Low Latency Peak Streaming over UDP On the Hyperion Platform

The Hyperion platform of interrogators has the ability to detect peaks in the reflected response of a fiber sensing channel or network. Historically, these peaks are delivered on demand or as a stream over TCP. TCP is a robust protocol that provides a high level reliability in data transmission, but at some cost in terms of deterministic latency. Other industrial protocols, such as EtherCAT or ProfiNet, are able to provide deterministic low latency while also offering robustness and reliability, but at considerable additional development overhead. These protocols are currently on the roadmap for the Hyperion Platform. As an intermediate step towards real-time implementations of the Hyperion in control applications, the platform now is able to deliver peak streams using UDP, a protocol that can run using much of the same firmware stack as TCP, but with a very simplified set of features that provides more deterministic delivery of data at low latency, but with very little insurance of reliability, especially if implemented on a complex ethernet network. However, on a simple network with, for instance, just a single switch between the interrogator and the system controller that receives the data, UDP can be very reliable. This, then, enables a controller to receive the peak data within a fraction of the time of a single sample that is acquired at 1 kHz, opening up the potential for this platform to be used extensively for control applications. This paper will detail how to use this feature, while also detailing the results from initial testing.

In order to use UDP peak streaming, the system must be set up for peak detection as is detailed in the Hyperion Manual. All configuration and control is performed using the TCP interface. Once the system is configured and is acquiring peak data, the following command can be issued to initiate peak streaming:

```
#EnableUdpPeakDatagrams <client IP address> <client port>
```

The `client IP address` is the IP address of the controller or data acquisition system that is going to be receiving and processing the UDP data, and the `client port` is the port on that controller. The user must ensure that both of these are correct values, and that no firewall is blocking access to that port on the receiving system. As soon as this command is entered, data will begin being sent. In addition, the instrument will put itself in a low-latency mode that turns off many running processes and daemons that could interfere with timely delivery of data. One of these processes is the acquisition and transmission of full spectrum data. While UDP streaming is enabled, no full spectrum data can be acquired, and ENLIGHT will not function. Additionally, the daemons responsible for time synchronization over NTP and PTP are also disabled. We continue to explore ways to enable these functions without interfering with the data delivery, but the current implementation is limited in this regard and is optimized to produce minimal latency.

Disabling the UDP streaming will re-enable all of these processes that were stopped. This is performed with the command:

```
#DisableUdpPeakDatagrams
```

Testing UDP Latency

In order to test the latency and reliability of the UDP data stream, we used a dedicated communications processor. The processor chosen was a Tiva series ARM Cortex-M4 device from TI. This is a 32 bit processor operating at 120 MHz, and includes a 100 Mbps Ethernet peripheral device. Using the LWIP open source stack, we implemented a UDP receiver on an EK-TM4C1294XL evaluation kit (EK) from TI, and connected this directly to the Hyperion instrument. We also connected the SYNC output on the Hyperion to a capture pin on the EK. This SYNC signal emits a high pulse that is aligned to the start of each wavelength sweep on the Hyperion. A running hardware clock on the processor then kept track of the number of ticks between this sync pulse and the receipt of the UDP packet.

The Hyperion acquires data by sweeping the wavelength from low to high over the first half of a sample period (0.5 ms for a 1 kHz instrument). At the end of this 0.5 ms, the data is transmitted in a single UDP datagram.

The figures below show the data from twelve continuous hours of UDP data streaming. This constitutes 43,200,000 samples. No dropped samples were detected during this acquisition. We verify this by observing the sequence of serial numbers that the instrument attaches to each data packet. The plots below provide histograms of the measured latencies. All latencies greater than 1200 ms are reported in the final bin of the histogram. In Figure 1, we see the full range of the data, indicating that there is very tight distribution of latencies detected. In Figure 2 we zoom in on the x-axis and can observe that nearly half of all of the samples have a latency that falls within a single microsecond window with latency of 683 microseconds, and 99.0% arriving in less than 700 microseconds. However, in Figure 3, we see that this distribution has a long tail when zooming in on the y axis.

We still see some latencies that are above 1 ms. In total we see 66 such samples that exceeded 1 ms (note that at the beginning of the acquisition there were 13 spurious measurements as the sample and clock became properly aligned, and the are not included in this number, but they do appear on the histogram below). This amounts to about 1.5 out of every million samples having a delay greater than 1 ms.

The reason for these high latencies is that they Hyperion instrument is not built with a true real time operating system, and it is possible for different pieces of the firmware implementation to interfere and produce small delays. This becomes increasingly harder to detect and correct, as there is no simple way to detect the source of a few millisecond delay that only happens a couple of time per hour. Users that seek to implement this inside of a control loop need to be aware of the possibility of these rare delays occurring. The maximum observed latency was 2.38 ms, and values close to this (2.34 to 2.38) are repeated regularly at a rate of a few times per hour. These repeated values indicate a common source within the operating system, but we have not yet been able to detect the nature of this source of delay.

Our original target performance for this UDP implementation was to have %99.99 of all samples received within the 1 ms window, and these measurements verify that we have achieved this target. Such low latency is impossible with a TCP implementation, and so this UDP stream offers the best possible performance for real time fiber optic sensing applications.

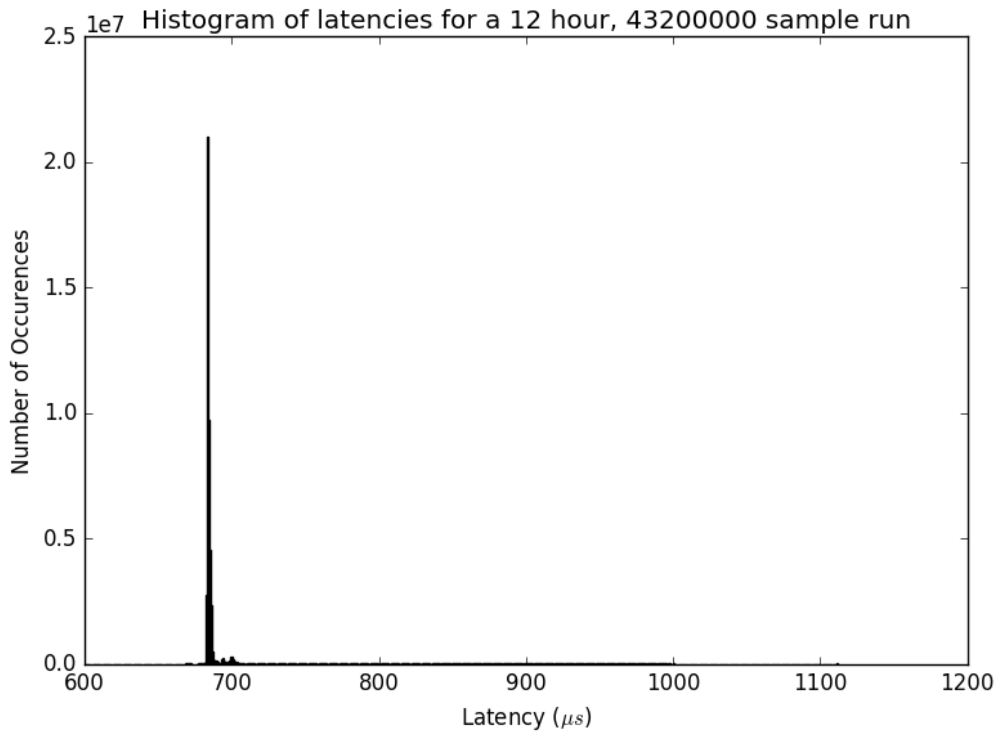

Figure 1

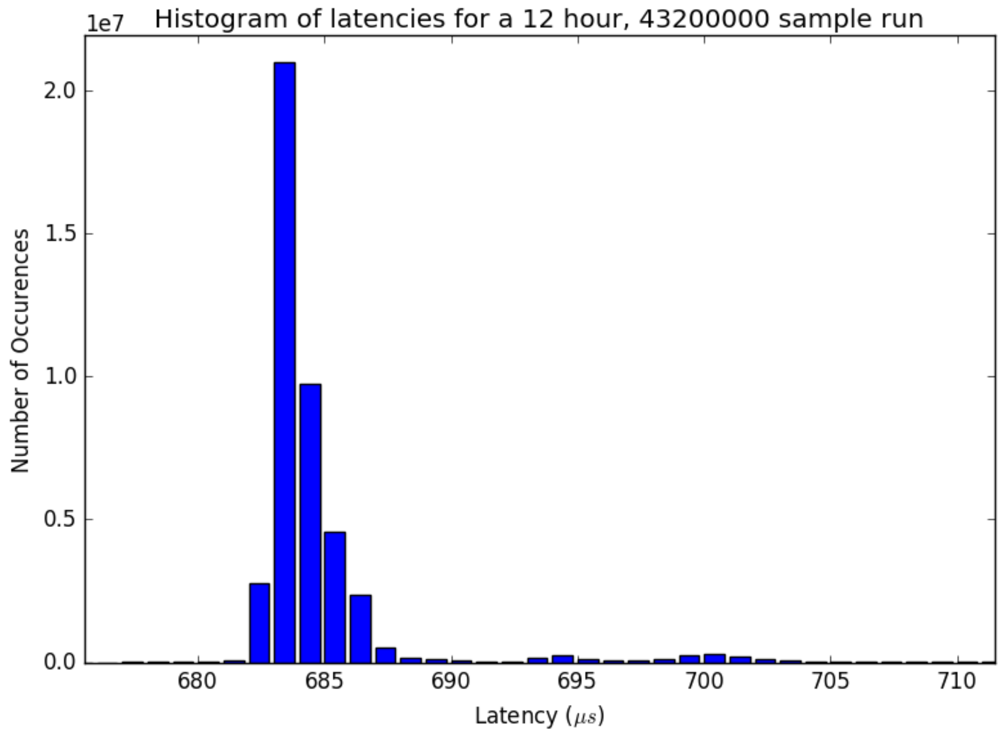

Figure 2

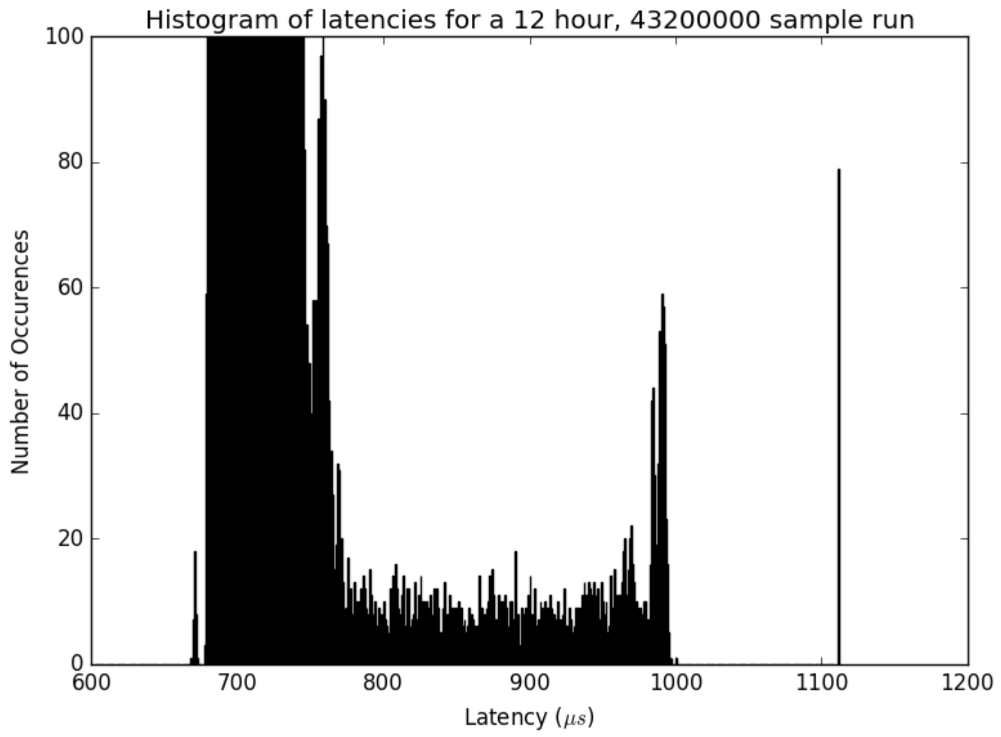

Figure 3