May 28, 2019

AN OPTIMIZED PRODUCER AND CONSUMER DESIGN PATTERN TO CONTROL DATA CONGESTION

HP INC

An Optimized Producer and Consumer Design pattern to control data congestion

Exponential growth in computer networks has resulted in a tremendous amount of traffic variability in time scales ranging from a few milliseconds to several hours. This leads to network traffic congestion which, if persistent, could lead to significant data losses. Congestion losses cannot be avoided by modest increases in buffer capacity, it requires far more creative solutions. Missing data points can result in misleading inferences which can, in turn, lead to wrong engineering and/or business decisions. It is therefore important to identify ways to avoid loss of data packets at the receiving end and thereby ensuring data integrity. Currently available solutions are based on altering the source transmission protocol which, as in the case of printers, may not always be possible. We are proposing a solution, that eliminates the need to alter the source in any way, using a multilevel Producer/Consumer with distributed buffering to consume UDP/TCP packets. Our solution focuses on making changes at the receiver or destination to handle congestion in transmission of data at a fixed rate.

Problem statement:

Commercial devices, such as 3D Printers & web presses have a large number of sensors for measuring pressure, acceleration, displacement, humidity, temperature etc. and servos (moving parts, motors etc.) which need to be monitored remotely. These sensors can emit data in a frequency ranging from 2Hz-10Hz. Handling data coming at such high frequencies over the network often leads to network congestion. This in turn results in a loss of data packets at the receiving end. Loss of data packets would negatively impact the accuracy and reliability of the inferences drawn from the dataset and hence it is important to eliminate or minimize data loss.

Data loss may occur at two locations.

1. loss at source (printer)

Data loss at the source can be identified by looking for missing data points for one or more sensors while all other sensors show a complete set of data points. In the above image one of the sensors shows data loss at three different intervals. Such data losses need to be address at the source. Figure1 highlights the gap in sensor data due to loss at the source.

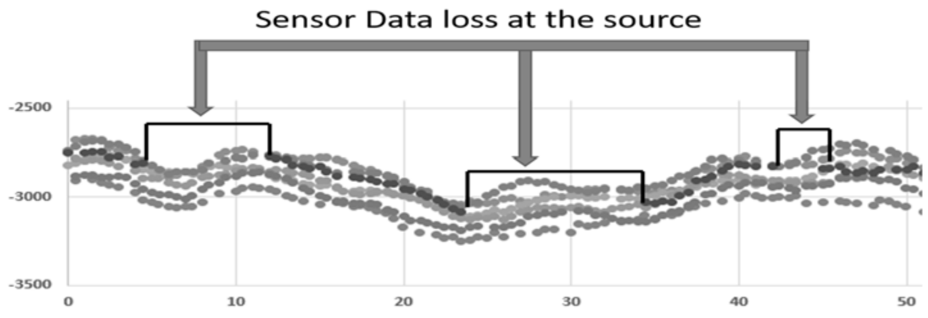

Figure1

2. loss at the destination (receiving end).

Loss of UDP data at the receiver shows a different characteristic representation than the loss at source. The data loss is clearly observed by distinct gaps in data-points across all sensors at a given instance. At the receiving end, data is interleaved with information of multiple sensors. The loss of data at the receiving end can be identified as missing chunks from the UDP packet index. Figure2 highlights the gap in sensor data due to transmission losses when received at the destination without our solution.

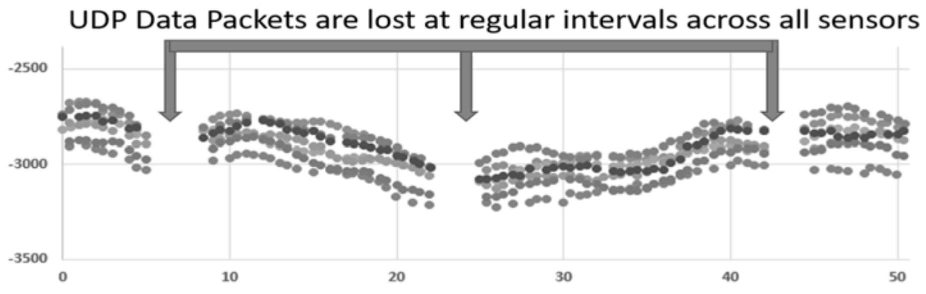

Figure2

Our solution focuses on minimizing the loss at the receiving end or destination.

Past Approaches:

In contrast to TCP, UDP is an unreliable protocol with no mechanism for data reliability or congestion control. UDP based data transfer congestion is handled by increasing round trip time (RTT). RTT is the time difference between sending a packet and receiving its acknowledgement from destination. Usually data sources adjust their packet rate depending on RTT. Alternatively, some solutions have multiple producers and consumers writing data to the same queue which is still an inefficient process as the size of queue is limited and not capable of handling heavy data inflow. Altering the rate of data at source using RTT is not feasible. Our solution focuses on a multilevel Producer Consumer with distributed Buffers design pattern to overcome the issue of data reliability and network congestion at the destination.

Our Solution:

The proposed idea uses multilevel Producer Consumer with distributed Buffers for minimizing data loss.

We are using a multi-level producer-consumer with distributed buffers design pattern to handle Network Congestion and avoid loss of data packets. Our goal is to create a versatile system that can handle data inflow at very high rates from various systems. We are illustrating our solution using a printer as an example.

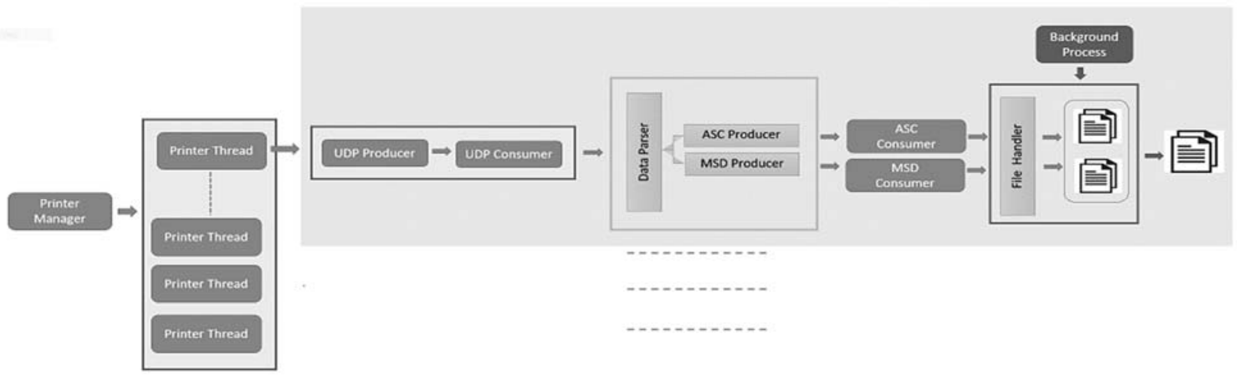

Figure3

Figure3 is a complete end to end block diagram depicting our solution at the destination. Figure4 is a representation of data flow for one printer and this can be extrapolated for multiple printers. A Print manager handles printer connectivity. For each connected printer, we create a printer thread. This printer thread is responsible for handling UDP stream data from that printer. Printer thread initializes UDP Producer which starts data collection from the printer on UDP. Data packets from UDP stream are dumped into a UDP queue continuously. A UDP Consumer is the first to consume these packets from the UDP queue and acts as the gatekeeper to avoid network data loss. UDP Consumer feeds the individual packets from the queue to a Dynamic data parser. The Data parser has a collection of parsing logic, each specific for a type of device. These parsers classify data into different buckets. Data in each bucket is put into individual queues. A consumer is initiated for each queue which fetches and processes data from it. Currently, we are classifying data into two categories namely 'MSD' and 'ASC'.

This approach allows us to free up UDP queue congestion and process data independently based on the parser type. Practically each parser queue has a size limit and in case there is a data overflow at any parser queue, we apply overflow correction based on preset priorities. In extreme cases, this condition based dequeuing prevents unintended data loss for analysis and predictability.

All data processing/refining e.g. conversion of units, checking of values out of range, scaling etc. are done at this level. After all these operations, the data is ready to be written in respective files. To reduce the number of I/O operations, we write data in respective csv's at specific checkpoints keeping memory in check. The checkpoint can be a simple timer, for example write after every second or it can be a calculated field such as write to a file when there are 3000 data points.

Complex systems such as 3D printers can spit data at a very high rate. Our proposed solution will enable capturing and parsing the various data points without any lag. The major advantage of using this approach is that it provides ample computing bandwidth to process every data-point and segregate data as per parsing logic. Systems with lower computational capability can avoid data congestion using this solution. Since the solution operates on a multiple producer and consumer approach, it can also handle multiple sources simultaneously.

Evidence the solution works

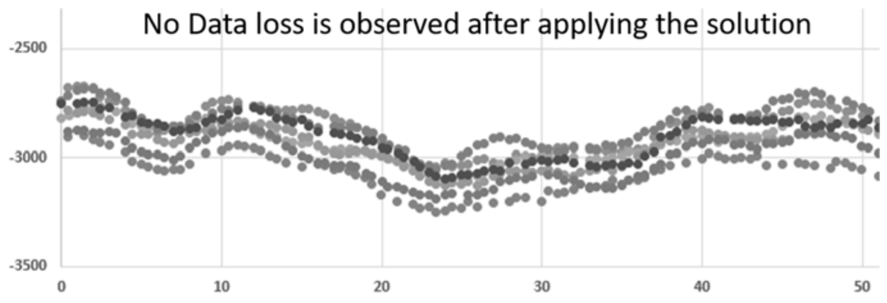

Figure4

We were able to successfully minimize data loss at the destination with the above-mentioned approach. As seen in the graph above, there are no missing datapoints for any of the sensors.

Disclosed by Damini Soni, Kumar Surni and Kumaresh Govindan, HP Inc.