EECS 428 – Final Project Report
Distributed Real-Time Process Control Over TCP and the Internet
Brian Robinson

1.0 Introduction

Distributed real-time process control, from a network communication view, involves sending and receiving data values from one controller to another controller. These data values are very sensitive to loss, but are also very sensitive to timing delays. Because of the sensitivity to loss, TCP is often chosen to implement a real-time data transfer. This is good from a loss point of view, but bad from a timing delay point of view. TCP offers no guarantees about round trip time (RTT), especially with regards to jitter. Jitter in TCP is mainly caused by two things: TCPs congestion control and routers and switches and their effect on IP and Ethernet packets. Both routers and switches can use a store and forward method involving the queuing of incoming packets, which can lead to variability in the delivery times of the packets in cases when the queues are growing and shrinking over time. By using TCP, with its slow start and congestion avoidance phases causing changes in queue sizes over time, these delays are very evident and contributes very heavily to the jitter in round trip times of packets.

2.0 Experimental Design

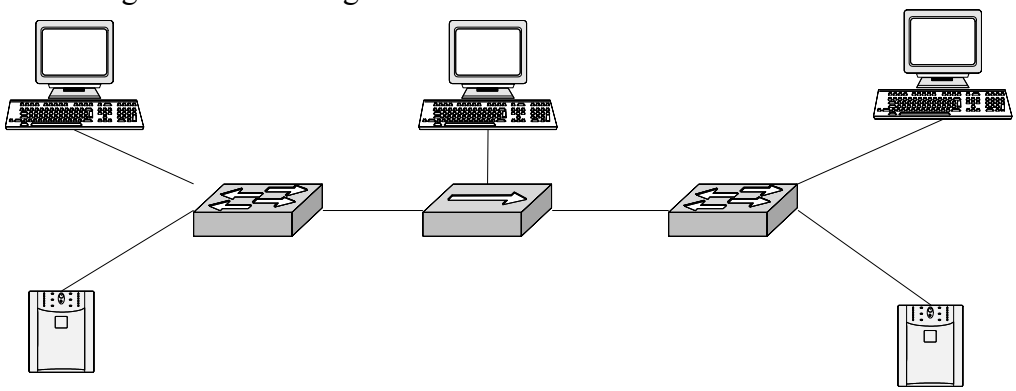
This project aims to use a set of experiments to show the effects of jitter and loss on a real-time Ethernet controller. Two controllers will be set up in various arrangements and with varying amounts of contentious cross traffic. This traffic will take the following forms: no traffic, continuous traffic, bursty traffic, and random traffic over the internet.

These experiments will be run with two controllers separated by varying distances and traffic, each with a distributed algorithm running. The local controller has the PID and is connected to the remote controller via a TCP connection. This connection passes the process variable from the remote controller to the local controller. This process variable is fed into the PID and the resulting output value is sent back to the remote controller over the TCP connection. Both controllers are run with a cycle time of 50 ms and a remote value update time of 100 ms. These settings are the normal 2:1 ratio of 2 scan cycles per remote value update period. In the remote controller, the output of the PID is inputted into the following equation:

$$NewPV = \alpha * OldPV + (1-\alpha)* PIDOutput$$

The alpha value will be changed from 0.9 to 0.7 in increments of 0.5, and the gain setting on the PID will be changed from 1.0 to 3.0 in increments of 1.0. The PID's set point was set to 50.0. Each test starts the output value and process variable at 0.0. Then, the PID is put in Auto mode, and the controller logic will calculate the time it takes for the process variable to converge to within 0.001 of the target set point. This is repeated 20 times for each of the different gain and alpha values.

The first three traffic cases will involve two controllers in the same physical location. These two controllers will each be connected to a switch. These switches will be connected together via a hub. This hub, which will only have the two switches and a monitoring PC connected, allows for network traffic to be monitored and utilization of the network to be displayed. Each end switch will have a PC connected. This PC will generate the bursty traffic and continuous traffic for those specific tests. Below you will find a diagram of the configuration for these tests.



The Ethernet controllers used above are ABB's AC800M controllers, currently in use world wide. The hub in the center allows for the monitoring PC to access all IP packets traveling across the link.

The next part of the experiment involved moving the remote controller farther away and seeing the effect that internet traffic would have on this process convergence algorithm. This was done in two steps. Step one involved moving the controller farther away in the local intranet of ABB Wickliffe. Using tracert, a windows trace route program, it was determined that the remote controller was only two hops away from the local controller. This was the maximum local distance on our intranet, and was the most convenient, so it was chosen. Step two involved getting a controller in Sweden hooked up, and running the distributed algorithm through that controller there. By sending data all the way from Sweden, round trip times will be larger, and traffic will be patterned after the internet more than the local experiments.

Windows XP
PC

3.0 Experimental Results

The first test completed on the above controller configuration was the no traffic test case. This test was to be a baseline for comparison of the jitter effects. Since there was no other traffic, jitter will be at a minimum. These tests yielded the following results:

Alpha	Gain 1.0				Gain 2.0				Gain 3.0			
Factor:	Avg:	Min:	Max:	St. Dev.	Avg:	Min:	Max:	St. Dev.	Avg:	Min:	Max:	St. Dev.
0.9	600	600	600	0	800	800	800	0	1805	600	2400	546
0.85	700	700	700	0	950	700	1000	100	1600	400	2400	604
0.8	700	700	700	0	965	500	1300	251.9	6930	1300	15500	3916
0.75	600	600	600	0	1310	800	1800	278.9	DNC	DNC	DNC	DNC
0.7	800	800	800	0	1490	700	2400	424.1	DNC	DNC	DNC	DNC

The above results show that for any alpha values when gain is 1.0 there is no standard deviation. This process is very stable. For a gain of 2.0, the standard deviation is growing as the alpha value shrinks. This process is stable, but takes more variance around the set point before converging to it. For the gain of 3.0, all values are stable except alpha values of 0.75 and 0.7, which never converged and remained unstable, bouncing wildly above and below the set point.

The next experiment run was the continuous traffic test case. For this set of runs, a large FTP transfer was started, and all tests were run while the file transfer was going on. Network utilization was listed on the hub at 80+% and both switches were reporting some collisions. The network was very highly utilized. Round trip times were seen fluctuating from as low as 10 ms to as high as 215 ms. The following data was recorded:

Alpha	Gain 1.0				Gain 2.0				Gain 3.0			
Factor:	Avg:	Min:	Max:	St. Dev:	Avg:	Min:	Max:	St. Dev:	Avg:	Min:	Max:	St. Dev:
0.9	730	600	1100	192.2	722.5	600	1000	150.9	1330	600	1800	465
0.85	685	500	1000	130.9	985	500	3000	560.5	3690	1100	4000	677
0.8	855	400	1000	179.1	1745	400	3900	953	22218	1500	110600	33116
0.75	1247.5	600	1600	339.3	2314	600	7100	1432	DNC	DNC	DNC	DNC
0.7	1682.5	800	1900	319.2	15439	800	140550	38277	DNC	DNC	DNC	DNC

The above data contains lots more variation than the base case. For gain of 1.0, the standard deviation was much larger than when no contentious traffic was present. It varied from 130 ms to 340 ms, which is relatively large compared to 0 ms without traffic. For a gain of 2.0, the standard deviation again changed greatly. In fact, alpha of 0.7 was unstable, taking 140 seconds to converge in one case. These standard deviation values are growing along with a reduction in the alpha value used in the process. Finally, for a gain of 3.0, we see that for alpha value of 0.8 the process was unstable taking up to 110 seconds to converge, where it was stable before when no traffic was present.

In addition to the standard deviation, it is interesting to note that the average of the data points for the larger alpha values are very close to what they were in the base line case. Only when alpha gets small does the average increase significantly from the first data set.

The final local tests run were with bursty traffic. A burst generator was created that allowed a PC to generate load on the network at a rate and delay period of the users choosing. Two different burst patterns were used. For the first burst experiment, a 50Kbyte burst every 150 milliseconds was generated. These bursts gave the network an average load of between 12% and 25% utilization, with occasional variation above 25% utilization. The following data was collected:

Alpha	Gain 1.0			St.	Gain 2.0			St.	Gain 3.0			St.
Factor:	Avg:	Min:	Max:	Dev:	Avg:	Min:	Max:	Dev:	Avg:	Min:	Max:	Dev:
0.9	837.5	400	1100	223.5	1090	600	1450	262.4	2063	1200	2800	509
0.85	1018	250	1500	284.4	1538	750	2500	496	4210	750	9500	2484
0.8	1190	650	1600	246.3	2313	1000	4600	834	72453	10600	379800	96216
0.75	1028	600	1750	337.7	5348	2450	8200	1614	DNC	DNC	DNC	DNC
0.7	1790	800	2400	357.1	11508	2700	38300	10527	DNC	DNC	DNC	DNC

This data shows that small bursty traffic can be worse than continuous high utilization traffic for distributed process control. In almost all cases, the average and standard deviations for the bursty traffic above are worse than those recorded in the continuous traffic case. In addition, one process was completely unstable with small bursty traffic loads. The Gain 3.0 with alpha of 0.8. It took up to 379 seconds to converge in one case. One additional thing to note: For the processes that went out of control under this bursty traffic, they came back under control once the traffic was stopped. This shows that for any bursty traffic over time, a process can go out of control and recover on its own, which could be seen as an anomaly to a plant operator.

In the second experiment, the bursts were 100Kbytes for 150 milliseconds. These settings gave the network between a 25% and 50% utilization rate, with occasional variation above 50% utilization. The following data was recorded:

Alpha	Gain 1.0			St.	Gain 2.0			St.	Gain 3.0			St.
Factor:	Avg:	Min:	Max:	Dev:	Avg:	Min:	Max:	Dev:	Avg:	Min:	Max:	Dev:
0.9	970	350	1550	348.1	1535	1050	2300	376.7	2613	650	3375	840
0.85	1000	200	1800	449	2205	1150	3800	685	9083	3300	22550	5579
0.8	1315	750	1850	256	4165	1300	8850	1863	53388	650	111250	37685
0.75	1443	750	2600	518	7685	600	17050	5136	DNC	DNC	DNC	DNC
0.7	2063	1000	3650	603	30088	2800	73500	20818	DNC	DNC	DNC	DNC

This data shows that for bursty traffic, the standard deviation is even worse than for the lower burst data. In fact, for almost all cases, the standard deviation is significantly worst than for the smaller burst traffic. Also, the averages for the convergence are higher. For the gain of 1.0 case, the average is about 100 to 300 ms longer. For the gain of 2.0 and 3.0 the average time to convergence is mostly double that

of the lower burst data with only very few being less than double. In addition, two processes could be considered unstable. The process with gain 2.0 and alpha of .7 and the gain of 3.0 and alpha of 0.8 both took really long times to converge in the worst case, while they also converged quickly in the best case. These two burst data sets show some form of relationship between bursty traffic (and its size) and the delays in convergence of the distributed algorithm.

Part two of this project dealt with moving the remote controller farther away from the local controller and having real internet traffic cause delays. Step one was moving the controller on our intranet here at ABB Wickliffe. The remote controller was moved two hops away to our test center. This area is traffic intensive, but it turned out to not be traffic intensive over a contentious link. The results are shown below:

Alpha	Gain 1.0			St.	Gain 2.0			St.	Gain 3.0			St.
Factor:	Avg:	Min:	Max:	Dev:	Avg:	Min:	Max:	Dev:	Avg:	Min:	Max:	Dev:
0.9	600	600	600	0	600	600	600	0	600	600	600	0
0.85	500	500	500	0	1000	1000	1000	0	1100	1100	1100	0
0.8	1575	1200	1900	144.6	2700	2700	2700	0	2160	1300	3000	441.8
0.75	1576.9	1000	2000	200.6	1205	700	2000	413.6	DNC	DNC	DNC	DNC
0.7	875	700	1100	125.1	5200	1400	9800	2370	DNC	DNC	DNC	DNC

This data shows some variation in the standard deviations from the base tests in the higher alpha values. But, other than this slight variance, there was no difference in standard deviation from this and the base class. This is due to a lack of contention across the two hops of the switch from our remote controller to our local controller. There is not much traffic out of the test center.

Step two of the remote data gathering dealt with using a remote controller in Rochester, NY. This remote controller was 5 hops away. The traffic is more random in this experiment, due to crossing 2 hops of an ISP connection. The results are shown below:

Alpha	Gain 1.0			St.	Gain 2.0			St.	Gain 3.0			St.
Factor:	Avg:	Min:	Max:	Dev:	Avg:	Min:	Max:	Dev:	Avg:	Min:	Max:	Dev:
0.9	1123	800	2100	293.1	2278	1450	3100	698	4910	2600	7200	1509
0.85	1878	1450	4050	569	3290	900	4900	1319	17983	4500	42350	11303
0.8	2085	800	4750	900	3365	1700	4900	924	86640	8850	272350	76764
0.75	2503	950	4300	862	5875	1300	10500	2792	DNC	DNC	DNC	DNC
0.7	3178	1400	7100	1341	85158	600	290000	76945	DNC	DNC	DNC	DNC

This data shows that as the alpha value decreases, the average convergence times and standard deviations increase. There are two processes in the above data sets that lose stability. The first is gain 2.0 and alpha value of 0.7. The maximum time to convergence is 290 seconds. The second is gain 3.0 and alpha of 0.8. Its maximum time to converge is 272 seconds. These two are the only that lose stability. All other data points are worse then the two hop data. This data set is similar in both average and standard deviation to the bursty data sets.

Step three of the remote data gathering dealt with setting up a remote controller in Sweden and controlling it via the internet. This remote controller was 9 hops away. The data is shown below:

Alpha	Gain 1.0			St.	Gain 2.0			St.	Gain 3.0			St.
Factor:	Avg:	Min:	Max:	Dev:	Avg:	Min:	Max:	Dev:	Avg:	Min:	Max:	Dev:
0.9	2895	2400	4400	776	5990	2000	9000	3073	DNC	DNC	DNC	DNC
0.85	3653	1600	6600	1409	37120	5800	128800	34847	DNC	DNC	DNC	DNC
0.8	4680	1600	7600	1852	DNC	DNC	DNC	DNC	DNC	DNC	DNC	DNC
0.75	5998	4400	9400	1430	DNC	DNC	DNC	DNC	DNC	DNC	DNC	DNC
0.7	6680	6200	7000	402.1	DNC	DNC	DNC	DNC	DNC	DNC	DNC	DNC

This traffic shows large variation from the 5 hop data. There were 5 processes that never converged at all. Three different runs for each non converging process were run and stopped after 300 seconds. In addition to the 5 non converging processes, there was one process that went unstable. This unstable process has a gain of 2.0 and alpha value of 0.85. It takes, at maximum, 128 seconds to converge. This data is the worst of all the data sets, in terms of both stability and convergence times.

The final factor looked at in this experiment was some characterization of the round trip times in the various experiments above. These values were captured right before running the data sets above.

RTT Data:
20 samples

Traffic Type	Avg:	Min:	Max:	St Dev:
None	10	10	10	0
FTP	16.45	10	110	22.17
50k, 150ms	24.6	10	63	18.23
100k, 150ms	40.7	10	93	27.42
2 hop	10.3	10	16	1.342
5 hop	39.95	31	47	8
9 hop	145.3	140	157	7.43

The round trip time data is interesting. It correlates well with the data dealing with convergence times. Starting at the no traffic case, we see that the RTT has a standard deviation of 0, and the convergence time has a deviation of 0 (for gain 1.0) and small standard deviations in convergence time for the higher gain values. Then for FTP traffic, we see that it has some standard deviation, but that deviation is less than that of the two burst traffic cases (if you take out the one 110ms outlier in the FTP). By looking at the FTP RTTs over time, you can see that occasionally, a big time comes through, but on the whole, things are very stable. Then, for the two burst traffic cases, the standard deviation for the smaller burst traffic is less than the deviation for the larger burst traffic. This fits well with the standard deviation for the convergence time of the smaller bursts being less than the deviation for the times of the larger bursts. Then, for the 2 hop data, we see that almost no variation is seen, which explains the lack of deviation in the convergence times for the 2 hop tests. The two remote tests vary from this pattern slightly. The 5 hop tests show some variation in RTT, but not as much as any of the local tests. But, the deviation in the convergence times for the 5 hop data is almost as bad as the large bursty traffic case. Finally, the 9 hop data shows that even with a small standard deviation, there is still a large variance in the convergence times for all processes. This 9 hop data had the most variation of any of the data sets.

4.0 Summary

This experiment shows that different types and forms of traffic cause different results in distributed real time process control. Most people would assume that constant, high utilization traffic would be the worst. But, in reality, it was shown that 12%-15% utilization traffic with high burstiness is as bad or worse then constant high utilization traffic. In addition, certain processes can seem stable when run with no network traffic, and can go out of control when bursty or heavy traffic appear. Finally, this experiment seems to show that processes can become unstable with either large round trip times or with large standard deviation, or both.

5.0 Future Work

There are definitely areas for further exploration from where this paper left off. First, only one controller cycle time was used. It would be interesting to see how longer cycle times are affected by the delays in the above experiments. Second, many different burst cycles could be developed to see if any correlations can be found between burst periods and burst size versus convergence time. Third, how would multiple imported values from one or more controllers react to the delays in this experiment. Finally, a real process could be run, showing visually that a process can go out of control from the network delays shown above.