RTC Differences Over Time, Calculated									
	Client Before			Synchronized			Client After		
	Trial 1	Trial 2	Trial 3	Trial 2	Trial 3	Trial 4	Trial 1	Trial 2	Trial 3
MAX	-119.97	-119.93	-119.95	0.05	0.05	0.07	120.15	120.06	120.07
MIN	-119.98	-119.97	-119.97	0.04	0.04	0.01	120.04	120.05	120.06
AVG	-119.97	-119.95	-119.96	0.04	0.05	0.05	120.05	120.05	120.06
STD DEV	0.00	0.01	0.00	0.00	0.00	0.01	0.01	0.00	0.00

Table 3 - Measured Values Summarized

RTC Differences Over Time, Measured									
	Client Before			Synchronized			Client After		
	Trial 1	Trial 2	Trial 3	Trial 2	Trial 3	Trial 4	Trial 1	Trial 2	Trial 3
MAX	-119.98	-119.98	-119.97	0.03	0.04	0.04	120.04	120.04	120.06
MIN	-120.02	-120.00	-120.01	0.01	-0.01	0.00	120.02	120.00	120.03
AVG	-120.00	-119.99	-119.98	0.02	0.02	0.02	120.03	120.03	120.04
STD DEV	0.02	0.01	0.02	0.01	0.02	0.02	0.01	0.02	0.01

Long Term RTC Verification Test

To further verify the algorithm's correct operation over time, testing was performed using the same configuration, but over a longer time frame. The testing time was over 60 minutes, during which the manual RTC difference measurements were made every 4-5 minutes.

The results of this testing are displayed in Figure 7.

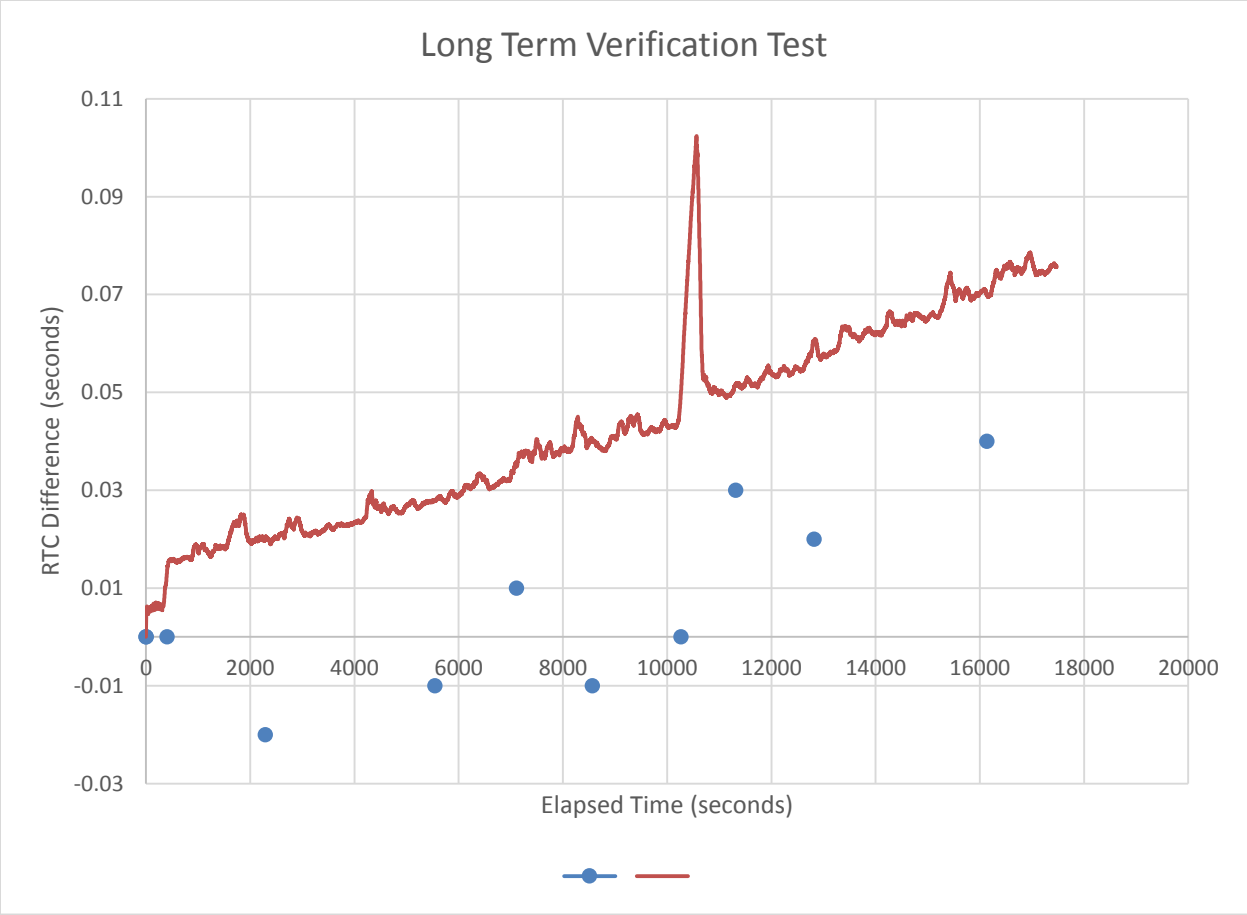

Figure 7 - Results of Long Term Verification Test

The results show good correlation between measured and calculated values over time, and it can be seen that the data plots trend together over the testing period. The increasing nature of the RTC difference values is a result of the drift of the clock crystals within the two computers. A useful feature of the algorithm that can be seen from this is the adaptive nature of the calculated values that adjust as the drift is occurring.

The large spike in the calculated data was caused by a large and sustained increase in the time between "Pong Sent" and "Pong Rec'd" values. That is, the time between when the client time stamped the message to send and the server received it. This anomaly lasted for 216 seconds and caused the *average_roundtrip_latency* term of Equation 1 to be between 0.2 and 2.5 seconds during this time. This caused the *best_case_latency* value to begin incrementing by the *best_case_latency_increase_calibration* amount until the unusual latency event ceased. It is unknown what the root cause of this issue was. The most likely candidates are something in the radio communications of the robot passing the messages back to the server, or something happening on the computer running the RobotRedirecter software that caused the time stamp to be placed in the message, but the message not sent until a later time. This highlights a weakness of the algorithm:

extended delays in the system due to latency – even one way latency, as in this case - will artificially increase the calculated RTC difference values.

The RTC Long Term Verification Test data is summarized in Table 4.

Table 4 - Long Term Verification Test Results Summarized

	RTC Differences Over Time	
	Manually Measured	Calculated
MAX	0.05	0.10
MIN	-0.02	0.00
AVG	0.01	0.04
STD DEV	0.02	0.02

Conclusions and Recommendations

Comparing the average calculated and measured values for each trial shows that the algorithm calculates the RTC difference between client and server within 30 millisecond of the measured values. This met and exceeded the design goal of 100 milliseconds. None of the Trials had a difference of measured and calculated values more than 40 milliseconds. The results were consistent whether the RTC differences between client and server were positive, negative, or very near zero.

There are two factors that may impact the results when considering these measurements.

First, the method of performing the "manual" measurement introduces some error. The command "time" was typed out on both computers sequentially, then the "Enter" key pressed simultaneously by a human operator. The physical pressing of these keys together must always be off by some time period and they would never be *truly* simultaneous, which introduces some error in the measured values.

Second, the latency was very low and consistent over the network these measurements were taken over. The latency was not measured independently, but it has been noted empirically that under similar conditions to those during the test - robot and laptop within 2 meters one another and no external (RF) Radio Frequency interference present – the latency is very low, probably less than 200 milliseconds.

This first factor did not appear to have a significant impact on the results. The standard deviation numbers of the results were relatively low, indicating consistency with the key presses.

The second factor however does mean that the data is not representative of results from a variable latency system, which is a more difficult environment and represents more accurately the "real world" in which this algorithm might be operating.

Before deploying this system to full scale usage, it is recommended to run these same tests over a network with a higher degree of latency variation and verify similar accuracy results.

Another recommendation is to find a way to be more robust to extended one way latency issues, as observed by the "spike" in the Long Term Verification Test results. One possible solution would be to keep track of client-server and server-client latency separately to make more intelligent inferences about short term disruptions to these communication paths. Alternatively, the *best_case_latency_increase_calibration* value could be made smaller than 10 milliseconds. This solution would slow the increase rate and make the spike smaller and less severe, but sacrifices the responsiveness of adjustments to ambient network degradation.

Appendix A – July 2011 Test Results

The algorithm described in this document was developed during July of 2011 and several informal Trials were performed at that time. The test setup and procedure was not documented thoroughly, however, so the results are not included in the body of this document. The results may still be of interest for the reader, so this appendix includes the known information and results from these tests.

The test setup was similar to the one described in Figure 1, but the details differ in significant ways. Figure 8 provides a pictorial representation of the system that was used for testing.

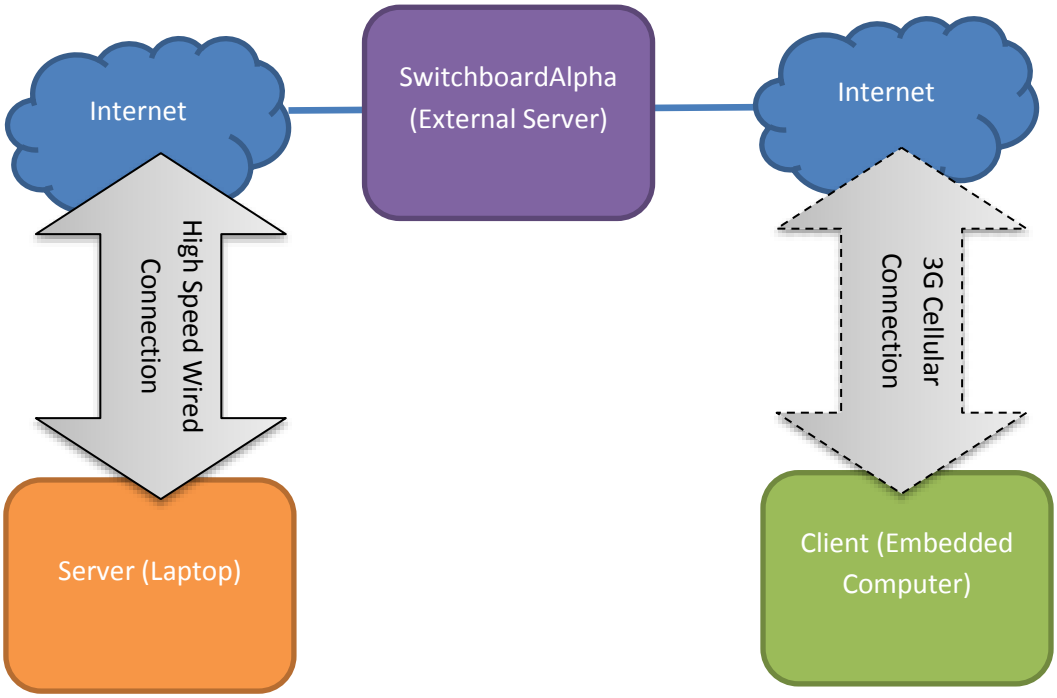

Figure 8 - Detail of July 2011 Test Configuration

In addition, the robot that the Client (Embedded Computer) was installed on was a QinetiQ NA Talon 4™, not a GVR-BOT.

The steps described in the Client / Server RTC Verification Test section of this document were not used, but a similar process was followed where the system was completely powered up and all subsystems were confirmed to be communicating to one another. While logging data, manual measurements were made by remotely logging in to the robot over the internet using the Windows Remote Desktop feature and "manual" simultaneous time measurements were made the same way. Since the key presses were immediate on the Server (Laptop) computer, but had to be passed over the internet and then to a 3G cellular network to the Client (Embedded Computer), those key presses would have had significant delay and the manual measurements are not as accurate as the ones taken for the body of this test report.

The data has been graphed and the five charts provided in Figure 9, Figure 10, Figure 11, Figure 12, and Figure 13.

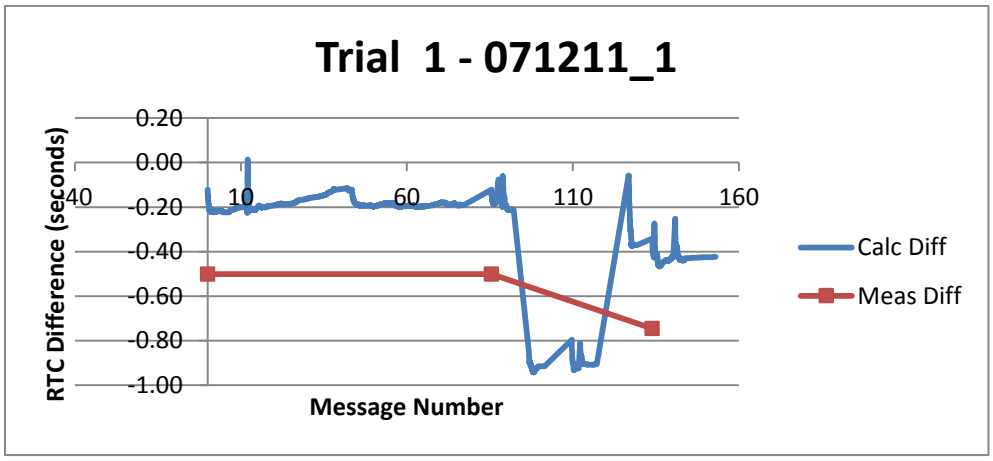

Figure 9 - Trial 1 from July 12, 2011

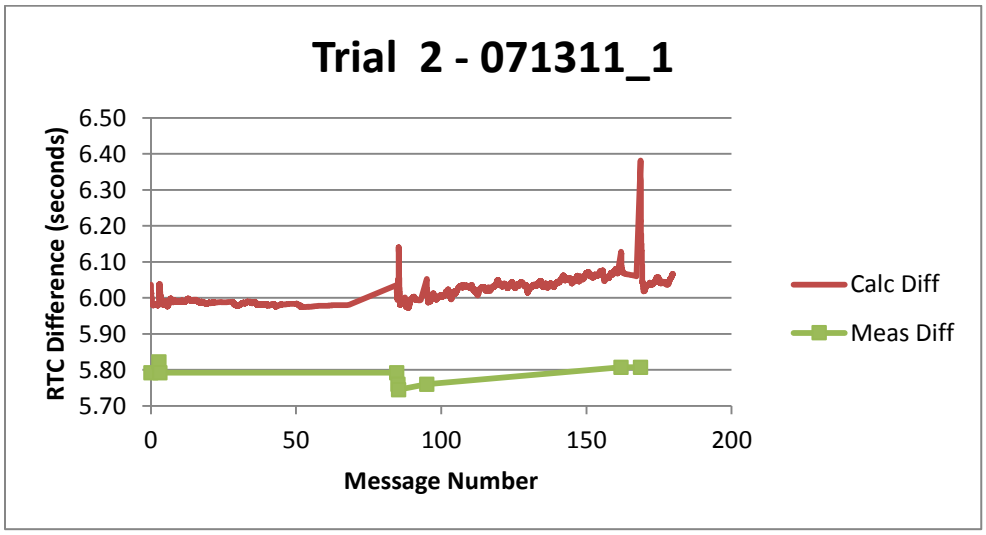

Figure 10 - Trial 2 from July 13, 2011

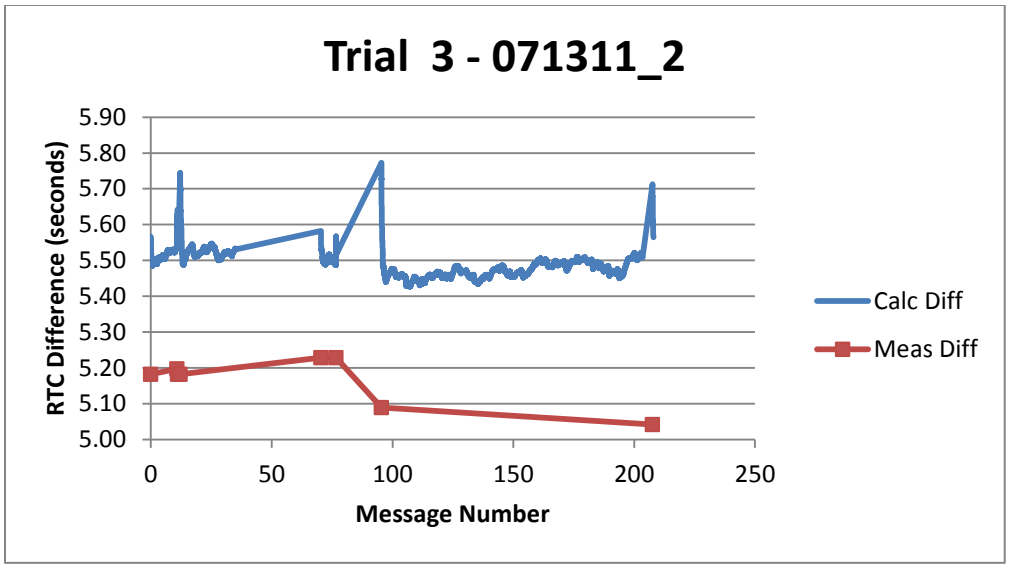

Figure 11 - Trial 3 from July 13, 2011

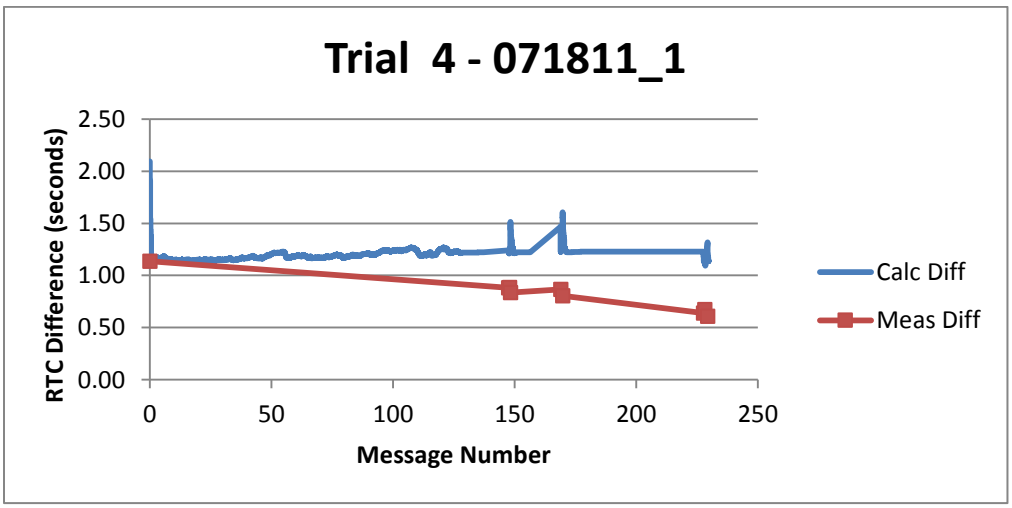

Figure 12 - Trial 4 from July 18, 2011

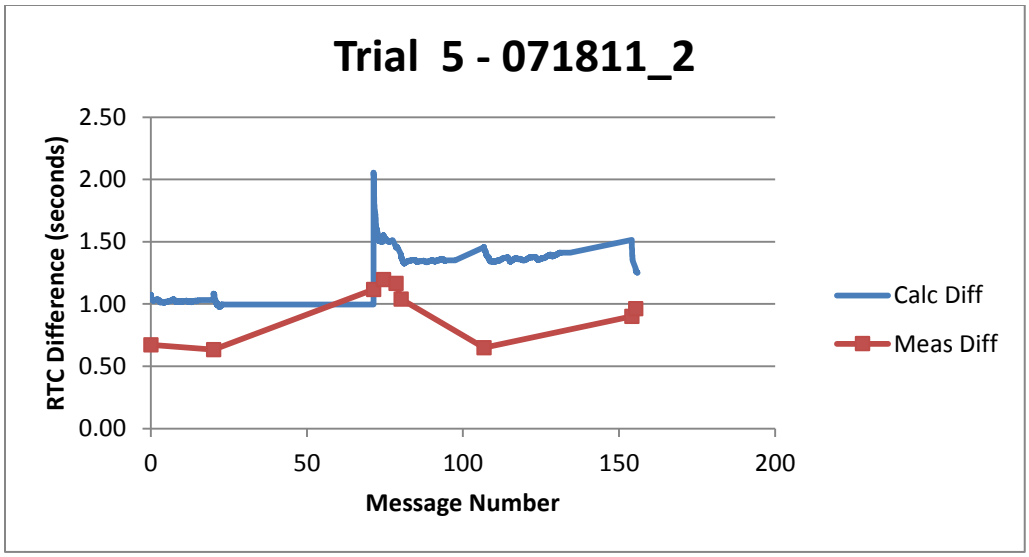

Figure 13 - Trial 5 from July 18, 2011

It can be seen that the measured and calculated values are not as accurate as the other test data in this document. It is theorized that this is due to the aforementioned delay in key presses to the Client. However, general trends in the data do appear to support the accuracy over time of the algorithm. That is, increases and decreases in the measured values are generally reflected as changes in the calculated values. Also, this data was taken over networks with greater latency variability – one of the conditions to which the algorithm is intended to be resilient.