



White Paper

Title: Recording Synchronized Data from Multiple Net F/T Sensors		Revision Level: 1
Written by: David Fleissner	Original Issue Date: 3/11/13	Effective Date: 3/11/13

Purpose:

A method to collect synchronized data from multiple Net F/T devices and save it with an accurate time stamp for future data analysis. This process enables time stamping data from multiple sensors on the same time scale. This does not enable hardware-synchronized data and this does not apply to real time synchronization applications such as controls.

Note:

- A sample Java application with source code is provided that collects data from multiple Net F/T devices and stores them in .CSV files. The sample application is located in the "Samples" folder on the CD and is on the software downloads page of the ATI website at http://www.atia.com/Products/ft/software/net_ft_software.aspx.

Problem:

Net F/T devices are not hardware synchronizable and there will be some variation in sampling rate between devices based on internal clock rates.

The standard timer function in Windows typically only has a resolution of about 10-15 ms. This is not accurate enough to timestamp a data sample because a Net F/T can transmit at up to 7000Hz, or ~143 μ S between packets.

Another timing problem arises from the jitter in arrival of UDP packets. Even with a highly accurate timer, the transmission delay of UDP packets will be somewhat random and will introduce unnecessary noise to data.

Solution Overview:

The first step towards giving data an accurate time stamp is using a high-resolution timer. A high-resolution timer is typically used to measure elapsed time in nano seconds or clock cycles. A high-resolution timer's accuracy can be as good as 1 μ S on a PC.

The second step is to record data samples over some time period; recording the start time, stop time, start RDT sequence, and stop RDT sequence. Timers on a PC are typically not accurate enough to record a sample time for each sample. The solution is to record only the start and stop times using the PC and calculate sample times for all of the samples that arrived in between the start and stop times. This greatly reduces the timing error per sample and is a good solution because the transmission rate of a Net F/T is very consistent. The sample Java application receives data for a period of one second before

Title: Recording Synchronized Data from Multiple Net F/T Sensors	Revision Level: 1	Effective Date: 3/11/13
---	----------------------	----------------------------

calculating the sample time for all of the received samples. Using a longer sample period allows greater compensation of inaccurate timers and UDP jitter but requires a larger data structure to hold the received packets.

Note:

- In an environment with heavy network congestion or a long network path from the sensors to the computer, it is possible that some UDP packets will drop or will arrive out of order. It is also possible for the computer to drop UDP packets if CPU load is too high.

Coding Process:

The first step is to research what high-resolution timers are available for the system performing data collection. For Windows, the functions required to get a high-resolution time stamp are *QueryPerformanceCounter* and *QueryPerformanceFrequency*. Some languages such as Java have built in high resolution timer functions (*nanoTime*) that utilize the most accurate timer available on the system. These timers can only be used for elapsed time because the actual number they return cannot be directly used to get the time of day. The sample Java application uses *nanoTime* to record the *startTime* and *stopTime* of the data-sampling period.

There are a number of methods to receive data from multiple sensors simultaneously. One method is to create a new thread for each Net F/T sensor. Each thread will create a UDP socket and initialize one sensor to stream data to that UDP socket. This model allows handling data from each sensor separately at the expense of program complexity. Another method involves initializing a group of Net F/T sensors to stream data to one UDP socket. This allows data collection to occur in a single thread and is the strategy that the sample Java application follows, but it requires identification of incoming UDP packets based on the source IP address.

The next step is to start receiving packets into an array or similar data structure. The packet array will need to be large enough to contain all of the samples from a sensor over one Sample Period. A dynamic array can be a good choice in this case. Using a shorter sample period will reduce the required size of the array, but this will result in less accurate averaging over time. Packets are recorded in the order they are received, which may not be in the order they were sent. If sorted data is desired, this can be implemented before recording the data to disk or it can be performed afterwards in separate software. The provided Java application sorts incoming packets from each sensor using the Java sorting method *Collections.sort()*. The Java application continuously reads packets into the packet *ArrayList* until the end of the sample period.

Finally, sample times are recorded by looping through the array of received packets and logging the calculated time followed by the relevant sensor data. The following formula is used to calculate each sample time:

$$Sample\ Time = startTime + \frac{(stopTime - startTime)}{lastRDTSeq - firstRDTSeq} * (packetRDTSeq - firstRDTSeq)$$

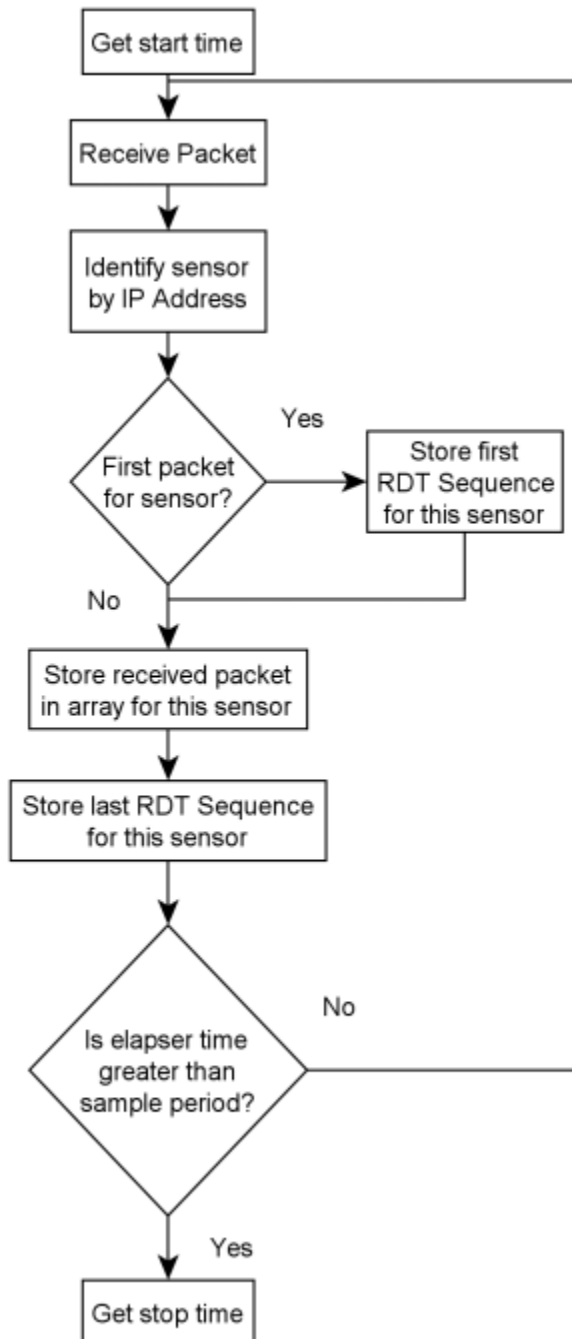
Title: Recording Synchronized Data from Multiple Net F/T Sensors	Revision Level: 1	Effective Date: 3/11/13
---	----------------------	----------------------------

Note:

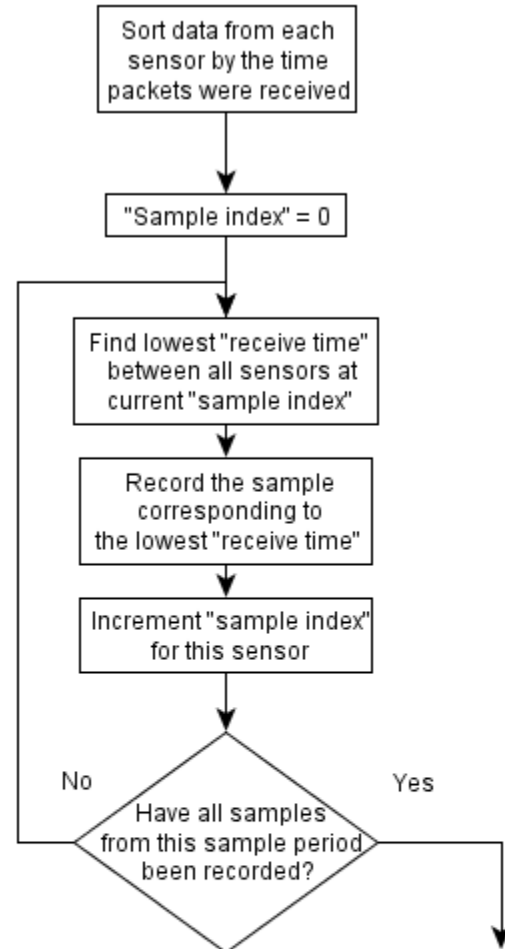
- For information on communicating with a Net F/T, please refer to the user manual.

Algorithm Block Diagrams:

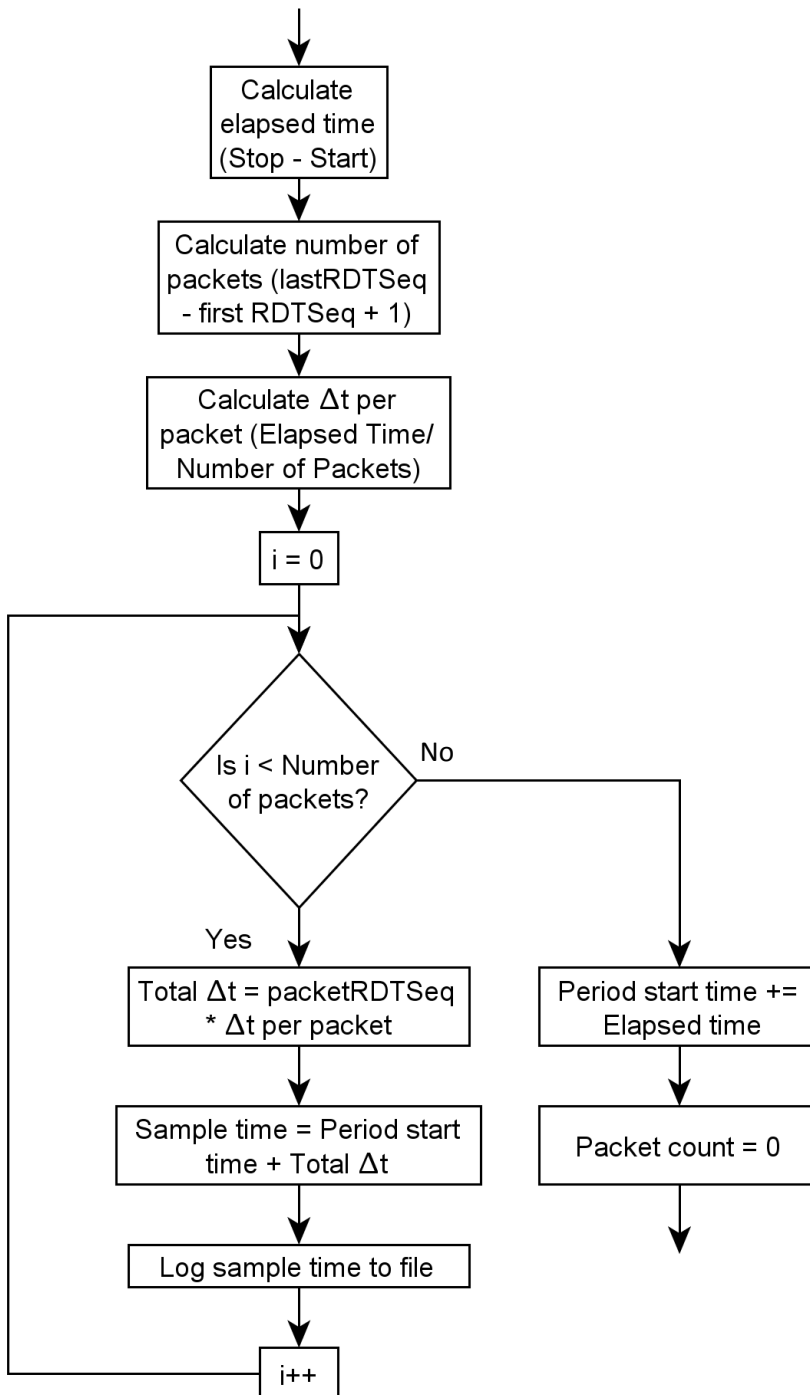
Receive Packets



Sort/Interleave Packets



Calculate Timestamps



Title: Recording Synchronized Data from Multiple Net F/T Sensors	Revision Level: 1	Effective Date: 3/11/13
---	----------------------	----------------------------

Sample Java Code:

Receive Packets:

```

    ///READ SAMPLES///
    //Set start time when first packet arrives
    boolean firstPacket = true;
    m_startTime = 0;
    //Collect samples for SAMPLE_PERIOD worth of time in ms.
    do{
        DatagramPacket dp = null;
        NetFTRDTPacket tempPacket;
        do{//Watch for rogue packets that are not in the sensor map
            try{
                dp = m_netFTGroup.readHighSpeedData(m_highSpeedSocket);
                if(firstPacket){
                    //Set start time - this is used to calculate the sample time for each packet
                    m_startTime = System.nanoTime();
                    firstPacket = false;
                }
                //Set stop time
                m_stopTime = System.nanoTime();
            }catch(Exception e){
                setApplicationError("All sensors timed out: " + e.toString());
            }
            tempPacket = m_netFTGroup.getNetFTRDTPacketFromDatagramPacket(dp);
            //Figure out which sensor sent the packet
        }while(m_sensorIPMap.get(dp.getAddress()) == null);
        int sensorNum = m_sensorIPMap.get(dp.getAddress());
        //Detect and log errors+
        detectSensorErrors(sensorNum, tempPacket);
        //save this packet to the most recently received packet array
        //for the GUI to display.
        m_sensors[sensorNum].m_mostRecentPacket = tempPacket;
        //If this is the first packet, save m_firstRDTSequence
        if(m_sensors[sensorNum].m_firstRDTSequence == -1){
            m_sensors[sensorNum].m_firstRDTSequence = tempPacket.getRDTSequence();
        }
        //Store the packet into the packet array
        m_sensors[sensorNum].m_receivedPackets.add(tempPacket);
        //Save m_lastRDTSequence
        m_sensors[sensorNum].m_lastRDTSequence = tempPacket.getRDTSequence();
    } while((m_stopTime - m_startTime) < (SAMPLE_PERIOD * 1000000));

```

Title: Recording Synchronized Data from Multiple Net F/T Sensors	Revision Level: 1	Effective Date: 3/11/13
---	----------------------	----------------------------

Calculate Timestamps:

```

//Calculate elapsed time
m_elapsedTimeNano = m_stopTime - m_startTime;
for(int i = 0; i < m_numSensors; i++){
    //Calculate number of received packets
    //First, make sure there was something received.
    if(m_sensors[i].m_receivedPackets.isEmpty()){
        continue;
    }
    long lastRDTSeq = m_sensors[i].m_lastRDTSequence;
    long firstRDTSeq = m_sensors[i].m_firstRDTSequence;
    int numReceivedPackets = (int)(lastRDTSeq - firstRDTSeq + 1);
    m_sensors[i].m_numReceivedPackets = numReceivedPackets;
    //Calculate elapsed time per packet in nano sec.
    double deltaTPerPacketNano = (double)m_elapsedTimeNano / (double)numReceivedPackets;

    for(int j = 0; j < (lastRDTSeq - firstRDTSeq) + 1; j++){
        //Check to see if this packet was missing
        //If so, do not record a time for this packet
        if(m_sensors[i].m_receivedPackets.get(j) == null) continue;
        //Calculate total time from beginning of sample period in nano sec.
        double totalDeltaTNano = deltaTPerPacketNano *
            (double)(m_sensors[i].m_receivedPackets.get(j).getRDTSequence() - firstRDTSeq);
        //Calculate total time from beginning of sample period in mili sec.
        double totalDeltaTMili = totalDeltaTNano / 1000000.0;
        //Calculate time in ms. from beginning of testing
        //currentBaseTime = time since beginning of last sample period
        double absoluteTime = totalDeltaTMili + m_periodStartTime;
        //Save sample time to array list
        //This is needed for sorting the data by sample time
        m_sensors[i].m_sampleTimes.add(j, absoluteTime);
    }
}

```

Title: Recording Synchronized Data from Multiple Net F/T Sensors	Revision Level: 1	Effective Date: 3/11/13
---	----------------------	----------------------------

Interleave Sorted Data:

```

//Data from each sensor has been sorted, now it is easy to interleave
//each sorted array into a sorted CSV log file.

//String to append to the log file
String logFileAppendString = "";
int[] lastStoredIndex = new int[m_numSensors];
for(int i = 0; i < m_numSensors; i++)lastStoredIndex[i] = -1;
double minValue;
int minIndex = 0;
boolean continueInterleaving = true;
while(continueInterleaving){
    //Order all data by sample time. Do this by finding the min
    //sample time between all m_sensors at some index, then increment
    //index with min sample time and log values. Also check for null
    //sampleTimes corresponding to a missing UDP packet
    minValue = Double.POSITIVE_INFINITY;
    for(int i = 0; i < m_numSensors; i++){
        //Make sure some packets were received.
        //if(m_sensors[i].m_receivedPackets.isEmpty()) continue;
        if((lastStoredIndex[i] < (m_sensors[i].m_numReceivedPackets - 1)) &&
            (m_sensors[i].m_sampleTimes.get(lastStoredIndex[i] + 1) != null) &&
            (m_sensors[i].m_sampleTimes.get(lastStoredIndex[i] + 1) <= minValue)){
            minValue = m_sensors[i].m_sampleTimes.get(lastStoredIndex[i] + 1);
            minIndex = i;
        }
    }
    if((lastStoredIndex[minIndex] < 0)
        || (m_sensors[minIndex].m_sampleTimes.get(lastStoredIndex[minIndex]) != null)){
        lastStoredIndex[minIndex]++;
    }
    //Log sorted data in CSV format
    try{
        NetFTRDTPacket logPacket =
m_sensors[minIndex].m_receivedPackets.get(lastStoredIndex[minIndex]);
        logFileAppendString = m_sensors[minIndex].m_sampleTimes.get(lastStoredIndex[minIndex]) +
", " + "sensor" + (minIndex + 1)
        + ", " + logPacket.getRDTSequence()
        + ", " + logPacket.getFTSequence()
        + ", " + String.format("%08X",logPacket.getStatus())//display all hex characters always
        + ", " + logPacket.getFx()
        + ", " + logPacket.getFy()
        + ", " + logPacket.getFz()
        + ", " + logPacket.getTx()
        + ", " + logPacket.getTy()
        + ", " + logPacket.getTz()
        + System.lineSeparator();
    }
}

```

Title: Recording Synchronized Data from Multiple Net F/T Sensors	Revision Level: 1	Effective Date: 3/11/13
---	----------------------	----------------------------

```

    } catch(Exception e){
        System.out.println(e);
    }
    try{
        //Which file we write to depends on whether we want
        //one or multiple files
        m_dataLogFileBufferedWriter[(m_separateLogFiles?minIndex:0)].append(logFileAppendString);
    } catch(Exception e){
        setApplicationError("Error writing to log file: " + e.toString());
        return;
    }
    //Test whether there are any samples that must be ordered
    continueInterleaving = false;
    for(int i = 0; i < m_numSensors; i++){
        if(lastStoredIndex[i] < (m_sensors[i].m_numReceivedPackets - 1)) continueInterleaving = true;
    }
}

```


Title: Recording Synchronized Data from Multiple Net F/T Sensors	Revision Level: 1	Effective Date: 3/11/13
---	----------------------	----------------------------

Document Revisions

Revision No.	Summary of Revision	Revised By	Date of Revision
Rev. 01	Initial Issue	David Fleissner	3/11/2013