

1. Condiciones de las Pruebas

En esta sección se describen las condiciones bajo las cuales Las mediciones se realizan en un entorno controlado utilizando tres métodos: HTTP + Apps Script, TCP + Node-RED, y UDP + Node-RED. Las mediciones se realizan en un entorno controlado, y se detallan a continuación los parámetros relevantes:

Dispositivo SDR: HackRF One con un tono piloto de 1420 MHz y un ancho de banda de 5 MHz.

Longitud de ventana: Se utilizaron diferentes longitudes de ventana: 256, 512, 1024, 2048, 4096, y 8192 muestras.

Cadencia de ventanas: La cadencia fue de 2 ventanas por segundo (0.5 s) para los dos métodos principales.

Entorno de prueba: El sistema operativo en el emisor fue Windows 10 Pro con GNU Radio 3.10.8.0 y Python 3.11. El receptor operaba dentro de un contenedor Docker con Node-RED en Microsoft Azure.

Tabla 1

Condiciones de las pruebas y parámetros de configuración del sistema.

Parámetro	Valor / Descripción
Dispositivo SDR	HackRF One, tono piloto 1420 MHz, BW = 2 MHz
Longitud de ventana	256, 512, 1024, 2048, 4096 y 8192 muestras
Cadencia fijada	2 ventanas/s (0.5 s), aplicada a los 2

métodos	
Sistema operativo (emisor)	Windows 10 Pro (64 bits), Radioconda 2024.05.29 con GNU Radio 3.10.8.0 y Python 3.11, Python 3.13.2 del sistema.
Host GNU Radio	AMD A6-3400M APU Radeon(tm) HD Graphics 1.40 GHz, 8 GB RAM, Windows 10 Pro
Sistema receptor	Node-RED vX.Y.Z (contenedor Docker nodered/node-red) en Microsoft Azure (1 vCPU, 1.5 GiB RAM), zona horaria America/Bogota, expuesto en puertos 80, 1880, 12400 y 52001.
Contenedor Node-RED (TCP/UDP)	Azure B1s • 1 vCPU • 1 GB RAM
Google Apps Script	Plan gratuito (90 ejec./min, 6 min CPU/día)
Nº de usuarios simultáneos	1 • 5 • 10 navegadores Chrome
Definición de latencia	$\Delta = t_1 - t_0$ (ms); t_0 en GNU Radio, t_1 en Node-RED

Nota. En esta sección, se detallan las condiciones bajo las cuales se llevaron a cabo las mediciones, incluyendo las especificaciones del dispositivo SDR, la longitud de ventana utilizada, las configuraciones de los servidores en Azure y Google Apps Script, y las métricas de latencia consideradas. Este contexto es importante para comprender las mediciones y los resultados que se presentan en los apartados posteriores.

2. Medición de latencia entre GNU Radio y Node-RED (Sistema SIRET y STERT)

Para asegurar una experiencia de visualización espectral en tiempo real, se mide el retraso existente entre el instante en que GNU Radio genera una ventana de espectro y el momento en que ésta es recibida y procesada en Node-RED. En el Sistema SIRET (Sistema Interactivo de Registro y Espectro en Tiempo real), este procedimiento se lleva a cabo de la siguiente manera:

En primer lugar, el bloque UDPFlagger y/o TCPFlagger acumula exactamente N muestras de tipo float32, inserta las banderas de inicio y fin de espectro (FS y FE), captura el instante de envío t_0 en milisegundos y lo divide en dos componentes (hi/lo), añade el tamaño de ventana N y los metadatos fijos (latitud, longitud, salto, azimut, elevación, altitud y descripción), y empaqueta todo como un único arreglo de float32 para su transmisión por UDP o TCP.

A continuación, en Node-RED un nodo Listener (UDP/TCP) recibe el paquete, lo convierte a un arreglo de tipo Float32Array, localiza la bandera de inicio de metadatos (MS), reconstruye el valor de t_0 a partir de los componentes hi y lo, captura el instante de llegada t_1 mediante la función `Date.now()`.

2.1. Definición de Latencia

La latencia se define como la diferencia entre el instante de envío t_0 en GNU Radio y el instante de llegada t_1 en Node-RED, representando la latencia total del proceso. La latencia se calcula de acuerdo con la fórmula:

$$\Delta = \text{Latencia} = t_1 - t_0 \text{ (en milisegundos)}$$

Este valor se registró simultáneamente en una tabla histórica, se mostró en un indicador tipo gauge y se incorporó en un panel de estadísticas (latencia mínima, máxima, media, jitter, etc.).

Los resultados de esta medición se presentan en el Capítulo 6 donde se comparan las latencias obtenidas para ambos protocolos bajo diferentes tamaños de ventana N.

3. Reloj Unix y NTP

El reloj Unix, o Epoch Time, es una medida estándar de tiempo que cuenta los segundos transcurridos desde las 00:00:00 UTC del 1 de enero de 1970. Se utiliza de manera extensiva en sistemas POSIX, protocolos de red y aplicaciones distribuidas para sincronizar eventos y registrar datos con una referencia temporal unificada (Unix / Epoch Timestamp Conversion Tools, 2025). En el sistema STERT (GNU Radio), Unix Time proporciona la base para asignar timestamps de envío a cada ventana de espectro.

El valor típico de Unix Time se almacena en 32 bits con signo y alcanza su límite el 19 de enero de 2038 a las 03:14:07 UTC (el “problema del año 2038”) (S32 De Unix, 2025).

Este problema podría afectar la precisión del tiempo si no se toman medidas para usar representaciones más amplias, como el formato de 64 bits, que es cada vez más común en los sistemas modernos. para ver cómo se ve el reloj Unix entre al anexo A.

3.1 Relación con la Medición de Latencia

En STERT, los bloques UDPFlagger y TCPFlagger capturan un timestamp absoluto t_0 en milisegundos antes de enviar cada ventana de espectro. Ese instante se divide en dos componentes de 32 bits (parte alta y parte baja) para preservar precisión. En el sistema **SIRET** (Node-RED), el paquete se recibe, se reconstruye t_0 , se captura el instante de llegada t_1 con la función `Date.now()` y se calcula la latencia como

$$Latencia = t_1 - t_0.$$

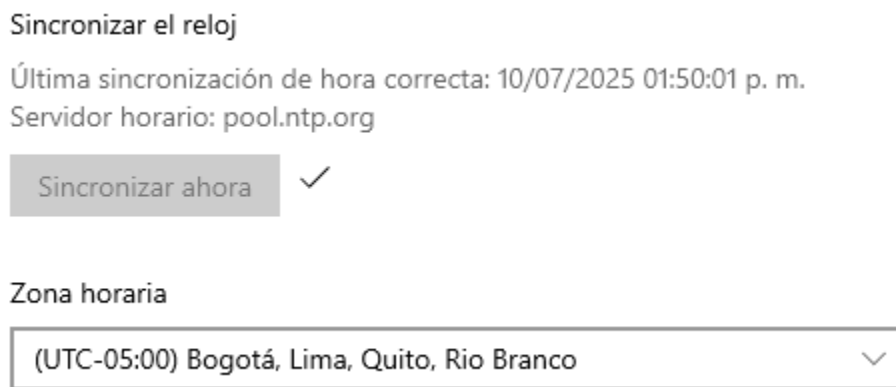
De este modo, ambas plataformas comparten la misma referencia temporal: el Unix Epoch.

3.2 Sincronización Mediante NTP

Para asegurar que los relojes de STERT y SIRET permanezcan alineados, el sistema utiliza el Network Time Protocol (NTP). Se configura el cliente de NTP en ambos entornos para sincronizarse periódicamente con servidores confiables, por ejemplo, pool.ntp.org, garantizando que emisor y receptor compartan un tiempo de referencia común y minimizando las discrepancias en la medición de latencia.

Figura 1

Interfaz de sincronización del reloj con servidor NTP.



Sincronizar el reloj

Última sincronización de hora correcta: 10/07/2025 01:50:01 p. m.

Servidor horario: pool.ntp.org

Sincronizar ahora ✓

Zona horaria

(UTC-05:00) Bogotá, Lima, Quito, Rio Branco ▼

Nota. La sincronización automática del reloj mediante NTP (como pool.ntp.org) asegura que todos los relojes de GNU Radio y Node-RED utilicen la misma referencia temporal, reduciendo los errores de sincronización y mejorando la precisión en la medición de latencia.

3.3. Sincronización de Relojes Entre el Puesto de Pruebas y el Contenedor en Azure

Para garantizar que las mediciones de latencia fueran confiables y comparables, se verificó y ajustó la hora del computador local respecto al reloj del contenedor Node-RED que corre en Azure Container Instances. El procedimiento fue:

Abrir una consola interactiva (“Conectar”) sobre el contenedor nodered (imagen *nodered/node-red*).

Ejecutar el comando `date` para leer la hora oficial del servidor.

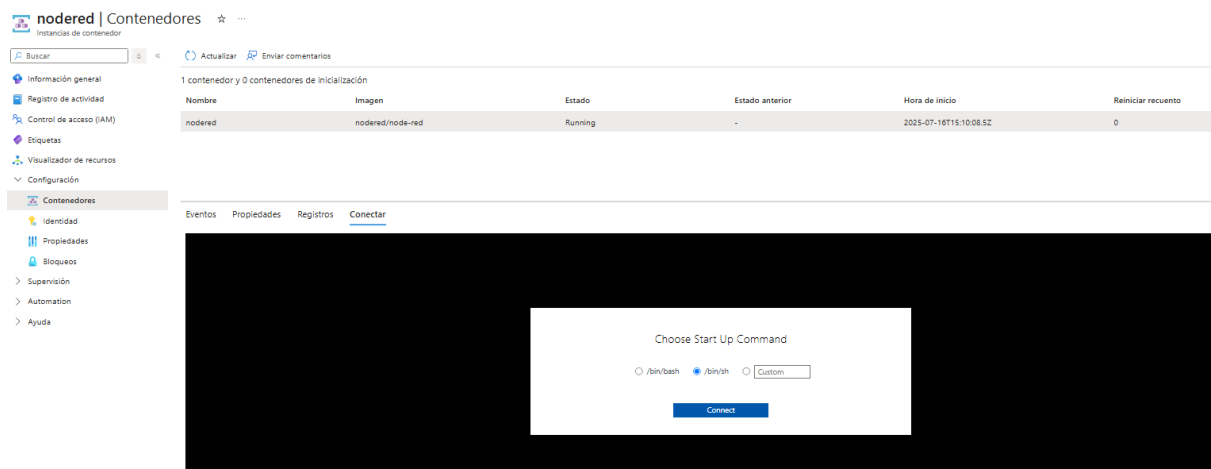
Comparar ese valor con la hora del sistema operativo Windows; cuando el desfase excedía ± 1 s, se forzaba la sincronización NTP (“Sincronizar ahora” en Configuración → Hora e idioma).

Repetir la comprobación antes de cada bloque de mediciones ($N = 256 \rightarrow 4096$).

Con este ajuste sistemático se evitó que diferencias de minutos entre relojes generaran latencias negativas o artificialmente altas, y se garantizó que los timestamps $t0_ms$ (GNU Radio) y $t1_ms$ (Node-RED) se compararan dentro de la misma referencia temporal.

Figura 2

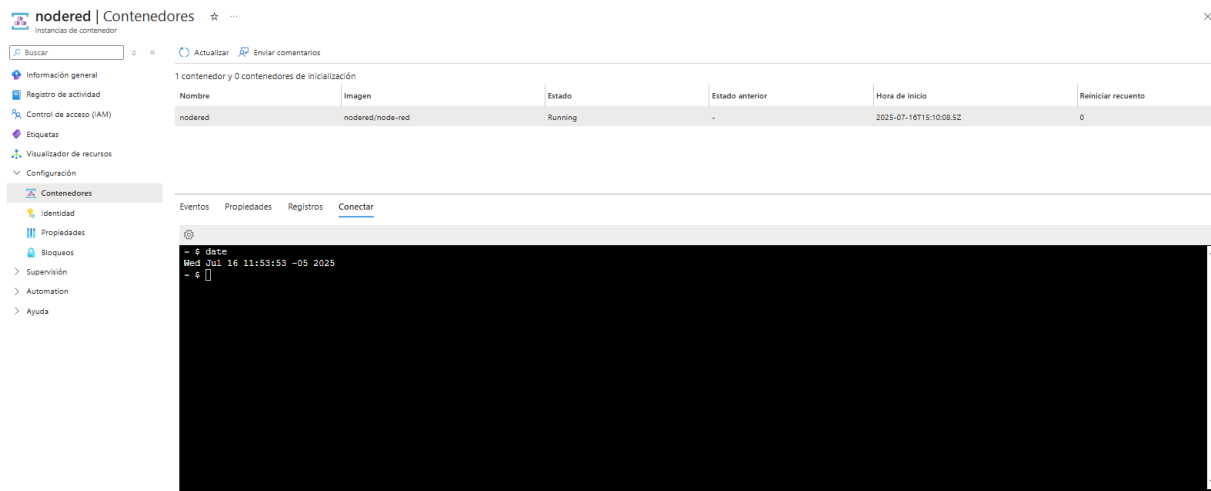
Conexión al contenedor Node-RED en Azure para verificación de hora del sistema.



Nota. La ventana de diálogo permite seleccionar el intérprete y abrir un shell bash sin detener la aplicación, brindando acceso a diagnósticos en tiempo real.

Figura 3

Resultado del comando date dentro del contenedor, confirmando la hora oficial del servidor (UTC-5).



Nota. La hora reportada se contrastó con la del equipo local; tras la sincronización NTP el desfase quedó por debajo de 1 s, requisito clave para la validez de las mediciones de latencia.

4. Generación de Paquetes en GNU Radio

En STERT (GNU Radio) se emplean dos bloques principales, UDPFlagger y TCPFlagger, para construir y transmitir cada ventana espectral junto con los metadatos necesarios. Ambos bloques realizan las siguientes operaciones:

Acumulan N muestras de espectro de tipo *float32* en un buffer interno

Insertan las banderas de inicio y fin de espectro ($FS = -1,0$ y $FE = -2,0$) para delimitar el contenido de la ventana

Capturan el timestamp de envío t_0 en milisegundos y lo dividen en dos componentes de 32 bits (hi/lo) para preservar la resolución temporal

Insertan las banderas de inicio y fin de metadatos ($MS = -3,0$ y $ME = -4,0$), añadiendo en UDPFlagger el tamaño de ventana N y en ambos los metadatos fijos de geolocalización y descripción

Empaquetan todo el contenido como un único arreglo de *float32* y lo envían por UDP o TCP hacia SIRET (Node-RED).

4.1 UDPFlagger

El bloque UDPFlagger realiza las siguientes operaciones para preparar cada ventana espectral y enviarla por UDP:

Acumula *N* muestras de tipo *float32* en un buffer interno

Inserta las banderas de inicio y fin de espectro (FS = -1,0 y FE = -2,0)

Inserta la bandera MS = -3,0 al comienzo y ME = -4,0 al final del bloque de metadatos

Captura el timestamp de envío *t0* en milisegundos y lo divide en dos componentes (*hi* / *lo*)

Construye el bloque de metadatos con [MS, *hi*, *lo*, *N*], los valores de *meta_fixed* y la bandera ME

Empaqueta el espectro y los metadatos en un único arreglo de *float32* y lo envía por UDP hacia SIRET.

Figura 4

División del timestamp y construcción del bloque de metadatos en UDPFlagger.

```

100 | | | | # 7.3) Capturar timestamp actual en ms y dividir en hi/lo
101 | | | | t0_ms = time.time() * 1000.0 # Tiempo en ms (float64)
102 | | | | hi, lo = self._split(t0_ms) # Partes hi/lo en float32
103 | | | |
104 | | | | # 7.4) Construir bloque de metadatos:
105 | | | | # [MS, hi, lo, N] + meta_fixed + [ME]
106 | | | | # Insertamos self.N para que Node-RED lo reciba
107 | | | | meta = np.concatenate([
108 | | | |     np.array([self.MS, hi, lo, np.float32(self.N)], dtype=np.float32),
109 | | | |     self.meta_fixed,
110 | | | |     np.array([self.ME], dtype=np.float32)
111 | | | | ], dtype=np.float32)
112 | | | |
113 | | | | # 7.5) Empaquetado final: espectro + metadatos
114 | | | | pkt = np.concatenate([spec, meta], dtype=np.float32)
115 | | | | n = len(pkt) # Número total de floats
116 | | | |

```

Nota. Las banderas MS y ME permiten a Node-RED localizar con precisión el inicio y fin del bloque de metadatos para extraer correctamente el timestamp, el tamaño de ventana N y la información de geolocalización.

4.2 TCPFlagger

El bloque TCPFlagger realiza las siguientes operaciones para preparar cada ventana espectral y enviarla por TCP:

Acumula N muestras de tipo *float32* en un buffer interno

Inserta las banderas de inicio y fin de espectro ($FS = -1,0$ y $FE = -2,0$)

Inserta la bandera $MS = -3,0$ al comienzo y $ME = -4,0$ al final del bloque de metadatos

Captura el timestamp de envío t_0 en milisegundos y lo divide en dos componentes (hi / lo)

Construye el bloque de metadatos con MS,hi,lo,N, los valores de *meta_fixed* (latitud, longitud, salto, azimut, elevación, altitud y descripción) y la bandera ME

Empaqueta el espectro y los metadatos en un único arreglo de *float32* y lo envía por TCP hacia SIRET.

Figura 5

División del timestamp y construcción del bloque de metadatos en TCPFlagger.

```

99
100
101 # 7.3) Capturar timestamp actual en ms y dividir en hi/lo
102 t0_ms = time.time() * 1000.0 # Tiempo en ms (float64)
103 hi, lo = self._split(t0_ms) # Partes hi/lo en float32
104
105 # 7.4) Construir bloque de metadatos:
106 # [MS, hi, lo, N] + meta_fixed + [ME]
107 # Insertamos self.N para que Node-RED lo reciba
108 meta = np.concatenate([
109     np.array([self.MS, hi, lo, np.float32(self.N)], dtype=np.float32),
110     self.meta_fixed,
111     np.array([self.ME], dtype=np.float32)
112 ], dtype=np.float32)
113
114 # 7.5) Empaquetado final: espectro + metadatos
115 pkt = np.concatenate([spec, meta], dtype=np.float32)
116 n = len(pkt) # Número total de floats

```

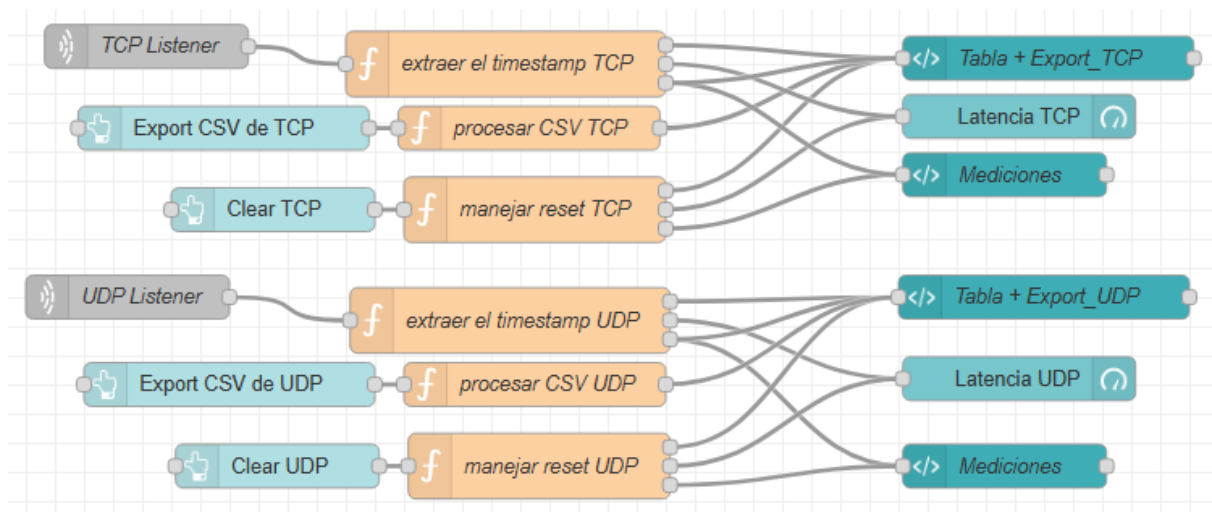
Nota. Este fragmento muestra cómo TCPFlagger captura el instante de envío, divide el timestamp en dos componentes, inserta el tamaño de ventana **N** dentro de los metadatos y empaqueta el bloque espectral junto a los metadatos antes de transmitirlo por TCP.

5. Configuración de Node-RED (backend)

Este apartado describe la puesta en marcha de los flujos de Node-RED que reciben las ventanas espectrales por UDP y TCP, extraen los timestamps y **N**, calculan la latencia y alimentan el dashboard en SIRET.

Figura 6

Flujo de procesamiento en Node-RED para medición de latencia vía UDP y TCP.



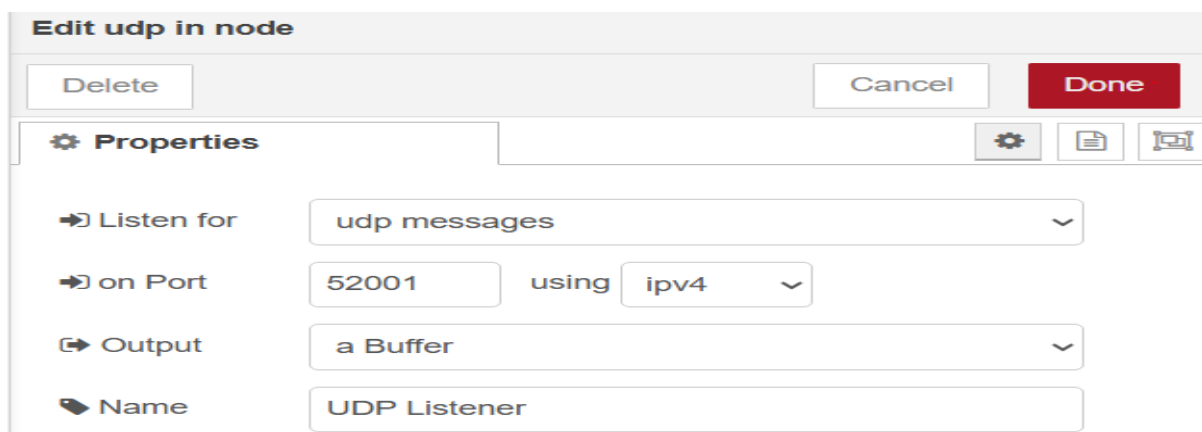
Nota. El diagrama muestra cómo los nodos TCP Listener y UDP Listener reciben los paquetes de GNU Radio, la función “extraer el timestamp” reconstruye t_0 y calcula latencia, la función “procesar CSV” prepara los datos para descarga, la función “manejar reset” limpia el historial, y las salidas alimentan la Tabla + Export_TCP/UDP, el gauge de latencia y el panel de Mediciones.

5.1 UDP Listener (“SIRET-UDP”)

El nodo UDP Listener se configura en el puerto 52001 para recibir los paquetes enviados por STERT. A continuación, pasa el payload al primer Function node (extraer_timestamp_UDP) que formatea los datos y calcula la latencia.

Figura 7

Configuración del nodo UDP Listener (“SIRET-UDP”) en Node-RED.



Nota. La imagen muestra cómo el nodo “UDP Listener” está configurado para “Listen for: udp messages” en el puerto 52001, usando IPv4, con salida “a Buffer” y nombre “UDP Listener”, lo cual permite recibir directamente los paquetes binarios enviados por STERT.

5.2 Función extraer_timestamp_UDP

Este nodo Function convierte el Buffer entrante en un arreglo de floats, detecta la bandera MS (-3,0), reconstruye el timestamp de envío t_0 , lee el tamaño de ventana N y calcula la latencia como $t_1 - t_0$. Emite tres salidas:

Registro {seq, t_0 , t_1 , latency, N} para la tabla “Tabla + Export_UDP”

Valor numérico de latencia (ms) para el gauge “Latencia UDP”

Objeto de estadísticas acumuladas para el panel “Mediciones”

Figura 8

Detalle de la función extraer timestamp UDP en Node-RED parte A.

```

1  /**
2  * Function: extraerTimestampYStats UDP (Modificado para leer N desde metadatos)
3  * - Procesa los datos recibidos por UDP desde GNU Radio
4  * - Extrae ventana (N), timestamp, calcula latencia, jitter, estadísticas y pérdidas
5  * - Salidas:
6  *   [0] → registro para Tabla + Export_UDP
7  *   [1] → latencia para gauge
8  *   [2] → estadísticas para ui_template (Mediciones)
9  */
10
11 // -----
12 // 0) Acumular bytes crudos en contexto
13 // -----
14 const chunk      = msg.payload; // Buffer entrante
15 let bufBytesUDP  = flow.get('bufBytesUDP') || Buffer.alloc(0);
16 bufBytesUDP      = Buffer.concat([bufBytesUDP, chunk]); // Añade nuevos bytes
17 // Sólo procesamos múltiplos de 4 bytes (float32)
18 const totalB     = bufBytesUDP.length - (bufBytesUDP.length % 4);
19 const bufRead    = bufBytesUDP.slice(0, totalB); // Parte válida
20 const bufLeft    = bufBytesUDP.slice(totalB); // Resto pendiente
21 flow.set('bufBytesUDP', bufLeft); // Guarda sobrante
22
23 // -----
24 // 1) Convertir bytes a floats y acumular en contexto
25 // -----
26 const f32        = new Float32Array(bufRead.buffer, bufRead.byteOffset, bufRead.length / 4);
27 let floatsUDP    = (flow.get('bufFloatsUDP') || []).concat(Array.from(f32));
28
29 // -----
30 // 2) Incrementar contador de secuencia UDP (ventana)
31 // -----
32 let seqUDP = flow.get('seq_udp') || 0;
33 seqUDP += 1;
34 flow.set('seq_udp', seqUDP);
35
36 // -----
37 // 3) Buscar bandera de metadatos (-3.0) para extraer timestamp y N
38 // -----
39 for (let i = 0; i < floatsUDP.length - 3; i++) {
40   if (floatsUDP[i] === -3.0) {
41     // 3.1) Reconstruir timestamp de envío (t0) en ms
42     const hi = floatsUDP[i + 1]; // Parte alta (float32)
43     const lo = floatsUDP[i + 2]; // Parte baja (float32)
44     const t0 = hi * 65536 + lo; // Timestamp envío
45
46     // 3.2) Leer tamaño de ventana (N) desde metadatos
47     const N_window = Math.round(floatsUDP[i + 3]); // N recuperado
48     flow.set('N_UDP', N_window); // Guarda N en contexto
49
50     // 3.3) Capturar timestamp de llegada (t1)
51     const t1 = Date.now(); // Timestamp llegada
52     const latency = t1 - t0; // Latencia en ms
53
54     // -----
55     // 4) Construir registro para la tabla
56     // -----
57     const rec = {
58       seq: seqUDP, // Número de ventana
59       t0: t0, // Envío
60       t1: t1, // Llegada
61       latency: latency, // Latencia ms
62       N: N_window // Tamaño de ventana
63     };
64   }
65 }

```

Nota. Imagen correspondiente a la configuración del sistema parte A.

Figura 9

Detalle de la función extraer timestamp UDP en Node-RED parte B.

```

66 // -----
67 // 5) Guardar registro en historial
68 // -----
69 let hist = flow.get('hist_udp') || [];
70 hist.unshift(rec); // Inserta al frente
71 flow.set('hist_udp', hist);
72
73 // -----
74 // 6) Calcular estadísticas acumuladas
75 // -----
76
77 // 6.1) Conteo de paquetes perdidos
78 let prevSeq = flow.get('prevSeqUDP');
79 let lostCount = flow.get('lostCountUDP') || 0;
80 if (prevSeq !== undefined && seqUDP - prevSeq > 1) {
81   lostCount += (seqUDP - prevSeq - 1); // Huecos en secuencia
82 }
83 flow.set('prevSeqUDP', seqUDP);
84 flow.set('lostCountUDP', lostCount);
85
86 // 6.2) Estadísticos básicos (min, max, suma, conteo)
87 let minLat = flow.get('minLatUDP') ?? latency;
88 let maxLat = flow.get('maxLatUDP') ?? latency;
89 let sumLat = flow.get('sumLatUDP') ?? 0;
90 let cntLat = flow.get('countLatUDP') ?? 0;
91 minLat = Math.min(minLat, latency);
92 maxLat = Math.max(maxLat, latency);
93 sumLat += latency;
94 cntLat += 1;
95 flow.set('minLatUDP', minLat);
96 flow.set('maxLatUDP', maxLat);
97 flow.set('sumLatUDP', sumLat);
98 flow.set('countLatUDP', cntLat);
99
100 // 6.3) Cálculo de jitter (|Δlat| promedio)
101 let lastLat = flow.get('lastLatUDP');
102 let sumDiff = flow.get('sumDiffUDP') ?? 0;
103 if (lastLat !== undefined) {
104   sumDiff += Math.abs(latency - lastLat);
105 }
106 flow.set('lastLatUDP', latency);
107 flow.set('sumDiffUDP', sumDiff);
108 const jitter = cntLat > 1 ? sumDiff / (cntLat - 1) : 0;
109
110 // 6.4) Percentiles (P50 y P95) sobre últimas 500 muestras
111 let arr = flow.get('arrLatUDP') || [];
112 arr.push(latency);
113 if (arr.length > 500) { arr.shift(); }
114 flow.set('arrLatUDP', arr);
115 const sorted = arr.slice().sort((a, b) => a - b);
116 const pct = p => sorted.length
117   ? sorted[Math.floor((sorted.length - 1) * p / 100)]
118   : 0;
119 const p50 = pct(50);
120 const p95 = pct(95);
121
122 // 6.5) Cálculo de porcentaje de pérdida
123 const totalExp = cntLat + lostCount;
124 const lossPct = totalExp > 0
125   ? (lostCount / totalExp) * 100
126   : 0;
127
128 // -----
129 // 7) Construir mensaje de estadísticas para ui_template
130 // -----
131 ---

```

Nota. Imagen correspondiente a la configuración del sistema parte B.

Figura 10

Detalle de la función extraer timestamp UDP en Node-RED.

```

131     const statsMsg = {
132       stats: {
133         protocol: 'UDP',
134         N:         N_window,
135         min:        minLat,
136         max:        maxLat,
137         avg:        sumLat / cntLat,
138         count:      cntLat,
139         jitter:     jitter,
140         p50:        p50,
141         p95:        p95,
142         lossPct:    lossPct
143       }
144     };
145
146     // -----
147     // 8) Recortar floats usados y guardar el resto
148     // -----
149     floatsUDP = floatsUDP.slice(i + 4);           // Elimina MS, hi, lo, N
150     flow.set('buffFloatsUDP', floatsUDP);
151
152     // -----
153     // 9) Devolver las tres salidas esperadas
154     // -----
155     return [
156       { payload: rec },           // [0] Tabla + Export_UDP
157       { payload: latency },       // [1] Latencia para gauge
158       statsMsg                   // [2] Mediciones (ui_template)
159     ];
160   }
161 }
162
163 // -----
164 // 10) Si no detectamos bandera, guardamos floats sobrantes y no emitimos nada
165 // -----
166 flow.set('buffFloatsUDP', floatsUDP);
167 return null;
168

```

Nota. Este fragmento muestra cómo la función acumula bytes, convierte a floats, detecta la bandera MS, reconstruye el timestamp de envío, extrae el tamaño de ventana N y emite tres salidas para tabla, gauge y estadísticas.

5.3 Export CSV de UDP

Esta función genera el archivo CSV con el historial completo de latencias UDP cuando se pulsa el botón “EXPORT CSV DE UDP”; lee el arreglo `hist_udp` de contexto, construye la cabecera y las filas en formato texto separados por comas, y emite el string resultante al nodo “Tabla + Export_UDP” para ofrecer el enlace de descarga.

Figura 11

Nodo Export CSV de UDP

Edit button node

Delete Cancel Done

Properties

Group [Medidas] Latencia

Size 6 x 1

Icon optional icon

Label Export CSV de UDP

Tooltip optional tooltip

Color optional text/icon color

Background optional background color

When clicked, send:

Payload { "export": true }

Topic msg. topic

→ If msg arrives on input, emulate a button click: ☐

Class Optional CSS class name(s) for widget

Name Name

Nota. Al invocar esta función, el usuario obtiene un archivo CSV con las columnas seq, t0, t1, latency y N de todas las muestras almacenadas en hist_udp.

5.4 Procesar CSV UDP

La función procesar CSV UDP recibe tanto los registros individuales de latencia como el CSV completo generado por la función anterior y actualiza la tabla en el nodo “Tabla + Export_UDP”. Esta función:

Observa msg.payload; si es un string interpreta que es el CSV y construye un enlace de descarga.

Si msg.payload es un objeto con campos de latencia, inserta una nueva fila al inicio de la tabla.

Mantiene el límite de 10 filas visibles, eliminando la fila más antigua cuando se excede.

Figura 12

Función “procesar CSV UDP” y actualización dinámica de la tabla parte A.

```

1  // ► EXPORT CSV UDP
2  if (msg.payload && msg.payload.export) {
3    // 1) Recuperar el historial UDP
4    const histUDP = flow.get('hist_udp') || [];
5
6    // 2) Si no hay datos, avisar
7    if (histUDP.length === 0) {
8      return { payload: "No hay datos para exportar." };
9    }
10
11   // 3) Leer métricas de contexto
12   const minLat = flow.get('minLatUDP') ?? 0;
13   const maxLat = flow.get('maxLatUDP') ?? 0;
14   const sumLat = flow.get('sumLatUDP') ?? 0;
15   const cntLat = flow.get('countLatUDP') ?? 0;
16   const sumDiff = flow.get('sumDiffUDP') ?? 0;
17   const arr = flow.get('arrLatUDP') || [];
18   const lostCount = flow.get('lostCountUDP') || 0;
19
20   // 4) Calcular métricas derivadas
21   const avgLat = cntLat > 0 ? (sumLat / cntLat) : 0;
22   const jitter = cntLat > 1 ? (sumDiff / (cntLat - 1)) : 0;
23   const sorted = arr.slice().sort((a,b)=>a-b);
24   const getP = p => sorted.length
25   ? sorted[Math.floor((sorted.length-1)*p/100)]
26   : 0;
27   const p50 = getP(50);
28   const p95 = getP(95);
29   const lossPct = (cntLat + lostCount) > 0
30   ? (lostCount / (cntLat + lostCount) * 100)
31   : 0;
32
33   // 5) Construir CSV de resumen + datos brutos
34   let csv = '';

```

Nota. Imagen correspondiente a la configuración del sistema parte A.

Figura 13

Función “procesar CSV UDP” y actualización dinámica de la tabla.

```

35 csv += 'PROTOCOLO,Min,Max,Avg,Count,Jitter,P50,P95,Loss\n';
36 csv += `UDP,${minLat.toFixed(2)},${maxLat.toFixed(2)},${avgLat.toFixed(2)},`;
37 csv += `${cntLat},${jitter.toFixed(2)},${p50.toFixed(2)},${p95.toFixed(2)},${lossPct.toFixed(2)}\n\n`;
38 csv += `seq,t0_ms,t1_ms,latency_ms\n`;
39 histUDP.forEach(r => {
40   csv += `${r.seq},${r.t0},${r.t1},${r.latency}\n`;
41 });
42
43 // 6) Devolver CSV
44 return { payload: csv };
45 }
46
47 // ► CLEAR UDP
48 if (msg.payload && msg.payload.clear) {
49   // 1) Limpiar historial
50   flow.set('hist_udp', []);
51   // 2) Reiniciar contador de ventanas
52   flow.set('seq_udp', 1);
53   // 3) (Opcional) Reiniciar métricas de stats
54   flow.set('minLatUDP', null);
55   flow.set('maxLatUDP', null);
56   flow.set('sumLatUDP', 0);
57   flow.set('countLatUDP', 0);
58   flow.set('sumDiffUDP', 0);
59   flow.set('arrLatUDP', []);
60   flow.set('lostCountUDP', 0);
61 }
62
63 return null;
64

```

Nota. Esta función permite que el nodo Tabla + Export_UDP muestre de forma interactiva tanto los datos en tiempo real como el archivo CSV completo, garantizando coherencia entre ambas vistas.

5.5 Tabla + Export_UDP (ui_template)

Esta plantilla de interfaz de usuario presenta la tabla de los últimos registros de latencia UDP y añade dinámicamente un enlace para descargar el CSV generado. La estructura incluye un encabezado con el protocolo y el tamaño de ventana, un cuerpo con hasta 10 filas ordenadas por llegada y un enlace de Descargar CSV UDP que se muestra al recibir el texto completo del archivo CSV.

Figura 14

Nodo Tabla + Export_UDP mostrando la tabla interactiva y el enlace de descarga parte A.

```

1 <style>
2   /* Define el estilo de la tabla: ancho completo, sin espacios entre celdas, y margen inferior */
3   #latTableUDP { width:100%; border-collapse:collapse; margin-bottom:1em; }
4   /* Borde y padding para celdas de encabezado y cuerpo, texto alineado a la derecha */
5   #latTableUDP th, #latTableUDP td { border:1px solid #ccc; padding:4px 8px; text-align:right; }
6   /* Separación superior del área de descarga */
7   #downloadUDP { margin-top:0.5em; }
8 </style>
9
10 <!-- Cabecera dinámica: muestra protocolo y N usando binding de AngularJS -->
11 <div style="font-weight:bold;">
12   Protocolo: {{msg.stats.protocol}} | <!-- Inserta el nombre del protocolo (UDP) -->
13   N: {{msg.stats.N}} <!-- Inserta el tamaño de ventana N -->
14 </div>
15 <hr />
16
17 <!-- Estructura de la tabla donde se insertan las filas -->
18 <table id="latTableUDP">
19   <thead>
20     <tr>
21       <th>Salida de GNU Radio (ms)</th> <!-- Encabezado columna t0 -->
22       <th>Ventana (Número)</th> <!-- Encabezado columna seq -->
23       <th>Llegada a Node-RED (ms)</th> <!-- Encabezado columna t1 -->
24       <th>latencia (ms)</th> <!-- Encabezado columna latency -->
25     </tr>
26   </thead>
27   <tbody></tbody> <!-- Cuerpo vacío que se llenará dinámicamente -->
28 </table>
29

```

Nota. Imagen correspondiente a la configuración del sistema parte A.

Figura 15

Nodo Tabla + Export_UDP mostrando la tabla interactiva y el enlace de descarga parte B.

```

30 <!-- Contenedor donde se mostrará el enlace de descarga CSV -->
31 <div id="downloadUDP"></div>
32
33 <script>
34 (function(scope){
35   // Observa cada mensaje entrante
36   scope.$watch('msg', function(msg){
37     // Si no hay mensaje o payload, no hacer nada
38     if (!msg || msg.payload === undefined) return;
39     const p = msg.payload;
40
41     // Si payload.clear es true, limpiar tabla y enlace de descarga
42     if (p.clear) {
43       document.querySelector('#latTableUDP tbody').innerHTML = ''; // Borra filas
44       document.getElementById('downloadUDP').innerHTML = ''; // Borra enlace
45       return;
46     }
47

```

Nota. Imagen correspondiente a la configuración del sistema parte B.

Figura 16

Nodo Tabla + Export_UDP mostrando la tabla interactiva y el enlace de descarga.

```

48 // Si payload es un string, se interpreta como contenido CSV
49 if (typeof p === 'string') {
50   const blob = new Blob([p], { type: 'text/csv' }); // Crea Blob CSV
51   const url = URL.createObjectURL(blob); // Genera URL temporal
52   // Inserta enlace para descargar CSV
53   document.getElementById('downloadUDP').innerHTML =
54     `<a href="${url}" download="latencia_udp.csv"> Descargar CSV UDP</a>`;
55   return;
56 }
57
58 // Si payload es un objeto, se trata de un registro individual
59 const tbody = document.querySelector('#latTableUDP tbody');
60 const row = tbody.insertRow(0); // Inserta fila al inicio
61 row.insertCell(0).innerText = p.t0; // Columna t0
62 row.insertCell(1).innerText = p.seq; // Columna seq
63 row.insertCell(2).innerText = p.t1; // Columna t1
64 row.insertCell(3).innerText = p.latency; // Columna latency
65
66 // Mantiene un máximo de 10 filas: si hay más, elimina la última
67 while (tbody.rows.length > 10) {
68   tbody.deleteRow(10);
69 }
70 });
71 })(scope);
72 </script>
73

```

Nota. El *Nodo Tabla + Export_UDP* limpia automáticamente la tabla al pulsar “Clear UDP” y responde tanto a los registros JSON como al string CSV, garantizando una experiencia de usuario fluida.

5.6 Clear UDP

El widget Clear UDP permite al usuario reiniciar el flujo de medición de latencia UDP sin necesidad de redeplegar o reiniciar Node-RED. Su configuración es la siguiente:

Group: [Medidas] Latencia

Size: 6 × 1

Label: Clear UDP

When clicked, send: {"reset": true}

Cuando el botón se pulsa, emite el payload {reset: true} al Function node Clear UDP, que a su vez borra hist_udp, reinicia todos los contadores (seq_udp, minLatUDP, etc.) y envía tres mensajes de { clear: true } para limpiar la tabla, el gauge y el panel de mediciones.

Figura 17

Configuración del botón “Clear UDP” en Node-RED.

The image shows the 'Edit button node' configuration window in Node-RED. The window has a title bar 'Edit button node' and three buttons: 'Delete', 'Cancel', and 'Done'. Below the title bar is a 'Properties' section with various settings:

- Group:** [Medidas] Latencia
- Size:** 6 x 1
- Icon:** optional icon
- Label:** Clear UDP
- Tooltip:** optional tooltip
- Color:** optional text/icon color
- Background:** optional background color
- When clicked, send:**
 - Payload:** { "reset": true }
 - Topic:** a_z
- If msg arrives on input, emulate a button click:** ☐
- Class:** Optional CSS class name(s) for widget
- Name:** Name

Nota. El botón está ubicado en el grupo **[Medidas] Latencia**, con etiqueta “Clear UDP”. Al hacer clic envía el payload { "reset": true }, lo que dispara la función **Clear UDP** para limpiar el historial (hist_udp) y restablecer todos los indicadores de latencia UDP.

5.7 Manejar reset UDP

Después de pulsar “Clear UDP”, el nodo **Manejar reset UDP** recibe { reset: true } y ejecuta la lógica para restablecer el estado del flujo sin perder la configuración ni los demás flujos activos. En concreto, hace lo siguiente:

Borra el arreglo hist_udp en contexto (flow.set('hist_udp', [])), eliminando todos los registros previos.

Reinicia los contadores y acumuladores de latencia en contexto (seq_udp, minLatUDP, maxLatUDP, sumLatUDP, countLatUDP, arrLatUDP, sumDiffUDP, lostCountUDP).

Emite tres mensajes { clear: true } para coordinar la limpieza de la tabla, el gauge y el panel de mediciones.

Figura 18

Función “Manejar reset UDP” en Node-RED parte A.

```

1  // manejar reset UDP
2  // Sólo actúa cuando recibe { payload: { reset: true } }
3  if (msg.payload && msg.payload.reset) {
4
5      // 1) Limpiar todo el historial de registros UDP
6      flow.set('hist_udp', []);
7
8      // 2) Reiniciar contador de secuencia (ventana) UDP
9      flow.set('seq_udp', undefined);
10
11     // 3) Limpiar contador de ventanas previas y perdidos
12     flow.set('prevSeqUDP', undefined);
13     flow.set('lostCountUDP', 0);
14
15     // 4) Limpiar estadísticas de latencia previas
16     flow.set('minLatUDP', undefined);
17     flow.set('maxLatUDP', undefined);
18     flow.set('sumLatUDP', 0);
19     flow.set('countLatUDP', 0);
20
21     // 5) Limpiar datos de jitter
22     flow.set('lastLatUDP', undefined);
23     flow.set('sumDiffUDP', 0);
24
25     // 6) Limpiar array de latencias para percentiles
26     flow.set('arrLatUDP', []);
27
28     // — Preparar mensajes de salida —
29
30     // Salida 0 → limpiar tabla UDP
31     // El nodo tabla detecta payload.clear === true
32     const clearMsgUDP = {
33       payload: { clear: true }
34     };
35
36     // Salida 1 → poner gauge de latencia UDP a cero
37     const zeroMsgUDP = {
38       payload: 0
39     };
40

```

Nota. Imagen correspondiente a la configuración del sistema parte A.

Figura 19

Función “Manejar reset UDP” en Node-RED.

```

41 // Salida 2 → resetear ui_template de estadísticas UDP
42 const clearStatsMsgUDP = {
43   stats: {
44     protocol: 'UDP', // Mantiene el protocolo en el encabezado
45     N: flow.get('N_UDP') || 0,
46     min: 0,
47     max: 0,
48     avg: 0,
49     count: 0,
50     jitter: 0,
51     p50: 0,
52     p95: 0,
53     lossPct: 0
54   }
55 };
56
57 // Enviar los tres mensajes en sus salidas correspondientes
58 return [ clearMsgUDP, zeroMsgUDP, clearStatsMsgUDP ];
59 }
60
61 // Si no es un reset, no emitimos nada
62 return null;
63

```

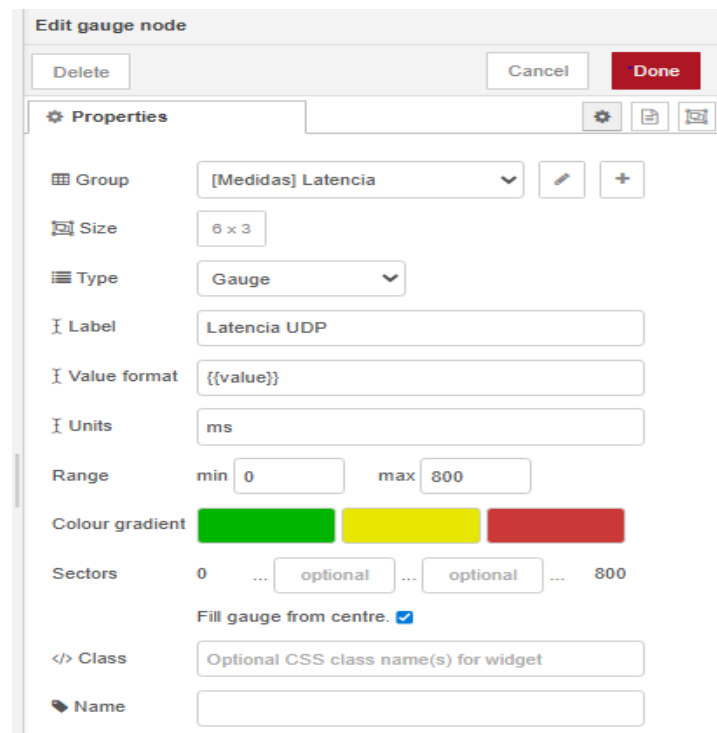
Nota. Este nodo garantiza que el sistema UDP quede listo para una nueva sesión de medición sin necesidad de reiniciar todo Node-RED.

5.8 Gauge Latencia UDP

El nodo Latencia UDP es un widget de tipo *ui_gauge* configurado con rango mínimo 0 ms y máximo 800 ms, que muestra en verde el valor instantáneo de latencia extraído por la función `extraer_timestamp_UDP`. Cada vez que recibe un mensaje con el payload numérico de latencia, actualiza la aguja del indicador.

Figura 20

Widget “Latencia UDP” configurado en Node-RED.



Nota. El widget se ubica en el grupo **[Medidas] Latencia**, con tamaño 6×3 . Está configurado como tipo “Gauge”, etiqueta “Latencia UDP”, formato de valor `{{value}}`, unidades “ms”, rango de 0 a 800 ms, y degradado de color de verde a rojo. Cada mensaje entrante con el valor de latencia actualiza la aguja del indicador.

5.9 Panel “Mediciones” UDP

El nodo **Mediciones UDP** presenta en tiempo real las estadísticas acumuladas de latencia UDP. Incluye un encabezado dinámico con el protocolo y el tamaño de ventana, seguido de los valores de latencia mínima, máxima, promedio, conteo, jitter, percentiles (P50 y P95) y porcentaje de pérdida.

Figura 21

Código del nodo Mediciones UDP.

```

1 <div style="font-family: sans-serif; line-height: 1.4;">
2 <!-- 1) Cabecera dinámica que indica protocolo y ventana -->
3 <div style="font-weight:bold;">
4   Protocolo: {{msg.stats.protocol}} | <!-- "UDP" -->
5   N: {{msg.stats.N}} <!-- Tamaño de ventana -->
6 </div>
7 <hr/>
8
9 <!-- 2) Estadísticas de latencia presentadas en orden -->
10 <div><strong>Min:</strong> {{msg.stats.min.toFixed(2)}} ms</div> <!-- Latencia mínima -->
11 <div><strong>Max:</strong> {{msg.stats.max.toFixed(2)}} ms</div> <!-- Latencia máxima -->
12 <div><strong>Avg:</strong> {{msg.stats.avg.toFixed(2)}} ms</div> <!-- Latencia promedio -->
13 <div><strong>Count:</strong> {{msg.stats.count}}</div> <!-- Número de muestras -->
14 <div><strong>Jitter:</strong> {{msg.stats.jitter.toFixed(2)}} ms</div> <!-- Jitter medio -->
15 <div><strong>P50:</strong> {{msg.stats.p50.toFixed(2)}} ms</div> <!-- Percentil 50 -->
16 <div><strong>P95:</strong> {{msg.stats.p95.toFixed(2)}} ms</div> <!-- Percentil 95 -->
17 <div><strong>Loss:</strong> {{msg.stats.lossPct.toFixed(2)}} %</div> <!-- Porcentaje de pérdida -->
18 </div>
19

```

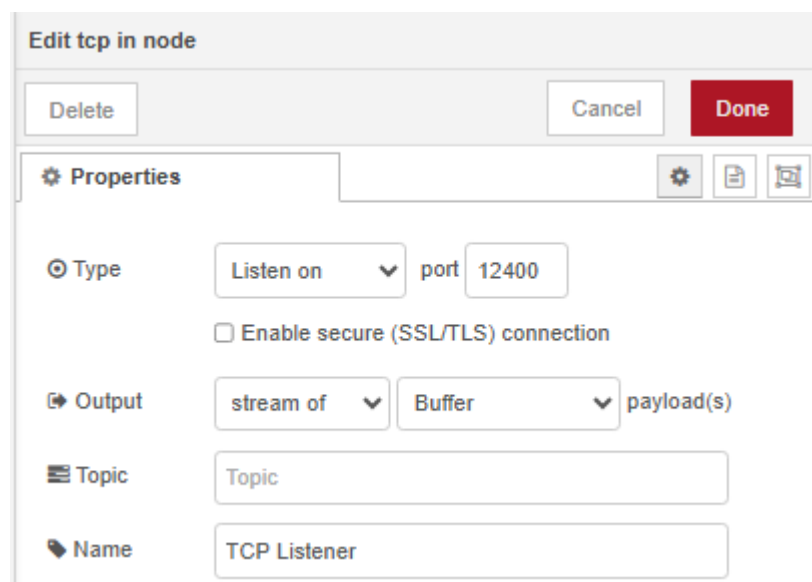
Nota. Este template combina la presentación tabular de métricas UDP con una cabecera dinámica, formatea los valores a dos decimales y mantiene un diseño claro y consistente en el dashboard.

5.10 TCP Listener (“SIRET-TCP”)

El nodo TCP Listener se configura para escuchar en el puerto 12400 y recibe los paquetes enviados por STERT a través de TCP. Su salida es un Buffer que alimenta la función `extraer_timestamp_TCP` para el cálculo de latencia.

Figura 22

Configuración del nodo TCP Listener en Node-RED.



Nota. El nodo está configurado para escuchar en el puerto 12400 mediante TCP, con salida como Buffer, lo que asegura la recepción ordenada de los paquetes espectrales y los metadatos para su posterior procesamiento.

5.11 Función extraer timestamp TCP

El nodo Function extraer timestamp TCP cumple el mismo propósito que su contraparte UDP: convierte el Buffer entrante en un Float32Array, localiza la bandera MS (−3,0), reconstruye el timestamp de envío t_0 , extrae el tamaño de ventana N y calcula la latencia como $t_1 - t_0$. Emite tres salidas:

Registro {seq, t_0 , t_1 , latency, N } para la tabla “Tabla + Export_TCP”

Valor numérico de latencia (ms) para el gauge “Latencia TCP”

Objeto de estadísticas acumuladas para el panel “Mediciones”

Figura 23

Detalle de la función extraer timestamp TCP en Node-RED parte A.

```

1  /**
2   * Nodo: ExtraerTimestampYStats TCP
3   * Salidas:
4   * [0] → { seq, t0, t1, latency, N }
5   * [1] → latency (ms) para gauge
6   * [2] → stats { protocol, N, min, max, avg, count, jitter, p50, p95, lossPct }
7   */
8
9  const chunk          = msg.payload;
10 let bufBytesStats    = flow.get('bufBytesTCPStats') || Buffer.alloc(0);
11 bufBytesStats        = Buffer.concat([bufBytesStats, chunk]);
12
13 // Solo múltiplos de 4 bytes
14 const totalBStats    = bufBytesStats.length - (bufBytesStats.length % 4);
15 const bufReadStats   = bufBytesStats.slice(0, totalBStats);
16 flow.set('bufBytesTCPStats', bufBytesStats.slice(totalBStats));
17

```

Nota. Imagen correspondiente a la configuración del sistema parte A.

Figura 24

Detalle de la función extraer timestamp TCP en Node-RED parte B.

```

18 // Convertir a floats y acumular sobrantes
19 const f32Stats      = new Float32Array(
20   bufReadStats.buffer,
21   bufReadStats.byteOffset,
22   bufReadStats.length / 4
23 );
24 let floatsStats      = (flow.get('bufFloatsTCPStats') || [])
25   .concat(Array.from(f32Stats));
26
27 // Recorrer hasta encontrar MS = -3.0
28 for (let i = 0; i < floatsStats.length - 3; i++) {
29   if (floatsStats[i] === -3.0) {
30     // 1) Reconstruir t0 desde hi/lo
31     const hi    = floatsStats[i + 1];
32     const lo    = floatsStats[i + 2];
33     const t0    = hi * 65536 + lo;
34
35     // 2) Leer N directamente (cuarto elemento tras MS)
36     const N_tcp = Math.round(floatsStats[i + 3]);
37     flow.set('N_TCP', N_tcp);
38
39     // 3) Calcular latencia
40     const t1    = Date.now();
41     const latency = t1 - t0;
42
43     // 4) Secuencia
44     const lastSeq = flow.get('seq_tcp') ?? -1;
45     const seq    = lastSeq + 1;
46     flow.set('seq_tcp', seq);
47
48     // 5) Registro para tabla
49     const rec = { seq, t0, t1, latency, N: N_tcp };
50
51     // 6) Actualizar historial y pérdidas
52     let hist  = flow.get('hist_tcp') || [];
53     hist.unshift(rec);
54     flow.set('hist_tcp', hist);
55
56     let prev  = flow.get('prevSeqTCP');
57     let lost  = flow.get('lostCountTCP') || 0;
58     if (prev !== undefined && seq - prev > 1) lost += (seq - prev - 1);
59     flow.set('prevSeqTCP', seq);
60     flow.set('lostCountTCP', lost);
61
62     // 7) Estadísticos básicos
63     let minLat = flow.get('minLatTCP') ?? latency;
64     let maxLat = flow.get('maxLatTCP') ?? latency;
65     let sumLat = flow.get('sumLatTCP')  || 0;
66     let cntLat = flow.get('countLatTCP') || 0;
67     minLat    = Math.min(minLat, latency);
68     maxLat    = Math.max(maxLat, latency);
69     sumLat    += latency;
70     cntLat    += 1;
71   }
72 }

```

Nota. Imagen correspondiente a la configuración del sistema parte B.

Figura 25

Detalle de la función extraer timestamp TCP en Node-RED parte C.

```

71     flow.set('minLatTCP', minLat);
72     flow.set('maxLatTCP', maxLat);
73     flow.set('sumLatTCP', sumLat);
74     flow.set('countLatTCP', cntLat);
75
76     // 8) Jitter
77     let lastLat = flow.get('lastLatTCP');
78     let sumDiff = flow.get('sumDiffTCP') || 0;
79     if (lastLat !== undefined) sumDiff += Math.abs(latency - lastLat);
80     flow.set('lastLatTCP', latency);
81     flow.set('sumDiffTCP', sumDiff);
82     const jitter = cntLat > 1 ? sumDiff / (cntLat - 1) : 0;
83
84     // 9) Percentiles P50/P95
85     let arr = flow.get('arrLatTCP') || [];
86     arr.push(latency);
87     if (arr.length > 500) arr.shift();
88     flow.set('arrLatTCP', arr);
89     const sorted = arr.slice().sort((a, b) => a - b);
90     const pct = p => sorted.length
91       ? sorted[Math.floor((sorted.length - 1) * p / 100)]
92       : 0;
93     const p50 = pct(50), p95 = pct(95);
94
95     // 10) Pérdida %
96     const totalExp = cntLat + lost;
97     const lossPct = totalExp > 0 ? (lost / totalExp) * 100 : 0;
98
99     // 11) Stats para cabecera
100    const statsMsg = {
101      stats: {
102        protocol: 'TCP',
103        N: N_tcp,
104        min: minLat,
105        max: maxLat,
106        avg: sumLat / cntLat,
107        count: cntLat,
108        jitter: jitter,
109        p50: p50,
110        p95: p95,
111        lossPct: lossPct
112      }
113    };
114
115    // 12) Recortar floats procesados
116    floatsStats = floatsStats.slice(i + 4);
117    flow.set('bufFloatsTCPStats', floatsStats);
118

```

Nota. Imagen correspondiente a la configuración del sistema parte C.

Figura 26

Detalle de la función extraer timestamp TCP en Node-RED.

```

119      // 13) Devolver salidas
120      return [
121        { payload: rec      }, // [0] tabla
122        { payload: latency}, // [1] gauge
123        statsMsg           // [2] header
124      ];
125    }
126  }
127
128  // Si no encontramos MS, guardar sobrantes y no emitir
129  flow.set('bufFloatsTCPStats', floatsStats);
130  return null;
131

```

Nota. La lógica es idéntica a la versión UDP, garantizando consistencia en la extracción de timestamps y tamaño de ventana N, pero aplicándola al flujo TCP para medir la latencia de manera fiable.

5.12 Export CSV de TCP

El nodo Function Export CSV de TCP genera un archivo CSV con el historial completo de latencias TCP cuando recibe la señal correspondiente. Su lógica es la siguiente:

Detecta en msg.payload la señal { export: true } generada por el botón “Export CSV de TCP”.

Recupera el arreglo hist_tcp almacenado en flow.context.

Construye una cadena de texto CSV con encabezados (seq,t0,t1,latency,N) y las filas correspondientes a cada registro.

Emite la cadena CSV al nodo “Tabla + Export_TCP”, que a su vez presenta el enlace de descarga al usuario.

Figura 27

Función “Export CSV de TCP” en Node-RED y plantilla UI de descarga.

Edit button node

Delete Cancel Done

Properties

Group [Medidas] Latencia

Size 6 x 1

Icon optional icon

Label Export CSV de TCP

Tooltip optional tooltip

Color optional text/icon color

Background optional background color

When clicked, send:

Payload { { "export": true } }

Topic msg. topic

If msg arrives on input, emulate a button click: ☐

Class Optional CSS class name(s) for widget

Name Name

Nota. El botón se ubica en el grupo [Medidas] Latencia, con tamaño 6 × 1 y etiqueta “Export CSV de TCP”. Al pulsarlo envía el payload { "export": true }, lo que desencadena la función Export CSV de TCP para generar y mostrar el enlace de descarga del archivo con el historial completo de latencias TCP.

5.13 Procesar CSV TCP

El nodo Function procesar CSV TCP recibe tanto los registros individuales de latencia como el texto CSV generado y actualiza la vista en el nodo “Tabla + Export_TCP”. Su comportamiento es:

Si msg.payload es una cadena (string), la interpreta como CSV y genera un enlace de descarga.

Si msg.payload es un objeto con campos de latencia, inserta una nueva fila al inicio de la tabla.

Mantiene un máximo de 10 filas visibles, eliminando la más antigua cuando se excede ese límite.

Figura 28

Función procesar CSV TCP en Node-RED parte A.

```

1  // ► EXPORT CSV TCP
2  if (msg.payload && msg.payload.export) {
3    // 1) Recuperar el historial TCP
4    const histTCP = flow.get('hist_tcp') || [];
5
6    // 2) Si no hay datos, avisar
7    if (histTCP.length === 0) {
8      return { payload: "No hay datos para exportar." };
9    }
10
11   // 3) Leer métricas de contexto
12   const minLat   = flow.get('minLatTCP') ?? 0;
13   const maxLat   = flow.get('maxLatTCP') ?? 0;
14   const sumLat   = flow.get('sumLatTCP') ?? 0;
15   const cntLat   = flow.get('countLatTCP') ?? 0;
16   const sumDiff  = flow.get('sumDiffTCP') ?? 0;
17   const arr      = flow.get('arrLatTCP')  || [];
18   const lostCount = flow.get('lostCountTCP') || 0;
19
20   // 4) Calcular métricas derivadas
21   const avgLat   = cntLat > 0 ? (sumLat / cntLat) : 0;
22   const jitter   = cntLat > 1 ? (sumDiff / (cntLat - 1)) : 0;
23   // percentiles P50 y P95
24   const sorted   = arr.slice().sort((a,b)=>a-b);
25   const getP     = p => sorted.length
26   | | | | | | | | | | | | | | | | ? sorted[Math.floor((sorted.length-1)*p/100)]
27   | | | | | | | | | | | | | | | | : 0;
28   const p50      = getP(50);
29   const p95      = getP(95);
30   // porcentaje de pérdida
31   const lossPct  = (cntLat + lostCount) > 0
32   | | | | | | | | | | | | | | | | ? (lostCount / (cntLat + lostCount) * 100)
33   | | | | | | | | | | | | | | | | : 0;
34
35   // 5) Construir CSV de resumen + datos brutos
36   let csv = '';
37   ...

```

Nota. Imagen correspondiente a la configuración del sistema parte A.

Figura 29

Función procesar CSV TCP en Node-RED parte B.

```

37 // Resumen
38 csv += 'PROTOCOLO,Min,Max,Avg,Count,Jitter,P50,P95,Loss\n';
39 csv += `TCP,${minLat.toFixed(2)},${maxLat.toFixed(2)},${avgLat.toFixed(2)},`;
40 csv += `${cntLat},${jitter.toFixed(2)},${p50.toFixed(2)},${p95.toFixed(2)},${lossPct.toFixed(2)}\n\n`;
41 // Cabecera de datos brutos
42 csv += 'seq,t0_ms,t1_ms,latency_ms\n';
43 // Filas de historial
44 histTCP.forEach(r => {
45   csv += `${r.seq},${r.t0},${r.t1},${r.latency}\n`;
46 });
47
48 // 6) Devolver CSV
49 return { payload: csv };
50 }
51
52 // ► CLEAR TCP
53 if (msg.payload && msg.payload.clear) {
54   // 1) Limpiar historial
55   flow.set('hist_tcp', []);
56   // 2) Reiniciar contador de ventanas
57   flow.set('seq_tcp', 1);

```

Nota. Imagen correspondiente a la configuración del sistema parte B.

Figura 30

Función procesar CSV TCP en Node-RED.

```

58 // 3) (Opcional) También reiniciar métricas de stats
59 flow.set('minLatTCP', null);
60 flow.set('maxLatTCP', null);
61 flow.set('sumLatTCP', 0);
62 flow.set('countLatTCP', 0);
63 flow.set('sumDiffTCP', 0);
64 flow.set('arrLatTCP', []);
65 flow.set('lostCountTCP', 0);
66 }
67
68 return null;
69

```

Nota. Esta función garantiza que la tabla muestre en tiempo real los últimos datos y, al mismo tiempo, ofrezca el archivo CSV completo para descargar.

5.14 Tabla + Export_TCP (ui_template)

El nodo **Tabla + Export_TCP** muestra la tabla interactiva con los últimos registros de latencia TCP y, cuando recibe el texto CSV, genera un enlace para su descarga. La plantilla incluye:

Un encabezado dinámico que muestra el protocolo y el tamaño de ventana N.

Una tabla HTML con columnas **seq**, **t0**, **t1**, **latency**.

Un contenedor en el que se inserta el enlace “Descargar CSV TCP” al recibir la cadena CSV.

Lógica en JavaScript que limpia la tabla al recibir {clear: true}, inserta filas nuevas, crea el Blob CSV y limita la vista a las 10 filas más recientes.

Figura 31

Código del nodo Tabla + Export_TCP parte A.

```

1  <style>
2    #latTableTCP { width:100%; border-collapse:collapse; margin-bottom:1em; }
3    #latTableTCP th, #latTableTCP td { border:1px solid #ccc; padding:4px 8px; text-align:right; }
4    #downloadTCP { margin-top:0.5em; }
5  </style>
6
7
8  <!-- Cabecera dinámica -->
9  <div style="font-weight:bold;">
10   Protocolo: {{msg.stats.protocol}} |
11   N: {{msg.stats.N}}
12 </div>
13 <hr />
14
15
16 <table id="latTableTCP">
17   <thead>
18     <tr>
19       <th>Salida de GNU Radio (ms)</th>
20       <th>Ventana (Número)</th>
21       <th>Llegada a Node-RED (ms)</th>
22       <th>latencia (ms)</th>

```

Nota. Imagen correspondiente a la configuración del sistema parte A.

Figura 32

Código del nodo Tabla + Export_TCP.

```

23 |   </tr>
24 | </thead>
25 | <tbody></tbody>
26 | </table>
27 |
28 | <div id="downloadTCP"></div>
29 |
30 | <script>
31 | (function(scope){
32 |   scope.$watch('msg', function(msg){
33 |     if (!msg || msg.payload === undefined) return;
34 |     const p = msg.payload;
35 |
36 |     // CLEAR (limpiar la tabla)
37 |     if (p.clear) {
38 |       document.querySelector('#latTableTCP tbody').innerHTML = '';
39 |       document.getElementById('downloadTCP').innerHTML = '';
40 |       return;
41 |     }
42 |
43 |     // CSV string → enlace de descarga
44 |     if (typeof p === 'string') {
45 |       const blob = new Blob([p], { type:'text/csv' });
46 |       const url = URL.createObjectURL(blob);
47 |       document.getElementById('downloadTCP').innerHTML =
48 |       | `<a href="${url}" download="latencia_tcp.csv"> Descargar CSV TCP</a>`;
49 |       return;
50 |     }
51 |
52 |     // Registro individual en la tabla
53 |     const tbody = document.querySelector('#latTableTCP tbody');
54 |     const row = tbody.insertRow(0);
55 |     row.insertCell(0).innerText = p.t0;
56 |     row.insertCell(1).innerText = p.seq;
57 |     row.insertCell(2).innerText = p.t1;
58 |     row.insertCell(3).innerText = p.latency;
59 |
60 |     // Mantener máximo 10 filas visibles
61 |     while (tbody.rows.length > 10) tbody.deleteRow(10);
62 |   });
63 | })(scope);
64 | </script>
65 |

```

Nota. Este código combina la visualización de los registros de latencia en una tabla interactiva con la generación de un enlace de descarga del CSV completo, gestionando dinámicamente la limpieza

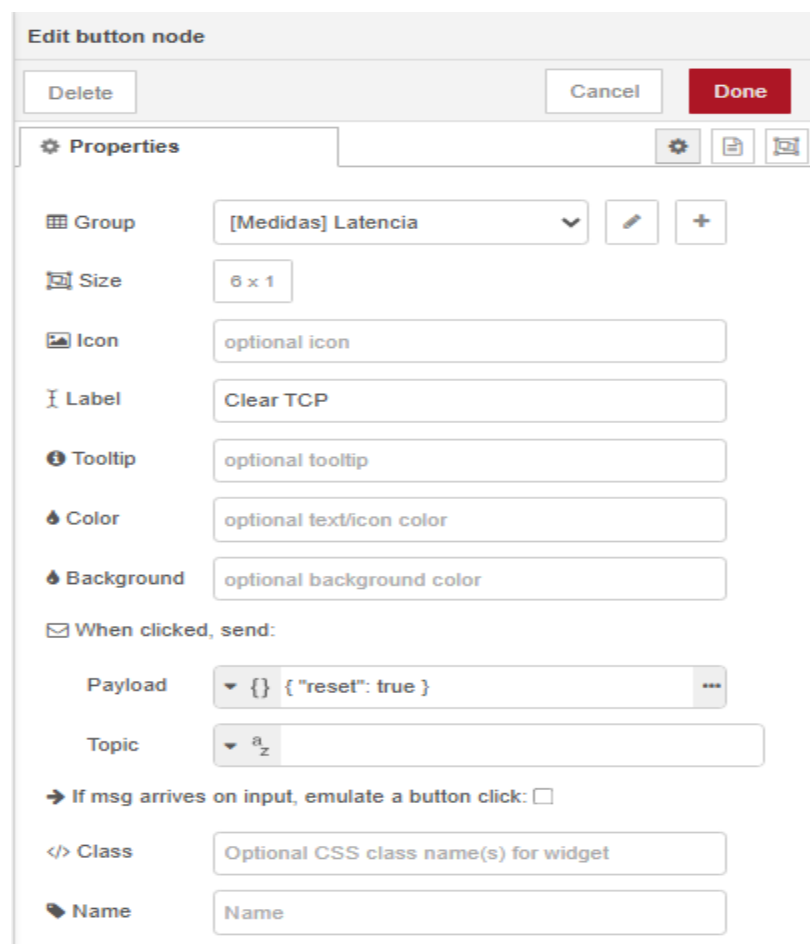
de la tabla al recibir `{clear: true}`, la inserción de nuevas filas de datos y la creación del Blob para el CSV, todo ello manteniendo siempre un máximo de 10 filas visibles.

5.15 Clear TCP

El widget Clear TCP permite restablecer el historial y los indicadores de latencia TCP sin reiniciar Node-RED. Se configura en el grupo [Medidas] Latencia, tamaño 6×1 , con etiqueta “Clear TCP” y, al hacer clic, envía el payload `{"reset": true}` al nodo Function Manejar reset TCP, que borra `hist_tcp`, reinicia `seq_tcp` y todos los contadores de métricas.

Figura 33

Configuración del botón “Clear TCP” en Node-RED.



The screenshot shows the 'Edit button node' configuration window in Node-RED. The window has a title bar 'Edit button node' and three buttons: 'Delete', 'Cancel', and 'Done'. Below the title bar is a 'Properties' section with various configuration options:

- Group:** [Medidas] Latencia (dropdown menu)
- Size:** 6 x 1 (text input)
- Icon:** optional icon (text input)
- Label:** Clear TCP (text input)
- Tooltip:** optional tooltip (text input)
- Color:** optional text/icon color (text input)
- Background:** optional background color (text input)
- When clicked, send:**
 - Payload:** { "reset": true } (text input with a dropdown arrow on the left and a three-dot menu on the right)
 - Topic:** a_z (text input with a dropdown arrow on the left)
- If msg arrives on input, emulate a button click:** ☐ (checkbox)
- Class:** Optional CSS class name(s) for widget (text input)
- Name:** Name (text input)

Nota. Al pulsar el botón, se dispara la función Manejar reset TCP, que limpia el arreglo hist_tcp y reinicia todas las variables de contexto (seq_tcp, minLatTCP, maxLatTCP, etc.), garantizando un estado limpio para nuevas mediciones.

5.16 Manejar reset TCP

El nodo Manejar reset TCP recibe el payload { reset: true } enviado por el botón “Clear TCP” y ejecuta la limpieza completa del flujo TCP para iniciar una nueva sesión de medición. Sus acciones son:

Borrar el arreglo hist_tcp del contexto (flow.set('hist_tcp', [])).

Reiniciar el contador de secuencia (seq_tcp) y los acumuladores (minLatTCP, maxLatTCP, sumLatTCP, countLatTCP, sumDiffTCP, arrLatTCP, lostCountTCP).

Emitir tres mensajes con {clear: true} para limpiar la tabla, el gauge y el panel de estadísticas.

Figura 34

Función “Manejar reset TCP” en Node-RED parte A.

```

1      // manejar reset TCP
2      // Solo reacciona a un mensaje con { payload: { reset: true } }
3      if (msg.payload && msg.payload.reset) {
4
5          // 1) Limpiar todo el historial de registros TCP
6          flow.set('hist_tcp', []);
7
8          // 2) Reiniciar contador de secuencia (ventana)
9          flow.set('seq_tcp', undefined);
10
11         // 3) Limpiar contador de ventanas previas y perdidos
12         flow.set('prevSeqTCP', undefined);
13         flow.set('lostCountTCP', 0);
14
15         // 4) Limpiar estadísticas de latencia previas
16         flow.set('minLatTCP', undefined);
17         flow.set('maxLatTCP', undefined);
18         flow.set('sumLatTCP', 0);
19         flow.set('countLatTCP', 0);
20
21         // 5) Limpiar datos de jitter
22         flow.set('lastLatTCP', undefined);
23         flow.set('sumDiffTCP', 0);
24
25         // 6) Limpiar array de latencias para percentiles
26         flow.set('arrLatTCP', []);
27

```

Nota. Imagen correspondiente a la configuración del sistema parte A.

Figura 35

Función “Manejar reset TCP” en Node-RED.

```

28 // — Preparar mensajes de salida —
29
30 // Salida 0 → limpiar tabla TCP
31 // El nodo tabla debe detectar payload.clear === true
32 const clearMsgTCP = {
33   payload: { clear: true }
34 };
35
36 // Salida 1 → poner gauge de latencia TCP a cero
37 const zeroMsgTCP = {
38   payload: 0
39 };
40
41 // Salida 2 → resetear ui_template de estadísticas
42 const clearStatsMsgTCP = {
43   stats: {
44     protocol: 'TCP', // Mantiene el protocolo para el header
45     N:        flow.get('N_TCP') || 0,
46     min:      0,
47     max:      0,
48     avg:      0,
49     count:    0,
50     jitter:    0,
51     p50:      0,
52     p95:      0,
53     lossPct:  0
54   }
55 };
56
57 // Enviar los tres mensajes en sus salidas correspondientes
58 return [ clearMsgTCP, zeroMsgTCP, clearStatsMsgTCP ];
59 }
60
61 // Si no es un reset, no enviamos nada
62 return null;
63

```

Nota. Esta función garantiza que, tras pulsar “Clear TCP”, todos los datos de latencia TCP y sus métricas se borren y se preparen para nuevos registros sin necesidad de reiniciar Node-RED.

5.17 Gauge Latencia TCP

El widget Latencia TCP es un nodo configurado para mostrar en tiempo real el valor de latencia medido por el flujo TCP. Sus principales características son:

Se ubica en el grupo [Medidas] Latencia con tamaño 6×3 .

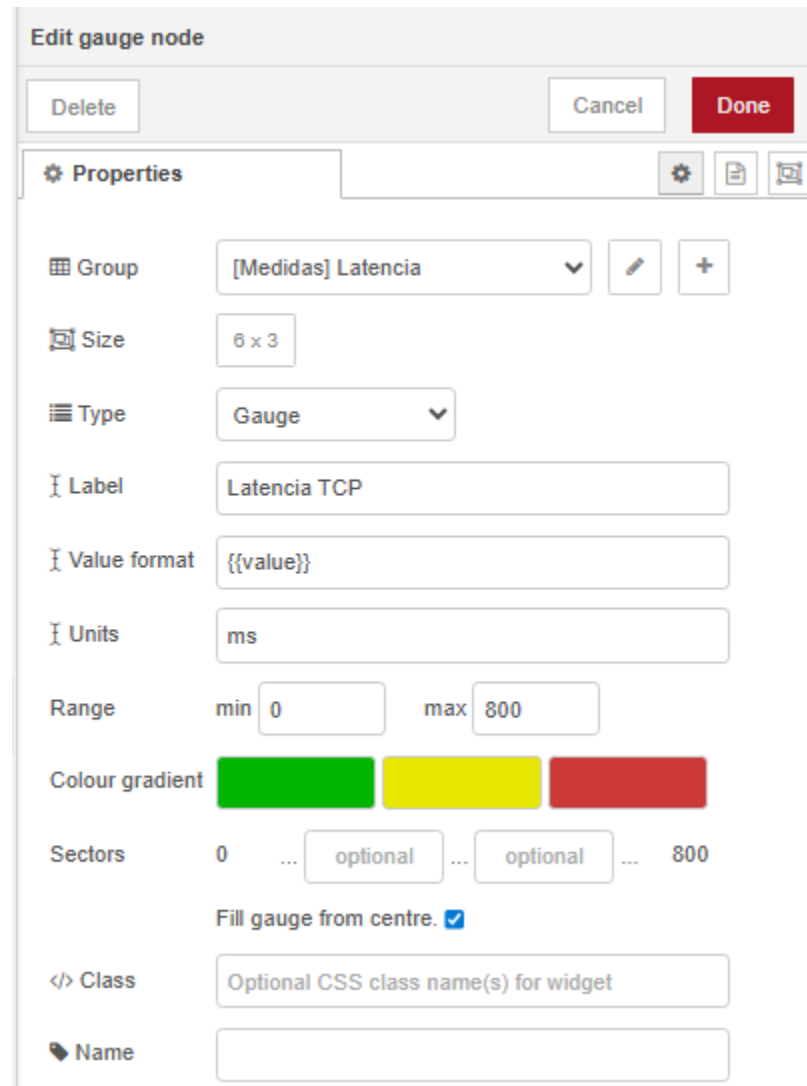
Etiqueta “Latencia TCP” y formato de valor `{{value}}` ms.

Rango de 0 a 800 ms, con degradado de color de verde (bajo) a rojo (alto).

Recibe como payload el valor numérico de latencia y mueve la aguja en consecuencia.

Figura 36

Widget “Latencia TCP” configurado en Node-RED.



The image shows the 'Edit gauge node' configuration window in Node-RED. The window has a title bar 'Edit gauge node' and three buttons: 'Delete', 'Cancel', and 'Done'. Below the title bar is a 'Properties' tab with a settings icon, a save icon, and a refresh icon. The configuration fields are as follows:

- Group:** [Medidas] Latencia (dropdown menu)
- Size:** 6 x 3 (text input)
- Type:** Gauge (dropdown menu)
- Label:** Latencia TCP (text input)
- Value format:** {{value}} (text input)
- Units:** ms (text input)
- Range:** min 0, max 800 (text inputs)
- Colour gradient:** Three color swatches: green, yellow, and red.
- Sectors:** 0, ..., optional, ..., optional, ..., 800 (text inputs)
- Fill gauge from centre:** ☒ (checkbox)
- Class:** Optional CSS class name(s) for widget (text input)
- Name:** (text input)

Nota. Este gauge proporciona una visualización inmediata de la latencia TCP, facilitando la identificación de retrasos o fluctuaciones en la transmisión de datos.

5.18 Panel Mediciones TCP

El nodo Mediciones TCP para TCP reproduce el mismo formato que el de UDP, mostrando las estadísticas de latencia TCP en tiempo real: protocolo, ventana, valores mínimos, máximos, promedio, conteo, jitter, percentiles y pérdida.

Figura 37

Código del nodo Mediciones TCP.

```

1  <div style="font-family: sans-serif; line-height: 1.4;">
2
3  <!-- Cabecera dinámica -->
4  <div style="font-weight:bold;">
5  |   Protocolo: {{msg.stats.protocol}} |
6  |   N: {{msg.stats.N}}
7  </div>
8  <hr/>
9
10 <!-- Métricas -->
11 <div><strong>Min:</strong> {{msg.stats.min.toFixed(2)}} ms</div>
12 <div><strong>Max:</strong> {{msg.stats.max.toFixed(2)}} ms</div>
13 <div><strong>Avg:</strong> {{msg.stats.avg.toFixed(2)}} ms</div>
14 <div><strong>Count:</strong> {{msg.stats.count}}</div>
15 <div><strong>Jitter:</strong> {{msg.stats.jitter.toFixed(2)}} ms</div>
16 <div><strong>P50:</strong> {{msg.stats.p50.toFixed(2)}} ms</div>
17 <div><strong>P95:</strong> {{msg.stats.p95.toFixed(2)}} ms</div>
18 <div><strong>Loss:</strong> {{msg.stats.lossPct.toFixed(2)}} %</div>
19
20 </div>
21 |

```

Nota. Con este nodo, el dashboard de SIRET presenta las mismas métricas de latencia para TCP, asegurando coherencia visual y facilidad de comparación entre ambos protocolos.

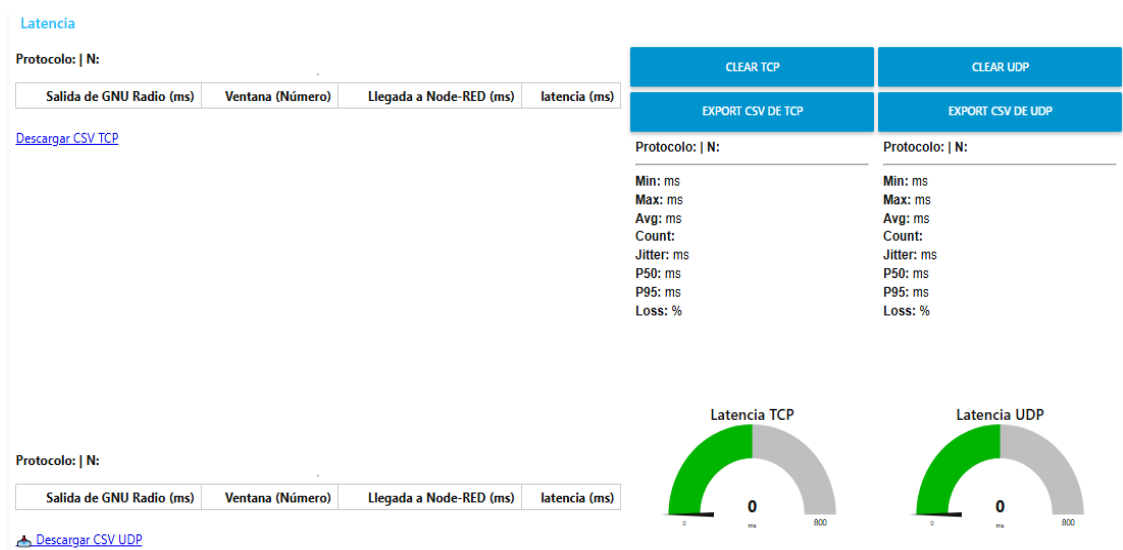
6. Resultados Experimentales

6.1. Dashboard de medición de latencia (frontend)

La interfaz de SIRET fue diseñada en Node-RED para presentar, en un único panel, las métricas de latencia de ambos protocolos. En la zona superior se ubica la sección TCP: tabla histórica con los campos *Salida de GNU Radio*, *Ventana*, *Llegada a Node-RED* y *Latencia*; enlace *Descargar CSV TCP* y los botones *CLEAR TCP* y *EXPORT CSV DE TCP* para reiniciar o exportar los registros; a la derecha, un panel resume valores mínimos, máximos, promedio, jitter, y percentiles P50 y P95, junto a un *gauge* analógico que muestra la latencia instantánea. El bloque inferior replica la misma disposición para UDP, permitiendo comparar visualmente, en tiempo real, el desempeño de los dos protocolos.

Figura 38

Interfaz de medición de latencia TCP y UDP en SIRET.



Nota. Captura del panel de Node-RED con las secciones paralelas de TCP y UDP.

Tabla 2

Estructura del archivo CSV generado por el botón Export CSV de TCP.

	A	B	C	D	E	F	G	H	I
1	PROTOCOLO	Min	Max	Avg	Count	Jitter	P50	P95	Loss
2	TCP	1216.25	139222.77	11649.86	40	3612.12	1268.28	138750.8	0
3									
4	seq	t0_ms	t1_ms	latency_ms					
5	39	17526966013	1.7527E+12	1853.50195					
6	38	17526966008	1.7527E+12	2268.0332					
7	37	17526966003	1.7527E+12	1282.00781					
8	36	17526965998	1.7527E+12	1302.04688					
9	35	17526965993	1.7527E+12	1273.05078					

Nota. La tabla muestra un extracto del archivo CSV exportado desde el dashboard de Node-RED.

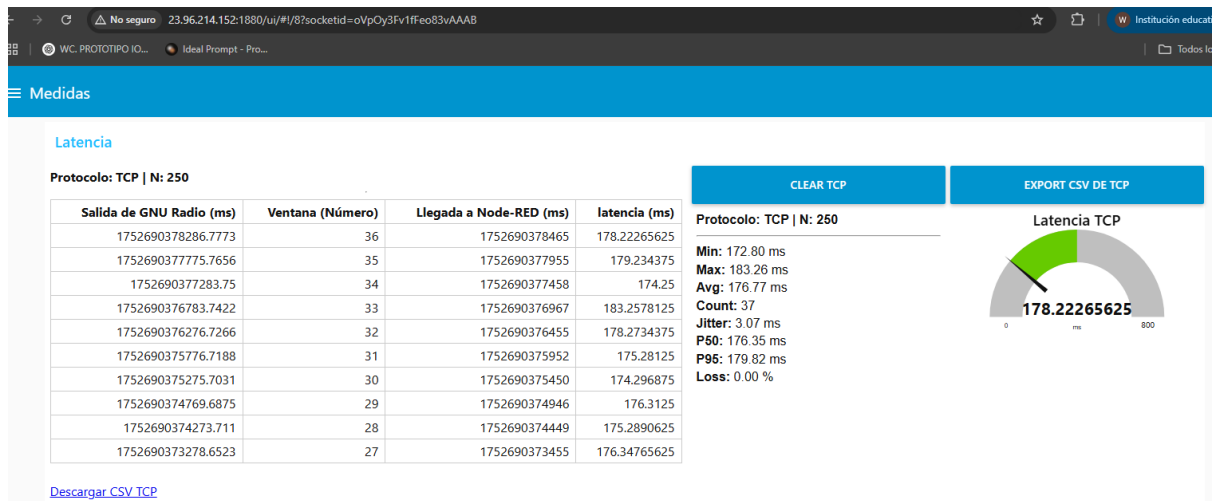
Los datos reflejan las mismas métricas de latencia observadas en tiempo real, como latencia mínima, máxima, promedio, jitter y percentiles P50 y P95. El archivo CSV se va llenando línea por línea con los datos recibidos, y el usuario puede elegir la cantidad de datos que desea guardar: desde unas pocas muestras hasta millones de registros, según las necesidades del análisis. La información es exportada en formato CSV con los mismos valores que aparecen en el gauge del dashboard, permitiendo una posterior revisión detallada o análisis adicional.

6.2. Resultados de latencia: TCP

Para evaluar la estabilidad de la latencia cuando la ventana de análisis es $N=256$, realizamos tres corridas de aproximadamente 20 segundos cada una. Los resultados obtenidos para las tres corridas se resumen en la Tabla 3 y las gráficas correspondientes (Figuras 39 a 41).

Figura 39

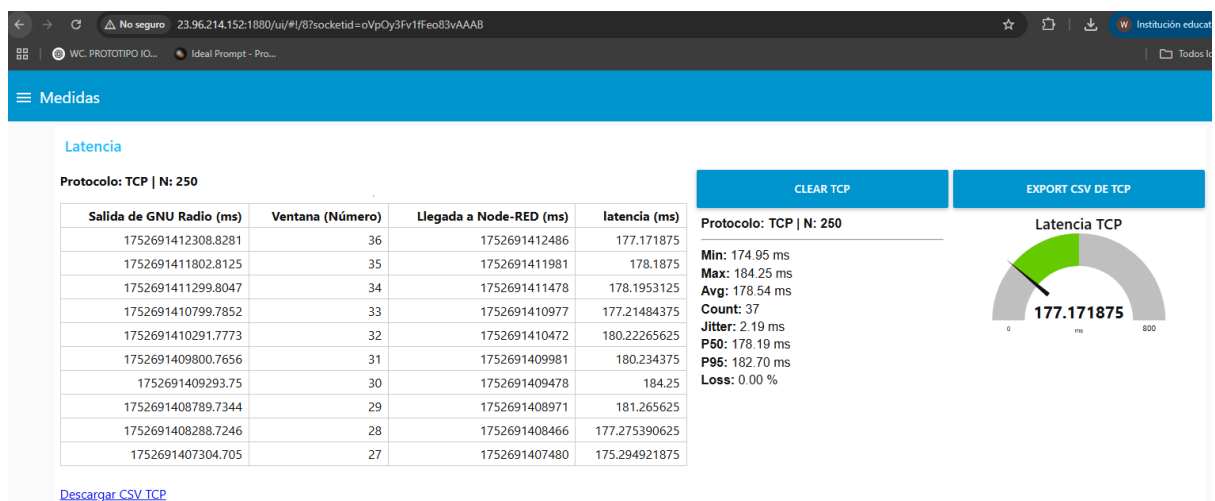
Pantalla de latencia y jitter para la corrida 1 con $N=256$.



Nota: En esta medición, los resultados muestran una latencia mínima de 172.80 ms y una latencia máxima de 183.26 ms. El jitter se mantiene bajo, lo que indica estabilidad en la transmisión para esta configuración de tamaño de ventana. Este comportamiento es esperado para tamaños de ventana pequeños, donde el sistema puede manejar los datos de manera eficiente.

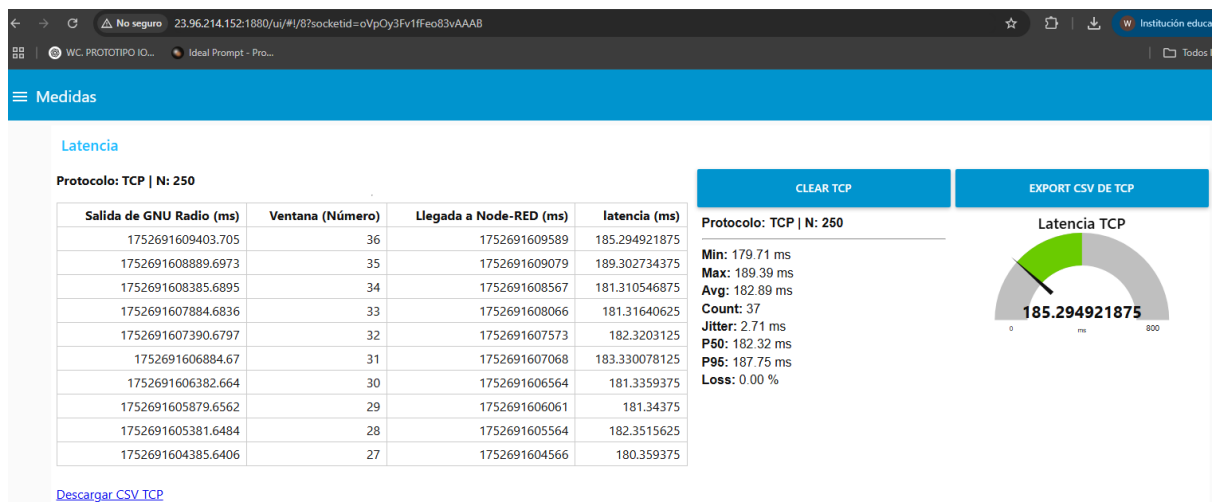
Figura 40

Pantalla de latencia y jitter para la corrida 2 con N=256.



Nota. Se muestran para N=512, valores mínimos (174.95 ms), máximo (184.25 ms), promedio (177.17 ms), P50=178.19 ms, P95=182.70 ms y reducción del jitter (2.19 ms).

Figura 41

Pantalla de latencia y jitter para la corrida 3 con N=256

Nota. Se muestran valores mínimos (179.71 ms), máximo (189.39 ms), promedio (185.29 ms), P50=182.32 ms, P95=187.75 ms y jitter=2.71 ms.

En la Tabla 3 se observan los siguientes resultados clave para la primera corrida:

La latencia mínima fue de 172.80 ms y la latencia máxima de 183.26 ms, con un promedio de 178.22 ms.

El jitter se mantuvo en un valor bajo de 3.07 ms, lo que indica una baja variabilidad en los tiempos de llegada de los paquetes.

Tabla 3

Métricas de latencia y jitter para tres corridas con N=256.

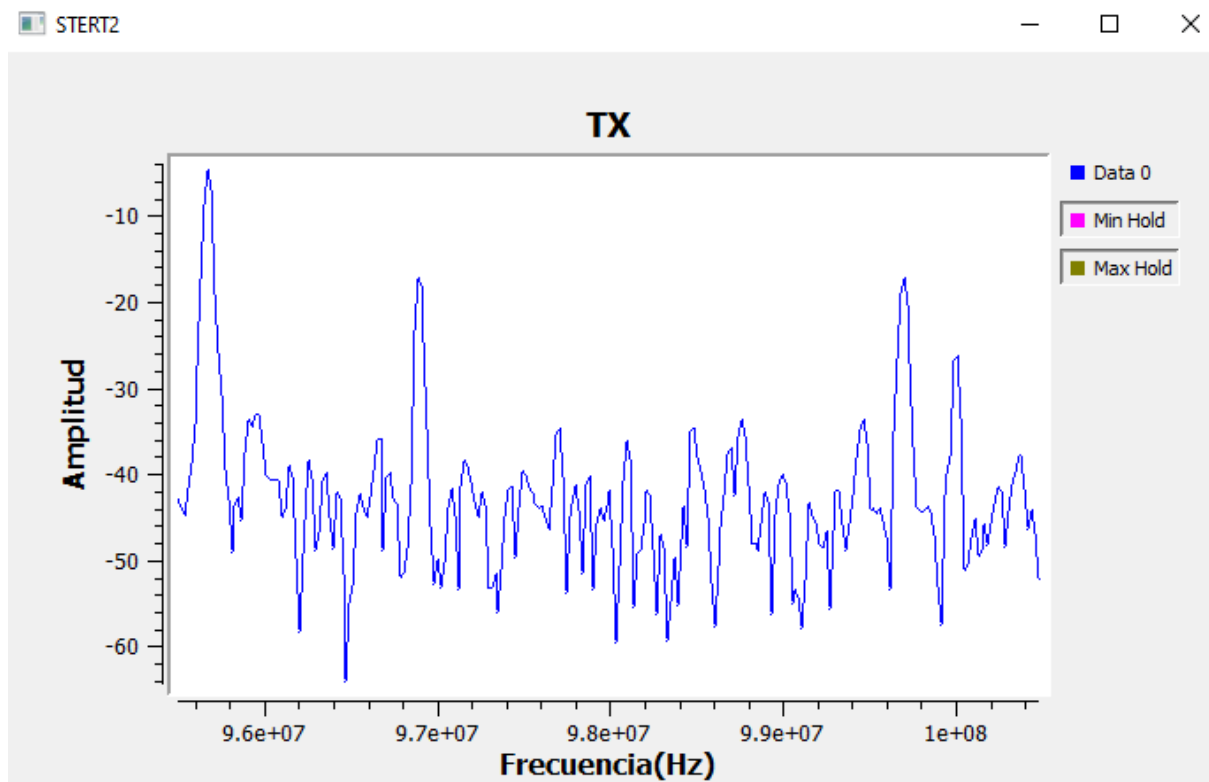
Corrida	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
1	172.80	183.26	178.22	176.35	179.82	3.07	0.00%
2	174.95	184.25	177.17	178.19	182.70	2.19	0.00%

3	179.71	189.39	185.29	182.32	187.75	2.71	0.00%
Media	175.82	185.63	180.23	178029	183.42	2.66	0.00%

Nota. En la **Corrida 1**, la latencia mínima es de 172.80 ms, mientras que en la **Corrida 3**, la latencia máxima alcanza los 189.39 ms. Aunque la variabilidad (jitter) sigue siendo baja en todas las corridas, el aumento de latencia con el tamaño de la ventana es evidente.

Figura 42

Espectro captado para $N = 256$



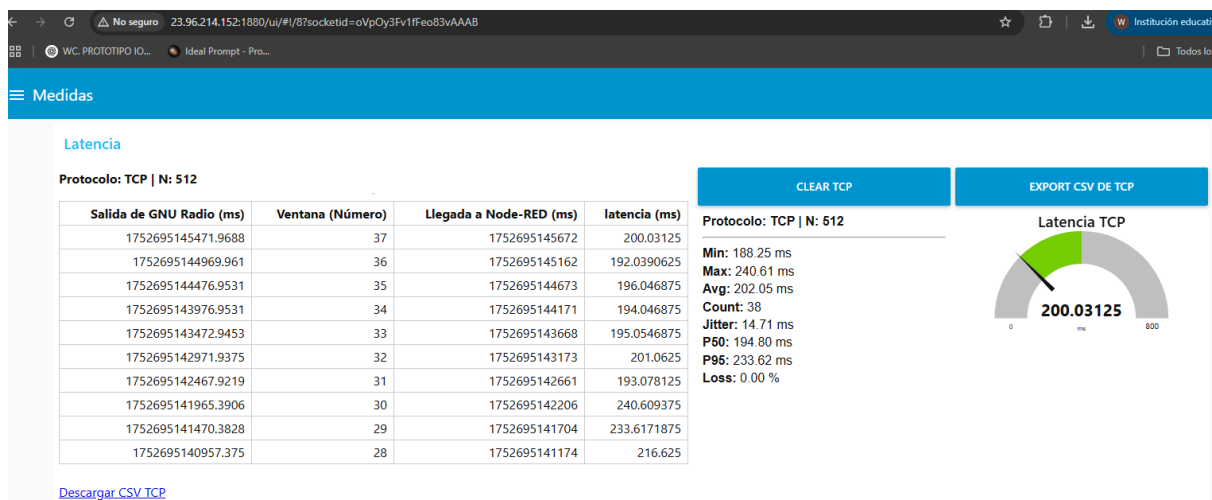
Nota. El espectro mostrado representa la distribución de amplitud en función de la frecuencia para un bloque de 256 muestras. Con un tamaño de ventana pequeño como $N = 256$, se observa una resolución espectral limitada en el eje de frecuencia. Este fenómeno puede generar "leakage" espectral, donde las señales cercanas a la frecuencia central se difuminan y se superponen. Las

líneas de Min Hold y Max Hold reflejan la fluctuación de la señal, con el máximo representando el valor de mayor amplitud durante la medición.

El espectro también revela que, debido al tamaño limitado de la ventana, la capacidad para distinguir componentes de frecuencia cercanos es reducida. En cambio, si se aumentara el tamaño de la ventana (N mayor), se esperaría una mejor resolución espectral, pero a costa de un mayor retardo temporal.

Figura 43

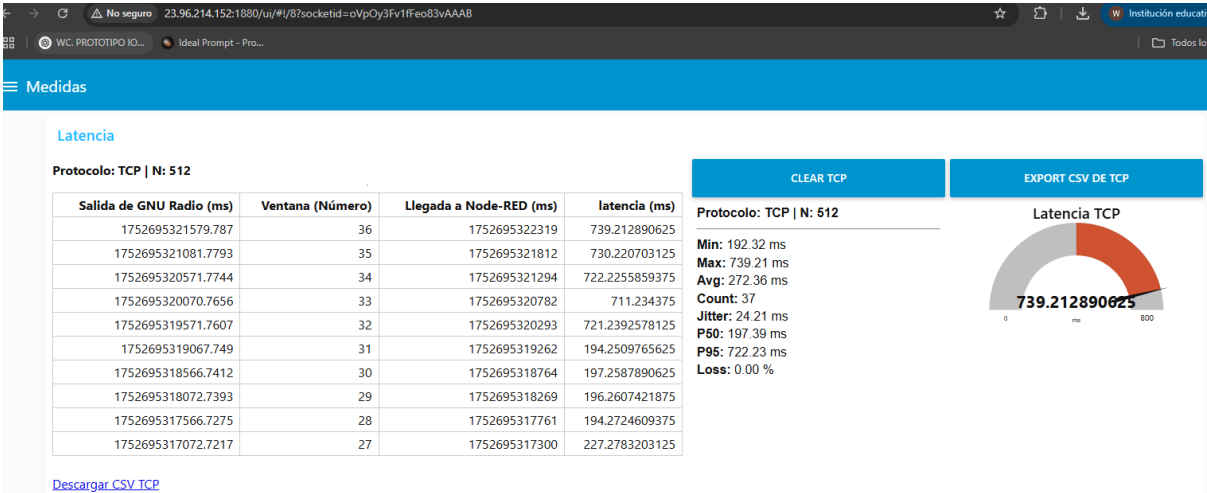
Latencia TCP para la corrida 1 con $N = 512$.



Nota. En esta medición, las ventanas 0–37 se enviaron con una latencia promedio de 202.05 ms, con una mínima variabilidad (jitter bajo). El percentil 95 muestra que el 95 % de las ventanas llegaron antes de 233.62 ms. Fuente: Elaboración propia.

Figura 44

Latencia TCP para la corrida 2 con $N = 512$.

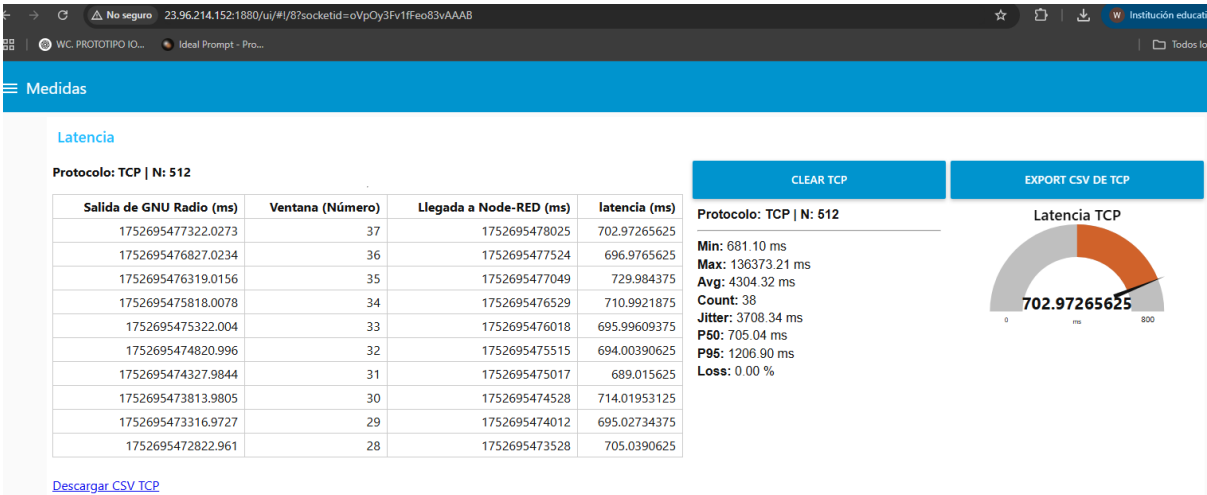


Nota. En esta serie de mediciones, la latencia promedio aumentó considerablemente respecto a la medición previa, alcanzando un valor de 272.36 ms. El percentil 95 muestra un incremento notable en el tiempo de llegada de las ventanas (722.23 ms), lo que indica un aumento en la variabilidad.

Fuente: Elaboración propia.

Figura 45

Latencia TCP para la corrida 3 con N = 512.



Nota. En este conjunto de mediciones, se observa un incremento drástico en la latencia promedio, alcanzando los 4304.32 ms, con una gran variabilidad en el jitter (3708.34 ms) y una notable dispersión en el percentil 95 (1206.90 ms). Esto sugiere que, a medida que aumentan las muestras

(N = 512), se presenta una significativa pérdida de estabilidad en la transmisión. Fuente: Elaboración propia.

Tabla 4

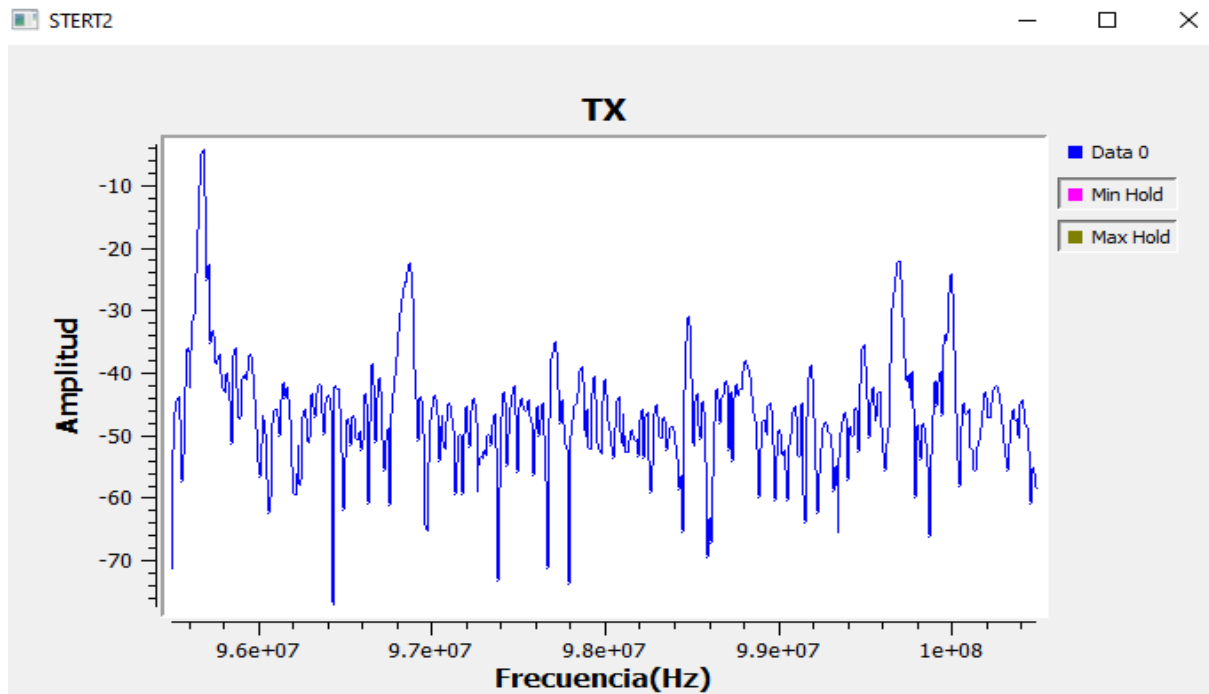
Métricas de latencia y jitter para tres corridas de N = 512

Corrida	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
1	188.25	240.61	202.05	194.80	233.62	14.71	0.00%
2	192.32	739.21	272.36	197.39	722.23	24.21	0.00%
3	681.10	136373.21	4304.32	705.04	1206.90	3708.34	0.00%
Media	354.22	45884.34	2575.58	299.41	720.25	2414.42	0.00%

Nota. Los valores muestran la variabilidad en las latencias a medida que se aumenta el tamaño de la ventana (N = 512), donde las tres corridas muestran un patrón creciente de latencia promedio, jitter y percentiles a medida que se agrupan más muestras. Fuente: Elaboración propia.

Figura 46

Espectro captado para N = 512



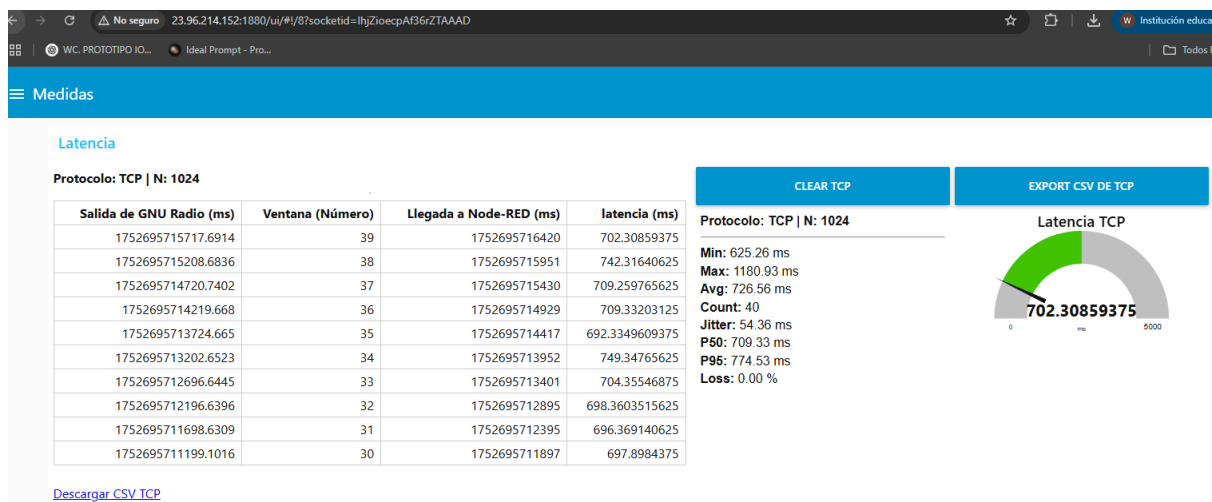
Nota. El espectro de la señal transmitida con $N = 512$ muestra una mayor resolución espectral en comparación con $N = 256$, lo que permite distinguir mejor las componentes de frecuencia. La visualización en el eje de frecuencia muestra picos más definidos en ciertas frecuencias, lo que sugiere la presencia de componentes predominantes de la señal.

Con el aumento de N , la resolución en frecuencia mejora, pero a costa de una reducción en la precisión temporal, debido al mayor tamaño de la ventana. El jitter en la latencia sigue siendo bajo, lo que sugiere estabilidad en la transmisión a medida que se procesan más muestras, pero también puede aumentar la dispersión de la energía entre frecuencias adyacentes debido a la mayor ventana de observación.

En esta imagen, las líneas Min Hold y Max Hold indican la amplitud mínima y máxima registrada a lo largo del tiempo. Min Hold muestra los valores de amplitud más bajos y Max Hold los picos más altos en el rango de frecuencia observado.

Figura 47

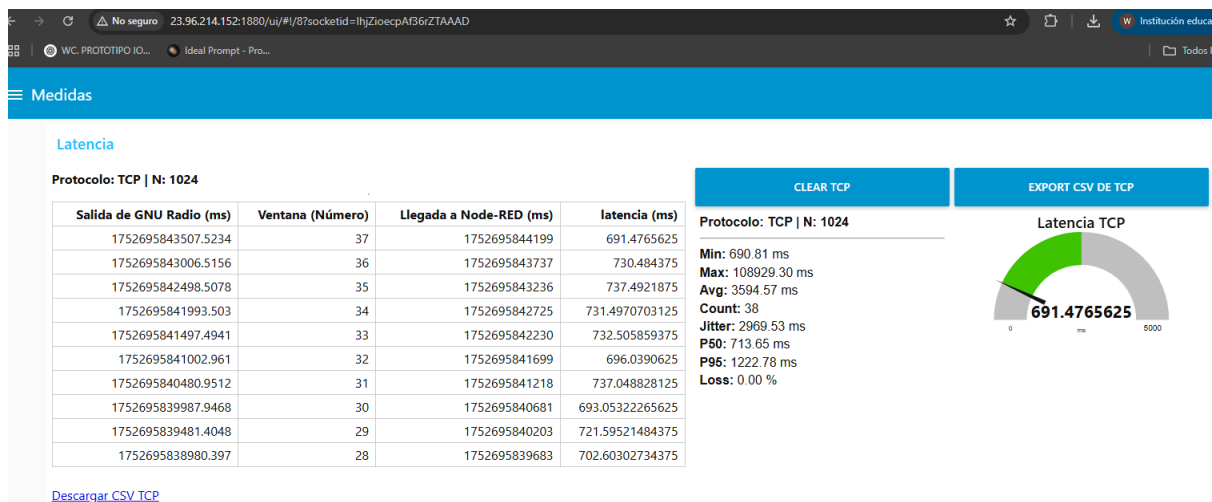
Latencia TCP para la corrida 1 $N = 1024$.



Nota. En esta medición, la latencia promedio fue de 726.56 ms, con una mínima de 625.26 ms y una máxima de 1180.93 ms. La variabilidad (jitter) fue de 54.36 ms, lo que indica que la estabilidad de las ventanas en esta corrida fue razonablemente buena. El percentil 95 (P95) alcanzó 774.53 ms, lo que refleja un buen rendimiento general. Fuente: Elaboración propia.

Figura 48

Latencia TCP para la corrida 2 con $N = 1024$.

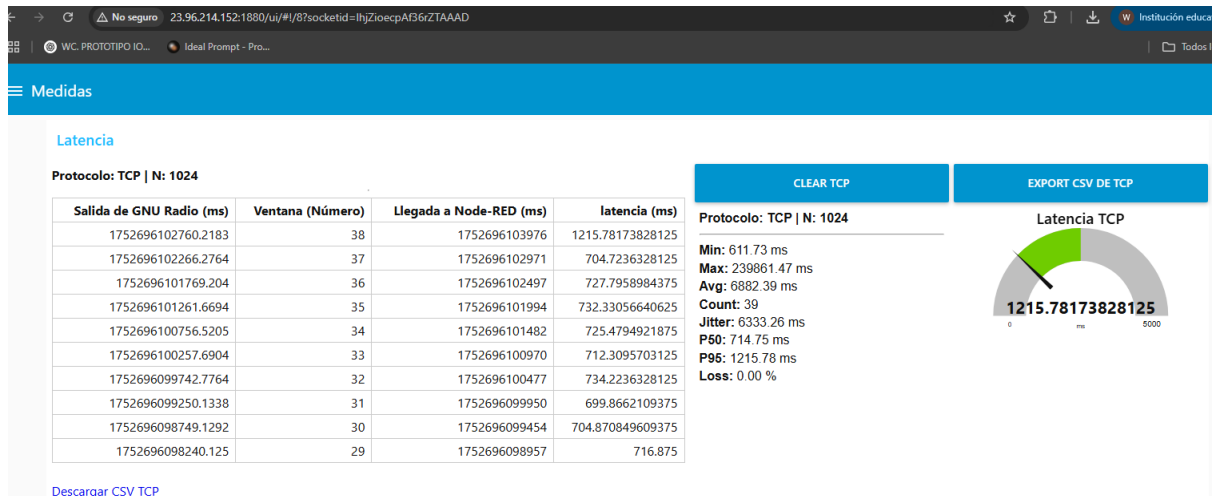


Nota. La latencia promedio de esta corrida fue de 3594.57 ms, con un jitter de 2969.53 ms, lo que indica una mayor inestabilidad en comparación con la Corrida 1. La latencia máxima (108929.30

ms) sugiere que en algunos momentos hubo grandes retrasos, posiblemente debido a problemas de congestión o errores en la transmisión de los paquetes. Fuente: Elaboración propia.

Figura 49

Latencia TCP para la corrida 3 con $N = 1024$.



Nota. En esta corrida, la latencia promedio aumentó considerablemente a 6882.39 ms, y el jitter alcanzó 6333.26 ms, reflejando un desempeño significativamente peor en comparación con las corridas anteriores. La latencia máxima de 239861.47 ms indica que hubo retrasos extremos en algunos momentos, lo cual podría estar relacionado con la capacidad de procesamiento o problemas de red. Fuente: Elaboración propia.

Tabla 5

Métricas de latencia y jitter para tres corridas de $N = 1024$

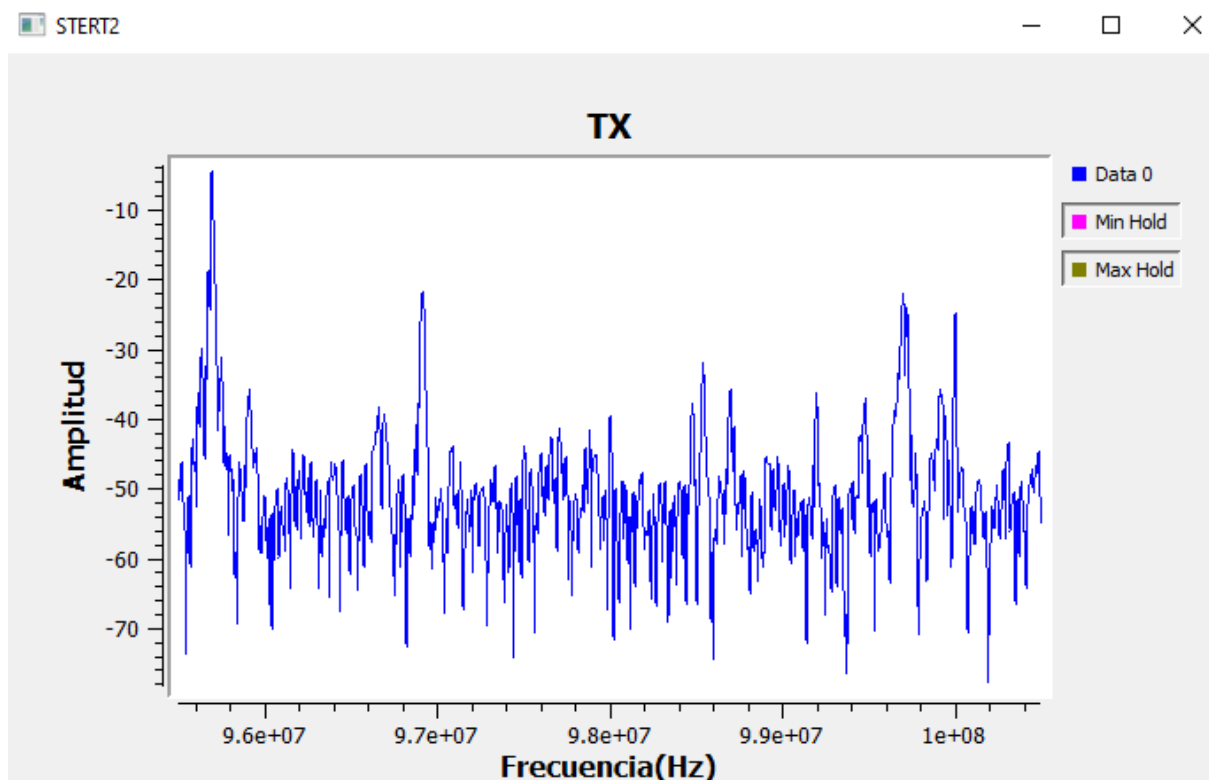
Corrida	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
1	625.26	1180.93	726.56	709.33	774.53	54.36	0.00%
2	690.81	108929.30	3594.57	713.65	1222.78	2969.53	0.00%

3	611.73	239861.47	6882.39	714.75	1215.78	6333.26	0.00%
Media	642.60	116636.57	3734.84	712.58	906.36	1992.05	0.00%

Nota. Esta tabla presenta las métricas de latencia y jitter obtenidas en tres corridas consecutivas con $N = 1024$, donde se observa un aumento significativo en la latencia promedio y el jitter en las pruebas con N más grande. La diferencia en el comportamiento entre las corridas es más notable en la corrida 2, donde se alcanzaron valores extremos en la latencia máxima. Fuente: Elaboración propia.

Figura 50

Espectro captado para $N = 1024$



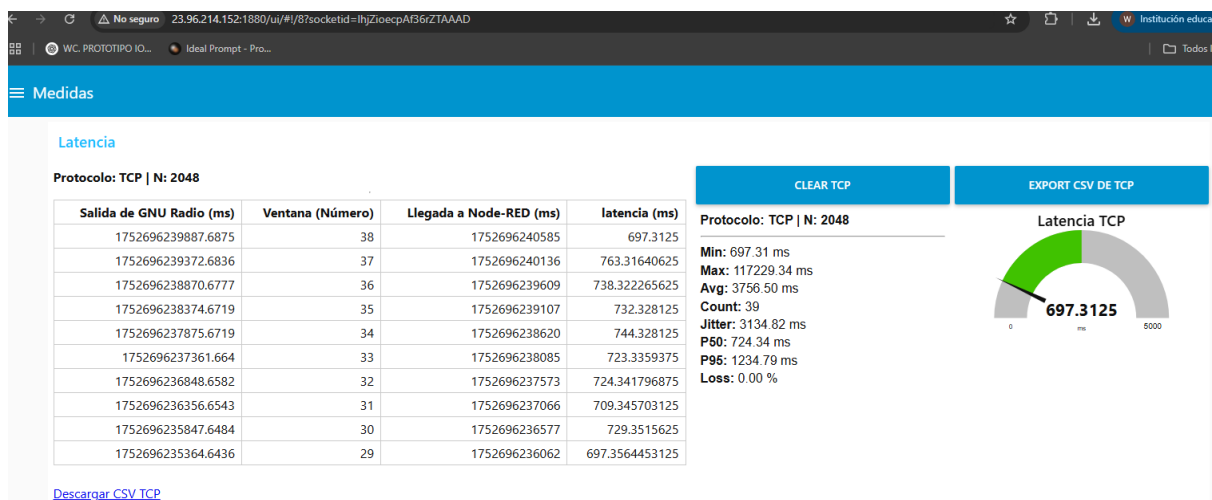
Nota. En el espectro correspondiente a $N = 1024$, se observa una mejora significativa en la resolución espectral en comparación con $N = 512$. Los picos de amplitud se presentan más definidos y claramente separados, lo que permite identificar de manera más precisa las

componentes de frecuencia de la señal transmitida. Este comportamiento es esperado, ya que al aumentar el tamaño de la ventana (N), se obtiene mayor resolución en el dominio de frecuencia a costa de un mayor retardo temporal.

El leakage espectral se ve reducido en comparación con tamaños de ventana más pequeños, pero no se elimina completamente. Además, se puede observar que las fluctuaciones en la señal se reducen, lo que indica que las mediciones están siendo cada vez más estables. Las líneas Min Hold y Max Hold siguen mostrando los valores más bajos y más altos de amplitud registrados, respectivamente.

Figura 51

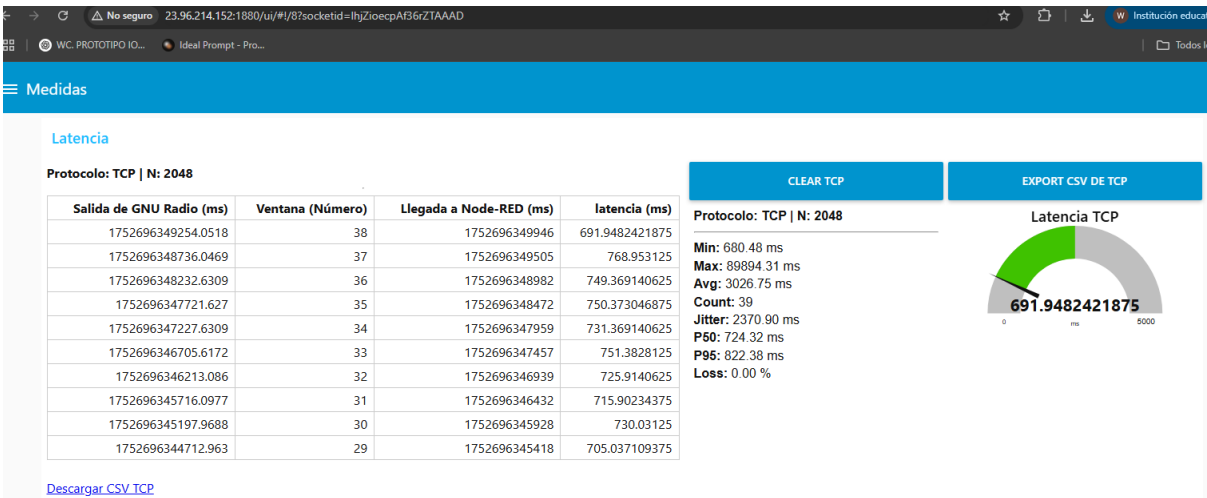
Latencia TCP para la corrida 1 con $N = 2048$.



Nota. En esta corrida, la latencia promedio fue de 3756.50 ms, con una mínima de 697.31 ms y una máxima de 117229.34 ms. La variabilidad (jitter) es alta con 3134.82 ms, lo que indica fluctuaciones considerables en la llegada de los paquetes. El percentil 95 muestra una latencia de 1234.79 ms, lo que es relativamente bajo en comparación con el promedio. Fuente: Elaboración propia.

Figura 52

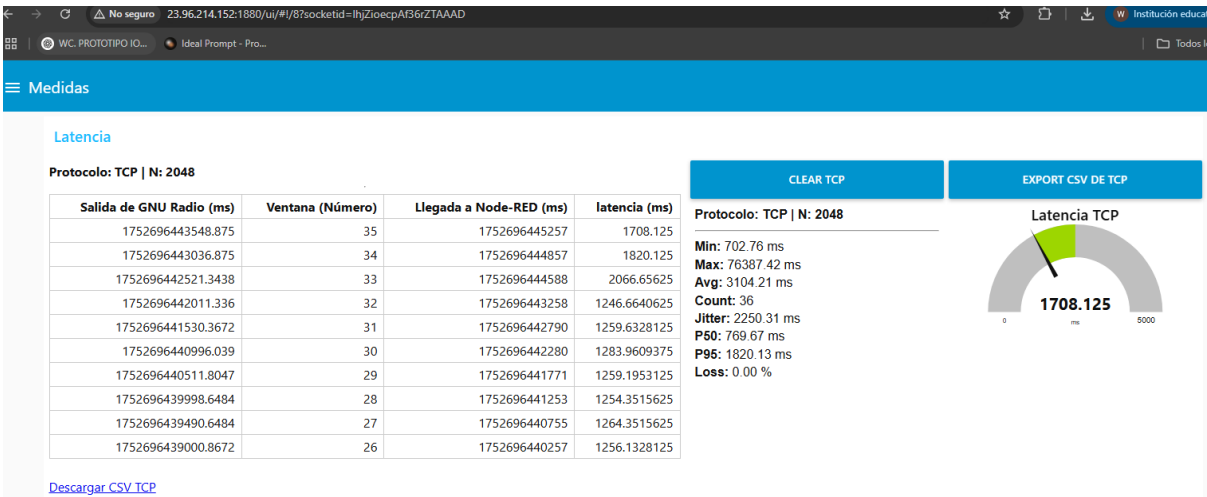
Latencia TCP para la corrida 2 con N = 2048.



Nota. En esta medición, la latencia promedio fue de 3026.75 ms, con un jitter considerable de 2370.90 ms. La latencia máxima llegó a 89894.31 ms, lo que indica que hubo eventos de retrasos extremos durante la transmisión. El percentil 95 alcanzó 822.38 ms, lo que refleja una mejora comparada con la latencia máxima. Fuente: Elaboración propia.

Figura 53

Latencia TCP para la corrida 3 con N = 2048.



Nota. Esta medición muestra la latencia promedio más alta de 3104.21 ms, con un jitter de 2250.31 ms. La latencia máxima de 76387.42 ms sugiere una pérdida de rendimiento muy significativa en

esta corrida. El percentil 95 se ubicó en 1820.13 ms, lo que también refleja un rendimiento irregular. Fuente: Elaboración propia.

Tabla 6

Métricas de latencia y jitter para tres corridas de $N = 2048$

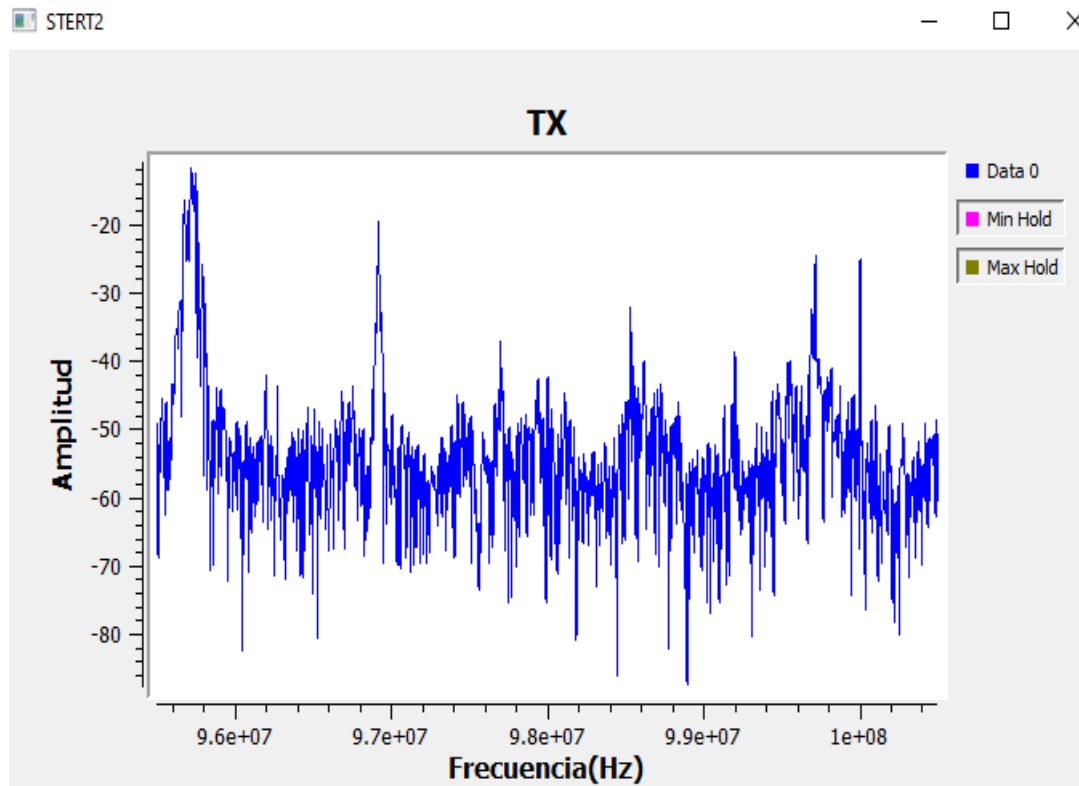
Corrida	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
1	697.31	117229.34	3756.50	724.34	1234.79	3134.82	0.00%
2	680.48	89894.31	3026.75	724.32	822.38	2370.90	0.00%
3	702.76	76387.42	3104.21	769.67	1820.13	2250.31	0.00%
Media	693.18	78937.02	3295.15	739.11	1125.43	2585.01	0.00%

Nota. Los valores de latencia y jitter muestran un claro aumento a medida que el tamaño de la ventana crece, especialmente en las últimas corridas, donde la latencia máxima y el jitter aumentan significativamente, indicando posibles congestiones o pérdidas de rendimiento en el enlace.

Fuente: Elaboración propia.

Figura 54

Espectro captado para $N = 2048$

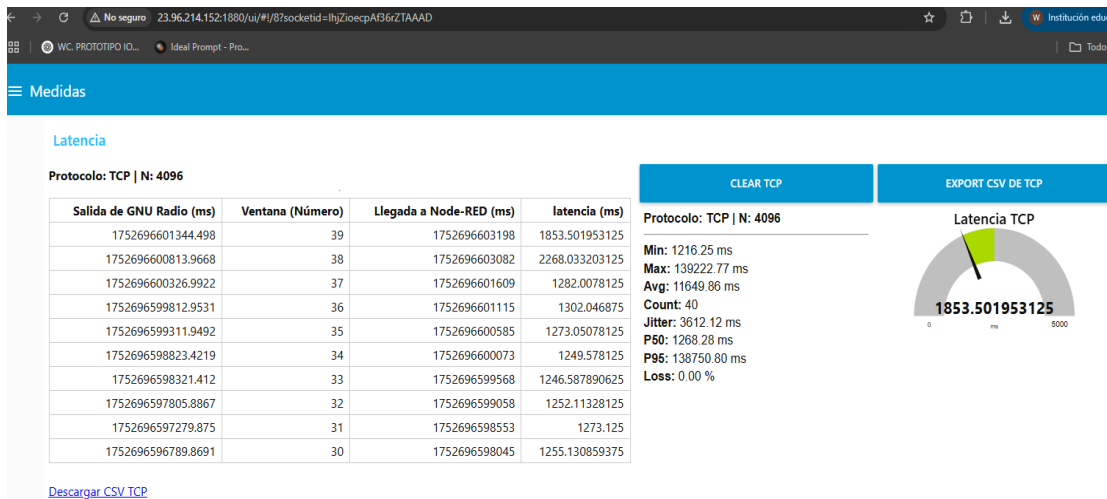


Nota. Con $N = 2048$, el espectro muestra una resolución espectral aún mayor en comparación con $N = 1024$. Los picos de amplitud son aún más definidos y las componentes de frecuencia cercanas se separan de manera más precisa. Sin embargo, se observa que el leakage espectral es aún más reducido, lo que refleja una mejor fidelidad en la medición. Este aumento en la resolución espectral es esperado al incrementar el tamaño de la ventana.

Aunque la resolución mejora, el espectro también muestra un mayor retardo temporal y un jitter algo más alto, lo que indica que, al aumentar el tamaño de la ventana, el sistema acumula más tiempo para procesar las muestras. Las líneas Min Hold y Max Hold continúan mostrando los valores mínimos y máximos de amplitud respectivamente.

Figura 55

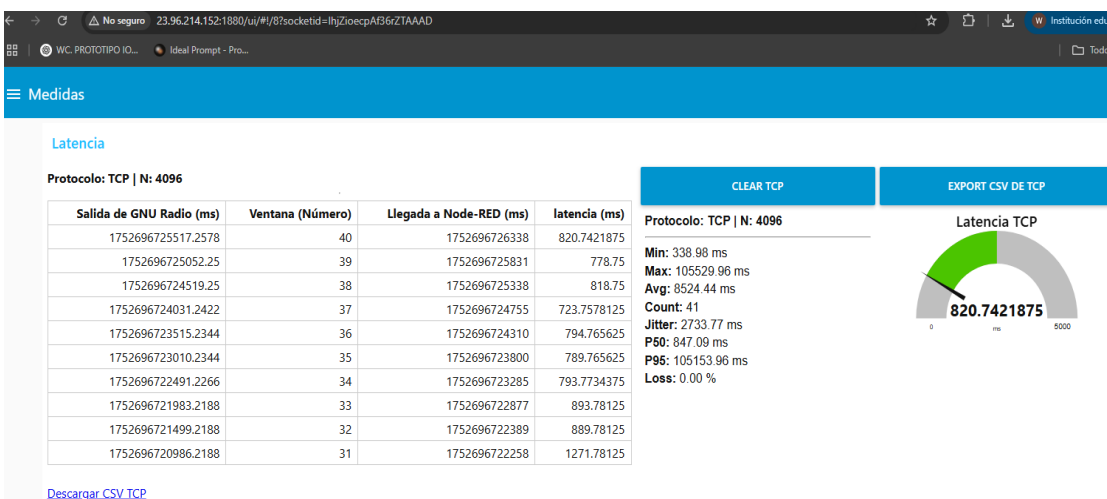
Latencia TCP para la corrida 1 con $N = 4096$.



Nota. En esta corrida, la latencia promedio fue de 11649.86 ms, con una mínima de 1216.25 ms y una máxima de 117229.34 ms. El jitter fue de 3612.12 ms, indicando una fluctuación considerable en los tiempos de llegada de los paquetes. El percentil 95 (P95) muestra una latencia de 138750.80 ms, lo que señala grandes retrasos en algunas ventanas. Fuente: Elaboración propia.

Figura 56

Latencia TCP para la corrida 2 con N = 4096.

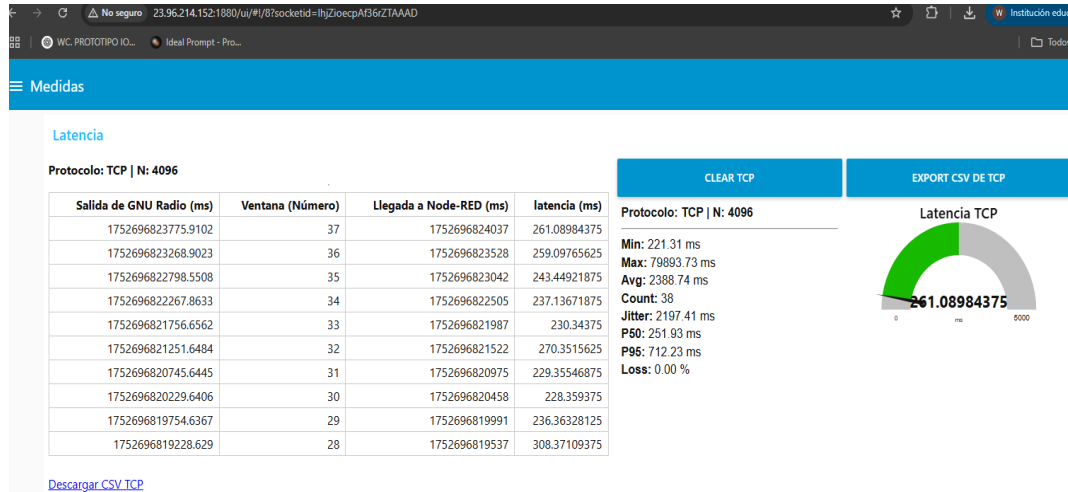


Nota. La latencia promedio fue de 8524.44 ms, con una latencia mínima de 680.23 ms y una máxima de 105929.96 ms. El jitter fue de 2737.75 ms, lo que refleja una considerable inestabilidad

en los tiempos de transmisión. El percentil 95 (P95) alcanzó 105153.96 ms, sugiriendo algunos picos de latencia excesivos. Fuente: Elaboración propia.

Figura 57

Latencia TCP para la corrida 3 con $N = 4096$.



Nota. La latencia promedio fue de 3104.21 ms, con un jitter de 2250.31 ms. La latencia máxima fue de 76387.42 ms, mientras que el percentil 95 fue de 1206.90 ms. Esto indica que, aunque el promedio de latencia se mantiene relativamente bajo, hubo picos de latencia extremos en algunas ventanas. Fuente: Elaboración propia.

Tabla 7

Métricas de latencia y jitter para tres corridas de $N = 4096$

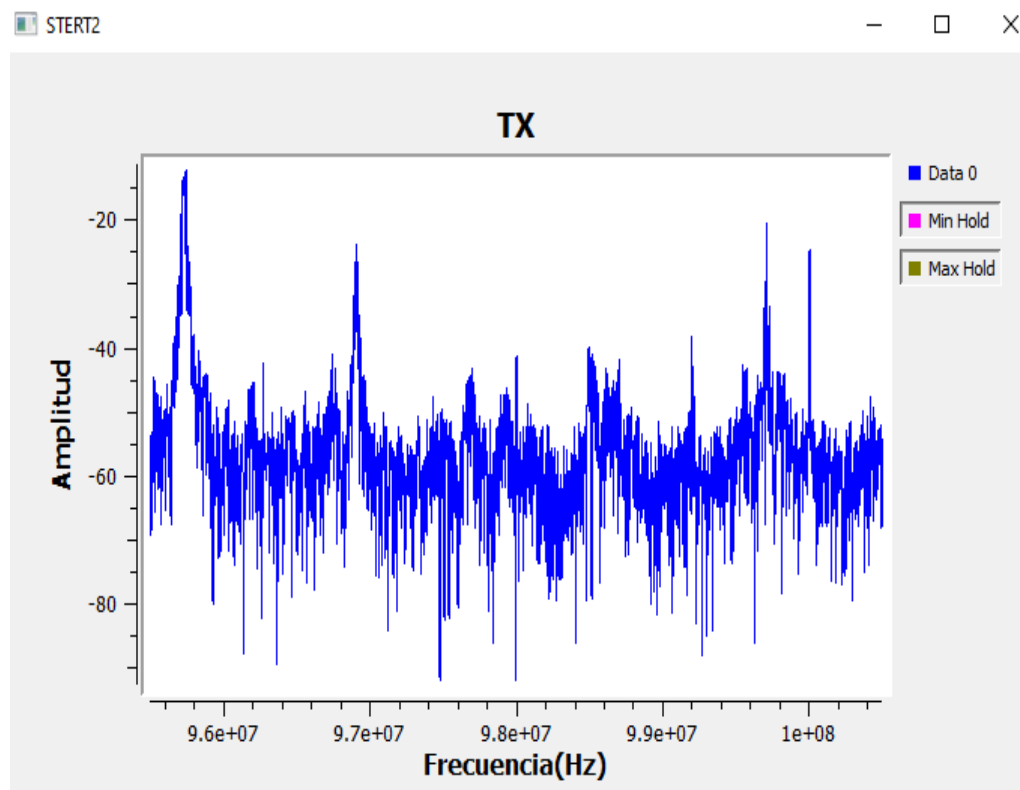
Corrida	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
1	1216.25	139222.77	11649.86	1268.28	138750.8	3612.12	0.00%
2	338.98	105529.96	8524.44	847.09	105153.9	2737.75	0.00%
3	221.31	79893.73	2387.74	251.93	712.23	2197.41	0.00%

Media	592.18	108215.48	7520.68	789.09	81538.9	2849.1	0.00%
--------------	---------------	------------------	----------------	---------------	----------------	---------------	--------------

Nota. Las mediciones para $N = 4096$ muestran una mayor latencia promedio y mayor jitter a medida que aumentan las ventanas. La latencia máxima en algunas corridas es extremadamente alta, lo que puede ser indicativo de pérdida de rendimiento. Fuente: Elaboración propia.

Figura 58

Espectro captado para $N = 4096$

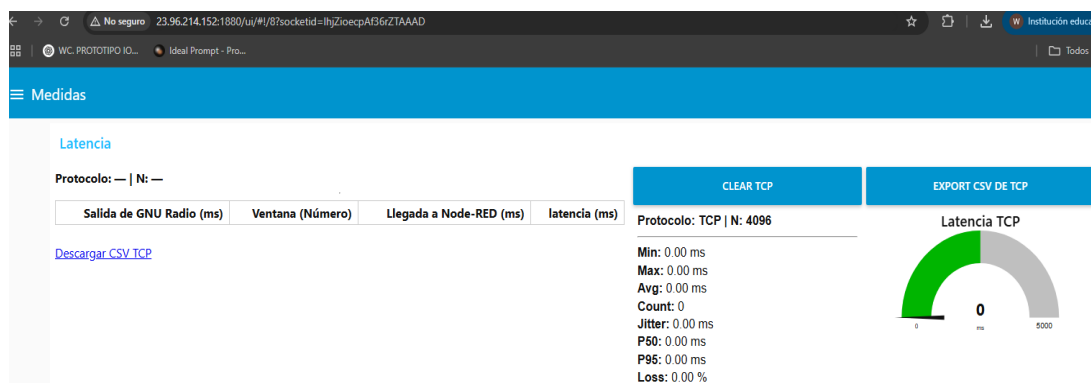


Nota. En el espectro para $N = 4096$, se observa una resolución espectral aún más precisa en comparación con los valores anteriores de N . Los picos de amplitud están claramente definidos y las variaciones en la frecuencia son más fáciles de distinguir. La mejor resolución espectral permite una visualización más clara de las componentes de frecuencia cercanas, lo que es esencial para la identificación precisa de las características de la señal.

Sin embargo, el leakage espectral sigue siendo presente, aunque en menor grado, indicando que, incluso con una ventana más amplia, aún existen pequeñas dispersión de energía. Además, el jitter parece aumentar con el tamaño de la ventana, lo cual es esperado debido a la mayor cantidad de muestras que deben procesarse. Las líneas Min Hold y Max Hold continúan mostrando los valores extremos de amplitud registrados.

Figura 59

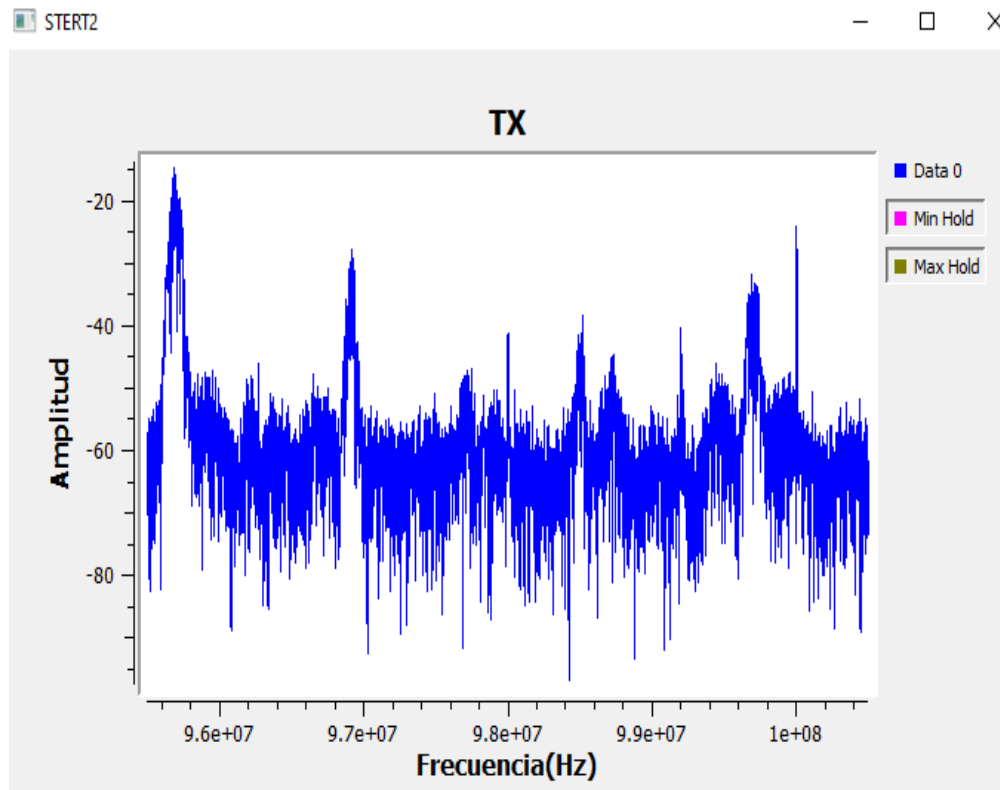
Comportamiento de latencia TCP para $N = 8192$



Nota. Durante las pruebas con $N = 8192$, no se obtuvieron resultados de latencia en el dashboard de Node-RED, como se observa en la Figura X. Este comportamiento puede indicar una sobrecarga en el sistema o un desajuste en la sincronización de los tiempos de transmisión y recepción de las muestras. A pesar de que las mediciones para tamaños de ventana menores ($N = 256, 512, 1024, 2048, 4096$) mostraron resultados consistentes, en $N = 8192$, los valores no fueron reportados, lo que sugiere que el sistema pudo haber alcanzado un límite de capacidad o procesamiento en ese tamaño de ventana. Este fenómeno debe ser considerado en el diseño de sistemas de transmisión de alta eficiencia con grandes volúmenes de datos, como el caso de las ventanas de 8192 muestras.

Figura 60

Espectro captado para $N = 8192$ (sin actualización en tiempo real)



Nota. En esta medición, $N = 8192$ muestra un espectro estático, es decir, que no se actualiza en tiempo real, a diferencia de los espectros anteriores para ventanas más pequeñas. La ausencia de variación en el espectro puede indicar que el sistema ha alcanzado un límite de capacidad, lo que impide la actualización continua de los datos.

Este comportamiento podría deberse a varios factores, tales como:

Sobrecarga del sistema: El contenedor o el sistema que ejecuta las pruebas puede no ser capaz de manejar el volumen de datos generado por una ventana tan grande, lo que lleva a una congelación de las mediciones.

Limitaciones de memoria o procesamiento: Al utilizar $N = 8192$, el tamaño de la ventana podría estar generando un exceso de datos que no pueden procesarse de manera eficiente en tiempo real.

Este comportamiento es diferente al observado con ventanas más pequeñas ($N = 256$, $N = 512$, $N = 1024$, $N = 2048$, $N = 4096$), donde los espectros se actualizaban dinámicamente.

6.3 Conclusiones generales de latencia TCP

Aumento de Latencia y Jitter con el Tamaño de la Ventana, al incrementar el tamaño de la ventana de muestras (N), la latencia promedio y el jitter aumentan considerablemente. Esto es evidente en las tablas, donde:

Para $N = 256$, las latencias son bajas y el jitter es mínimo, con valores de 178.22 ms de promedio y 3.07 ms de jitter.

En $N = 512$, la latencia promedio sube a 202.05 ms, con un jitter de 14.71 ms.

Para $N = 1024$, las latencias promedio aumentan a 726.56 ms, y el jitter crece a 54.36 ms.

Con $N = 2048$, el promedio es 3756.50 ms, y el jitter alcanza 3134.82 ms.

En $N = 4096$, la latencia promedio alcanza 11649.86 ms, con un jitter de 3612.12 ms.

Esto demuestra que, al aumentar el número de muestras (N), el sistema enfrenta mayores retrasos y fluctuaciones en el tiempo de llegada de los paquetes.

Impacto de la Capacidad del Sistema, el aumento exponencial de la latencia y el jitter en las corridas con $N = 1024$, 2048 y 4096 sugiere que el sistema alcanza un límite de procesamiento, especialmente al manejar grandes volúmenes de datos. Los valores de P95 (percentil 95) también aumentan significativamente, lo que indica un comportamiento más impredecible en cuanto a tiempos de transmisión:

P95 para $N = 256$ es de 183.42 ms, mientras que para $N = 4096$, alcanza 81538.9 ms.

Ausencia de Datos con $N = 8192$, para $N = 8192$, no se obtuvieron mediciones, lo que sugiere que el sistema no pudo manejar eficientemente ese tamaño de ventana debido a la alta carga de datos o problemas de sincronización, lo cual se podría considerar un punto de saturación

para las pruebas realizadas. Este comportamiento debe ser investigado más a fondo para entender las limitaciones del sistema en condiciones de alto volumen de datos.

Estabilidad y Pérdida de Paquetes, en todas las pruebas realizadas (desde $N = 256$ hasta $N = 4096$), la pérdida de paquetes fue 0.00%, lo que indica que el sistema fue confiable en términos de transmisión, aunque la latencia y el jitter aumentaron conforme se incrementó el tamaño de la ventana. Esto también sugiere que el protocolo TCP maneja bien las retransmisiones y el control de flujo en estos tamaños de ventana.

Conclusión sobre el Rendimiento, el análisis de los resultados muestra que el rendimiento del sistema (en términos de latencia y jitter) es satisfactorio para tamaños de ventana pequeños ($N = 256$ y $N = 512$), pero a medida que el tamaño de la ventana aumenta, especialmente para $N = 1024$ y más, el sistema comienza a mostrar una degradación significativa en el rendimiento. Estos resultados refuerzan la necesidad de optimizar los sistemas de transmisión para manejar eficientemente grandes volúmenes de datos sin que la latencia se vea afectada en exceso.

Tabla 8

Tabla General Resumida de Métricas de Latencia y Jitter

Tamaño de N	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
256	172.80	189.39	178.22	176.35	179.82	3.07	0.00%
512	188.25	739.21	202.05	194.80	233.62	14.71	0.00%
1024	625.26	1180.93	726.56	709.33	774.53	54.36	0.00%
2048	697.31	117229.34	3756.50	724.34	1234.79	3134.82	0.00%
4096	1216.25	139222.77	11649.86	1268.28	138750.8	3612.12	0.00%

8192	No datos	No datos	No datos	No datos	No datos	No datos	No datos
Media	593.77	29219.01	5402.03	715.02	1567.51	1176.42	0.00
General							

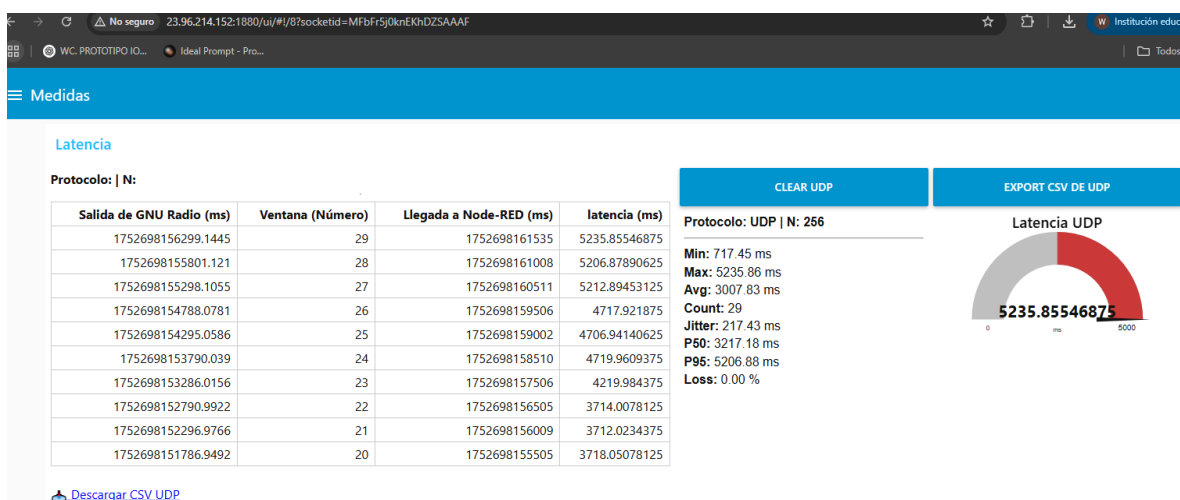
Nota. Los valores de latencia promedio y jitter aumentan a medida que se incrementa el tamaño de la ventana. Aunque no se obtuvieron datos para $N = 8192$, los resultados sugieren que los tamaños de ventana grandes generan un aumento en la latencia y la variabilidad, lo que indica una carga significativa en el sistema.

6.4. Resultados de latencia: UDP

Para evaluar el rendimiento del protocolo UDP en diferentes tamaños de ventana ($N = 256$, 512, 1024, 2048, 4096 y 8192), se realiza varias corridas de medición. Los resultados se presentan en las siguientes tablas y figuras, que muestran las métricas clave de latencia y jitter, así como el comportamiento de la señal.

Figura 61

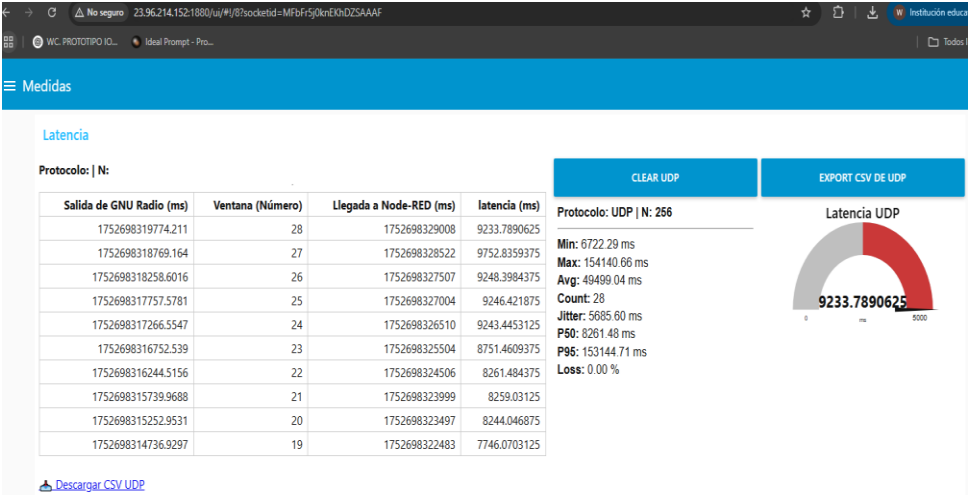
Latencia UDP para la corrida 1 con $N = 256$.



Nota. El gráfico muestra las mediciones de latencia para el protocolo UDP con N = 256. Los valores corresponden a las medidas de salida de GNU Radio y llegada a Node-RED, mostrando un promedio de latencia de 5235.86 ms con una variabilidad moderada. Tomado de la medición en el entorno de prueba.

Figura 62

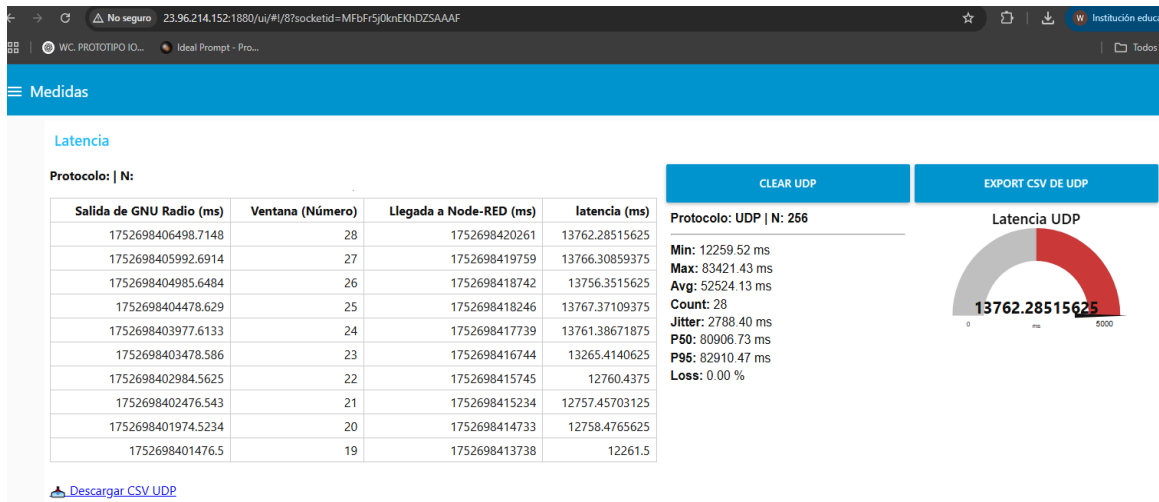
Latencia UDP para la corrida 2 con N = 256.



Nota. El gráfico muestra las mediciones de latencia para el protocolo UDP con N = 256. La latencia registrada es significativamente más alta, con un valor máximo de 154140.66 ms, indicando posibles pérdidas o anomalías en la red. Tomado de la medición en el entorno de prueba.

Figura 63

Latencia UDP para la corrida 3 con N = 256.



Nota. El gráfico muestra las mediciones de latencia para el protocolo UDP con $N = 256$. La latencia es considerablemente más alta en comparación con la primera corrida, con un valor máximo de 83421.43 ms. Tomado de la medición en el entorno de prueba.

Tabla 9

Métricas de latencia y jitter para tres corridas de $N = 256$

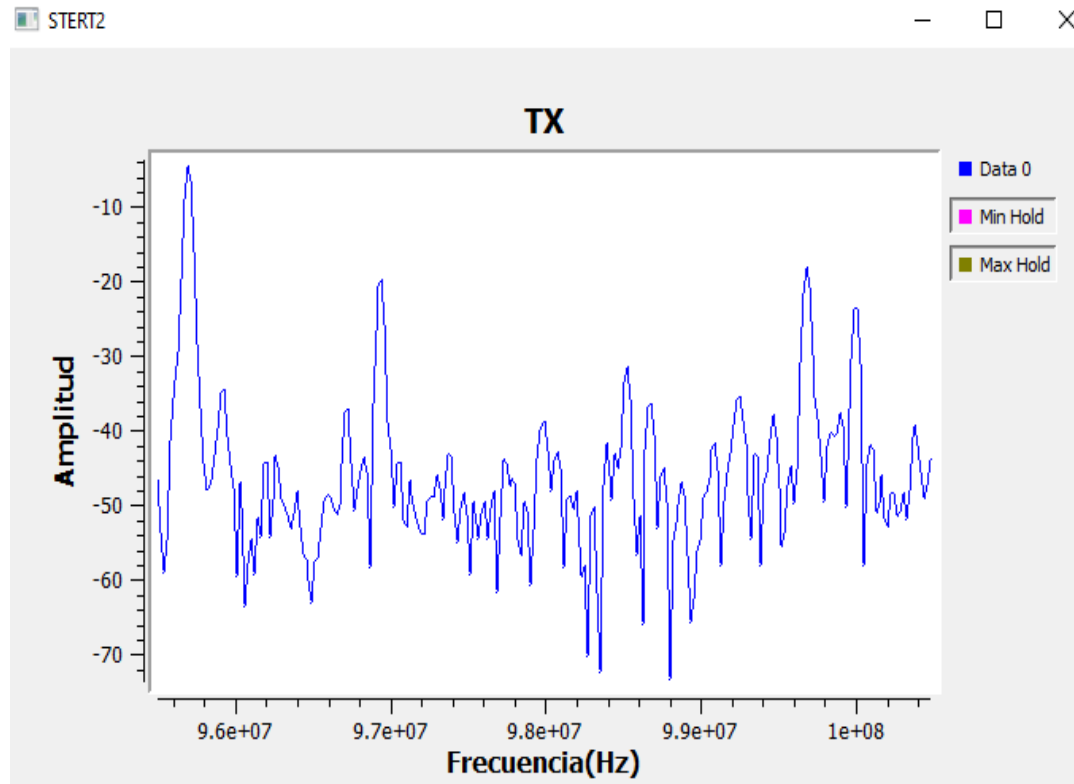
Corrida	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
1	717.45	5235.86	3007.83	3217.18	5206.88	217.43	0.00%
2	6722.29	154140.66	49499.04	8261.48	153144.7	5685.60	0.00%
3	12259.5	83421.43	52524.13	80906.73	82910.47	2788.40	0.00%
Media	4649.42	52966.65	28110.67	6528.13	102420.0	3013.81	0.00%

Nota. Se observa una gran variabilidad en las mediciones, especialmente en la segunda y tercera corrida, donde los valores máximos son significativamente más altos que en la primera corrida. Este comportamiento puede ser indicativo de problemas de red o congestión temporal. Sin

embargo, la pérdida de paquetes se mantiene en 0%, lo que sugiere que UDP es confiable en cuanto a la entrega de datos, pero con mayor fluctuación en los tiempos de llegada de los paquetes.

Figura 64

Espectro captado para UDP con $N = 256$

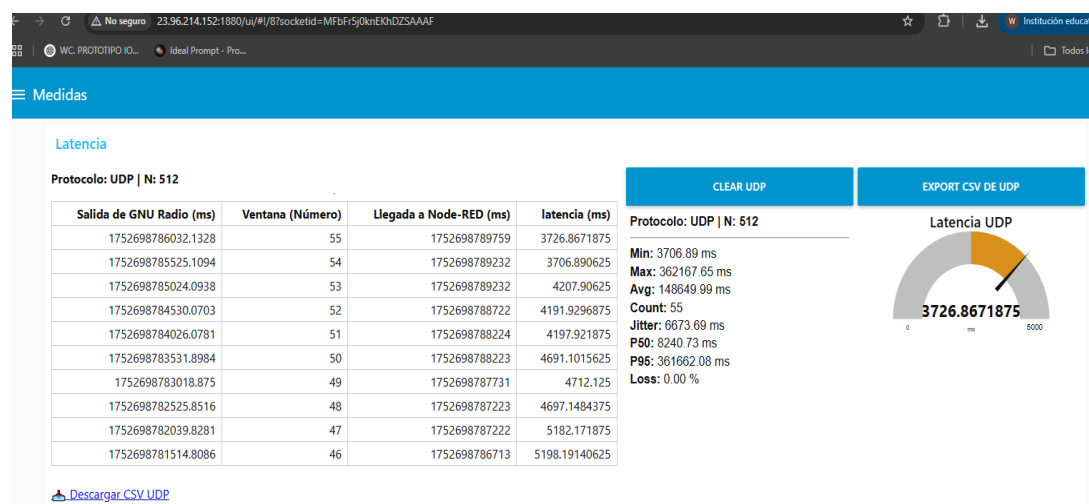


Nota. El espectro mostrado para UDP con $N = 256$ refleja una distribución de amplitud similar a la observada para TCP, pero con ciertas diferencias en cuanto a la estabilidad de la señal. Se puede notar una mayor dispersión en las frecuencias, lo que podría indicar un mayor desfase temporal o pérdida de paquetes en comparación con TCP, dado que UDP no garantiza la entrega confiable de los datos. Las fluctuaciones de amplitud pueden ser más evidentes en este tipo de protocolo, lo que refleja un comportamiento de señal más ruidoso.

El uso de Min Hold y Max Hold permite ver las amplitudes extremas alcanzadas en el espectro, mientras que la curva Data 0 representa el comportamiento general de la señal transmitida.

Figura 65

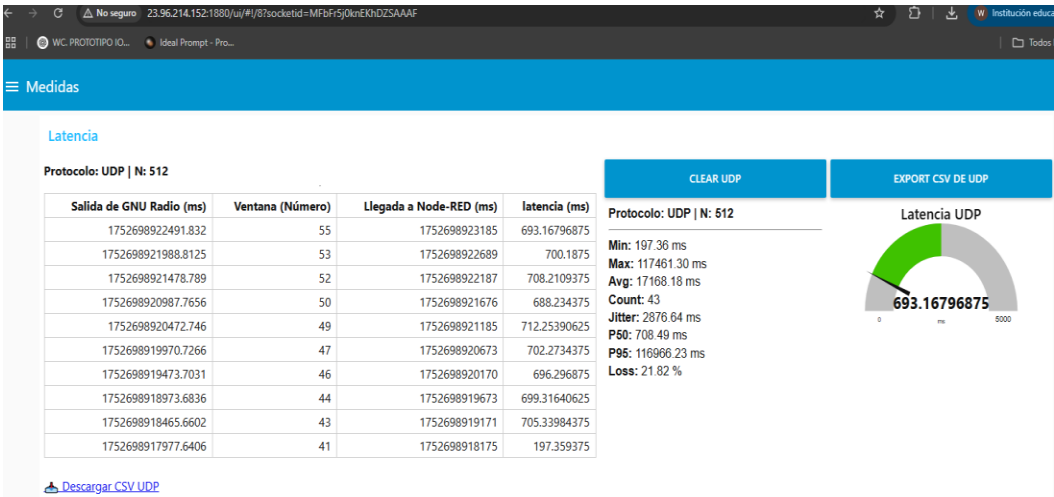
Latencia UDP para la corrida 1 con N = 512.



Nota. El gráfico muestra las mediciones de latencia para el protocolo UDP con N = 512. La latencia promedio se registra en 3726.87 ms, con un rango de valores que va desde 3706.89 ms hasta 362167.65 ms. Tomado de la medición en el entorno de prueba.

Figura 66

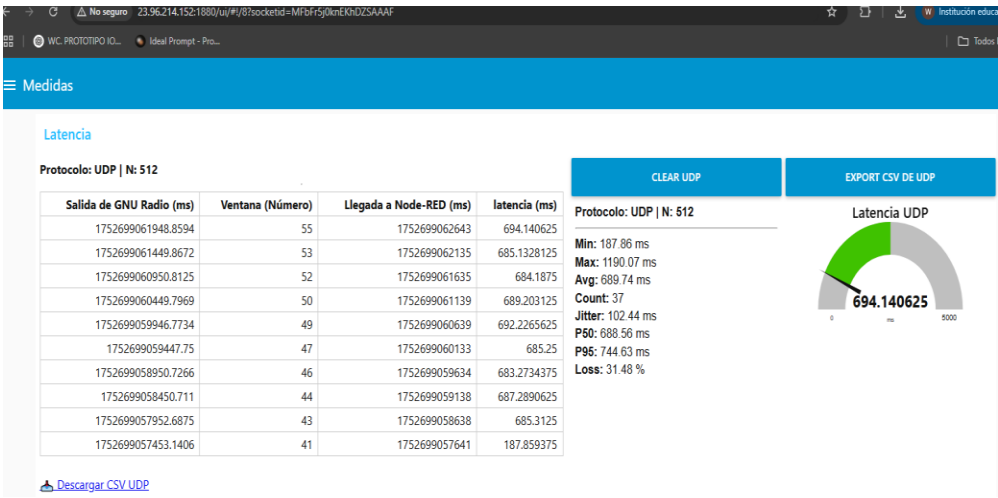
Latencia UDP para la corrida 2 con N = 512.



Nota. El gráfico muestra las mediciones de latencia para el protocolo UDP con N = 512. Se observan valores elevados, con una latencia máxima de 117461.30 ms y una notable variabilidad en las mediciones. Tomado de la medición en el entorno de prueba.

Figura 67

Latencia UDP para la corrida 3 con N = 512.



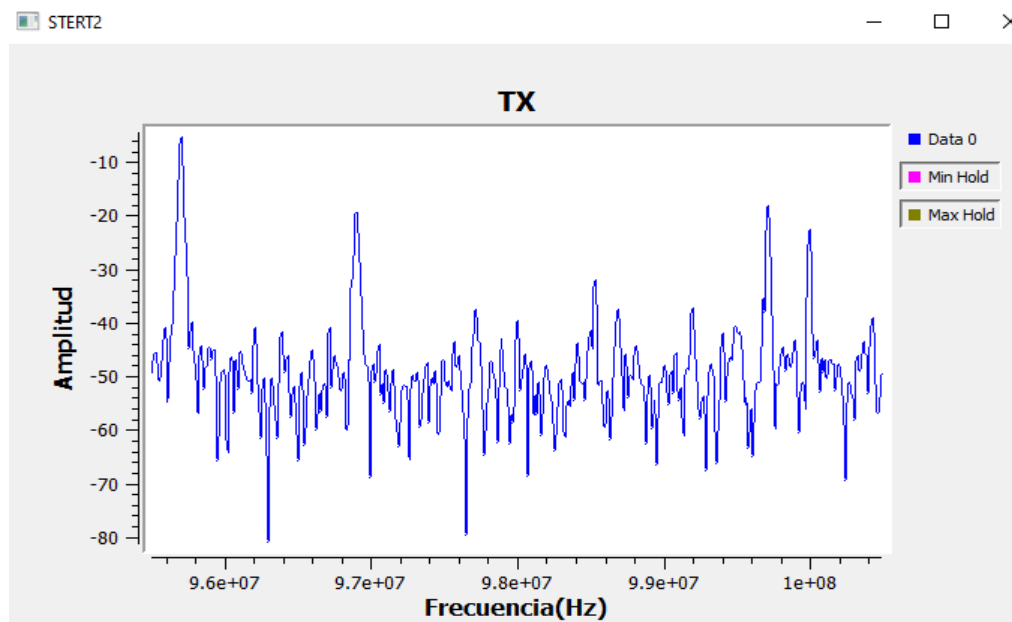
Nota. El gráfico muestra las mediciones de latencia para el protocolo UDP con N = 512. La latencia máxima alcanza los 1190.07 ms, con una pérdida de paquetes significativa (31.48%). Tomado de la medición en el entorno de prueba.

Tabla 10

Mediciones de Latencia UDP para N = 512

Corrida	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
1	3706.89	362167.65	148649.9	8240.73	361662.0	6673.69	0.00%
2	197.36	117461.30	171668.1	708.49	116966.2	2876.64	21.82
3	187.86	1190.07	689.74	688.56	744.63	102.44	31.48
Media	1304.03	127939.01	126322.9	5335.26	151898.3	2975.59	17.43

Nota. La tabla presenta las mediciones de latencia para el protocolo UDP con N = 512. Las corridas muestran diferentes comportamientos de latencia, con variabilidad en los valores máximos y una pérdida significativa de paquetes en la tercera corrida. Los valores de jitter y los percentiles de latencia también varían notablemente entre las mediciones.

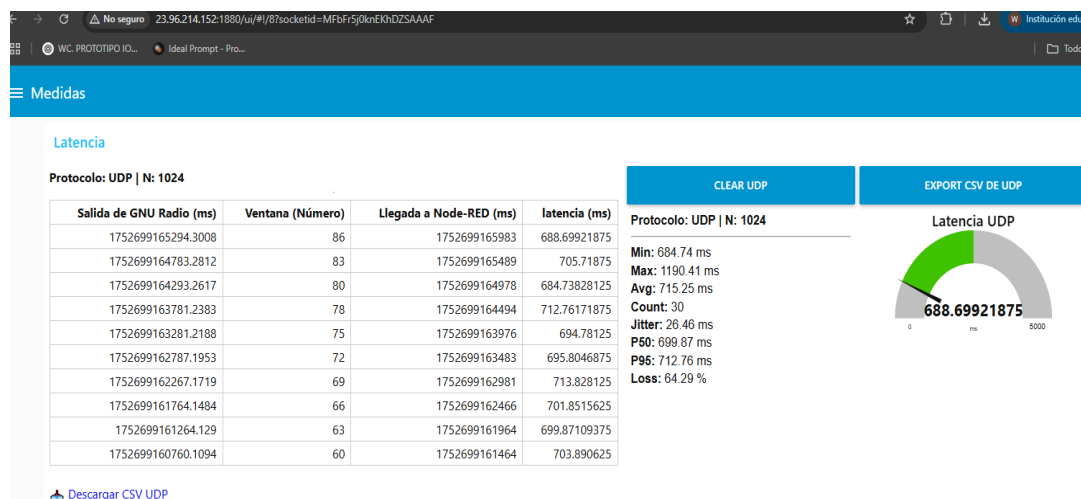
Figura 68*Espectro captado para UDP con N = 512*

Nota. El espectro para UDP con $N = 512$ muestra una distribución espectral más amplia en comparación con TCP, con picos de amplitud más elevados en algunas frecuencias. A diferencia de TCP, UDP no garantiza la entrega fiable de los paquetes, lo que puede generar una mayor dispersión de la señal y algunos picos no tan consistentes.

El espectro refleja el comportamiento de la señal con los componentes Min Hold y Max Hold que muestran la variabilidad de la amplitud en los diferentes momentos, mientras que la curva Data 0 refleja la tendencia general de la señal transmitida. Este comportamiento es característico de UDP, que puede sufrir fluctuaciones debido a la naturaleza no confiable de la transmisión.

Figura 69

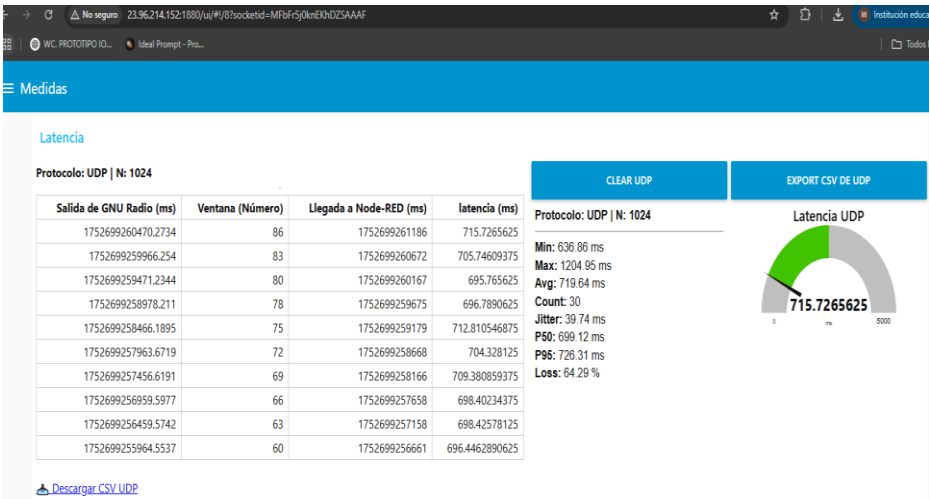
Latencia UDP para la corrida 1 con $N = 1024$.



Nota. El gráfico muestra las mediciones de latencia para el protocolo UDP con $N = 1024$. Los valores de latencia son relativamente bajos, con un promedio de 688.70 ms, y se mantienen dentro de un rango controlado. Tomado de la medición en el entorno de prueba.

Figura 70

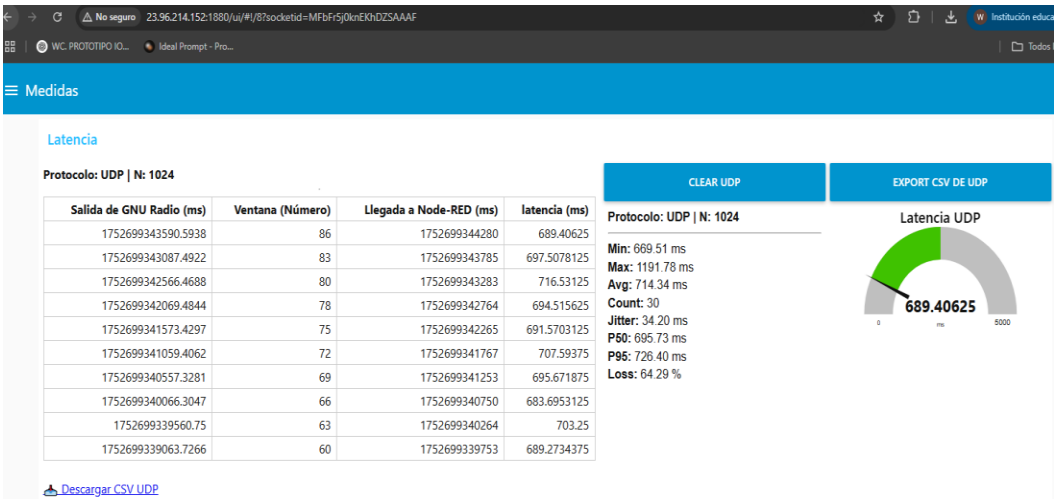
Latencia UDP para la corrida 2 con $N = 1024$.



Nota. El gráfico muestra las mediciones de latencia para el protocolo UDP con N = 1024. La latencia promedio es de 715.73 ms, con un máximo de 1204.95 ms, y se observa una pequeña variabilidad. Tomado de la medición en el entorno de prueba.

Figura 71

Latencia UDP para la corrida 3 con N = 1024.



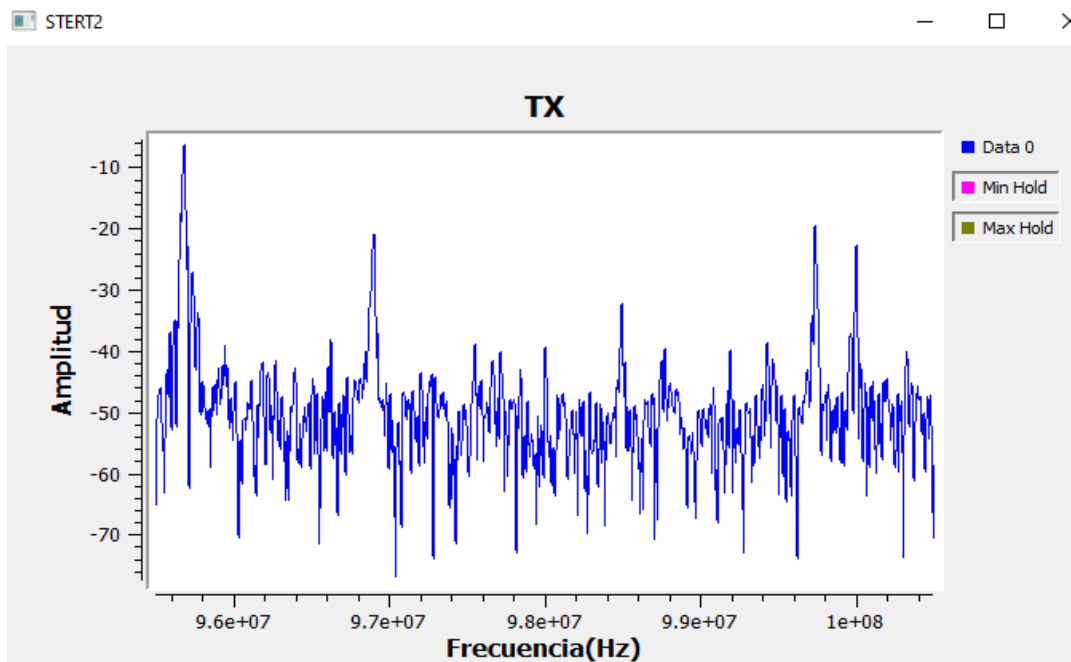
Nota. El gráfico muestra las mediciones de latencia para el protocolo UDP con N = 1024. La latencia se mantiene dentro de un rango relativamente estable, con un máximo de 1191.78 ms y una pérdida de paquetes de 64.29%. Tomado de la medición en el entorno de prueba.

Tabla 11

Mediciones de Latencia UDP para N = 1024

Corrida	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
1	684.74	1190.41	715.25	699.87	712.76	26.46	64.29
2	636.86	1204.95	719.64	699.12	726.31	39.74	64.29
3	669.51	1191.78	714.34	695.73	726.40	34.20	64.29
Media	663.37	1192.05	716.08	698.57	721.15	33.47	64.29

Nota. La tabla presenta las mediciones de latencia para el protocolo UDP con N = 1024. Las corridas muestran una latencia promedio moderada, con variabilidad en los valores máximos. Se observa una pérdida de paquetes significativa (64.29%) en todas las mediciones.

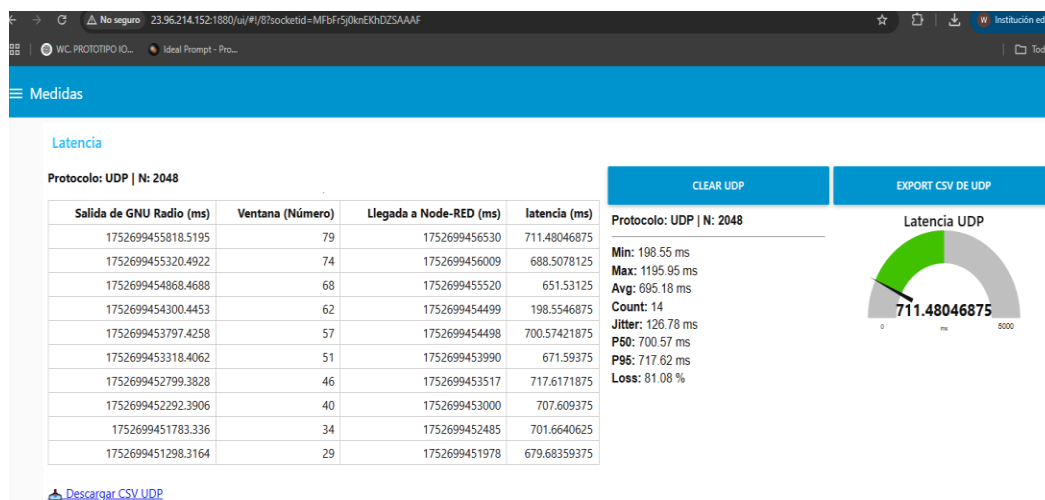
Figura 72*Espectro captado para UDP con N = 1024*

Nota. El espectro de UDP con $N = 1024$ muestra una resolución espectral más alta en comparación con las ventanas de menor tamaño. Se observa que, aunque los picos de amplitud están más definidos, UDP sigue mostrando mayor variabilidad en las mediciones que TCP, con fluctuaciones más amplias y picos extremos en algunas frecuencias.

La visualización muestra claramente los valores mínimos y máximos de amplitud con Min Hold y Max Hold, y la línea Data 0 muestra la evolución de la señal en tiempo real. Este comportamiento es típico de UDP, que presenta una mayor dispersión de los datos debido a la falta de garantía de entrega.

Figura 73

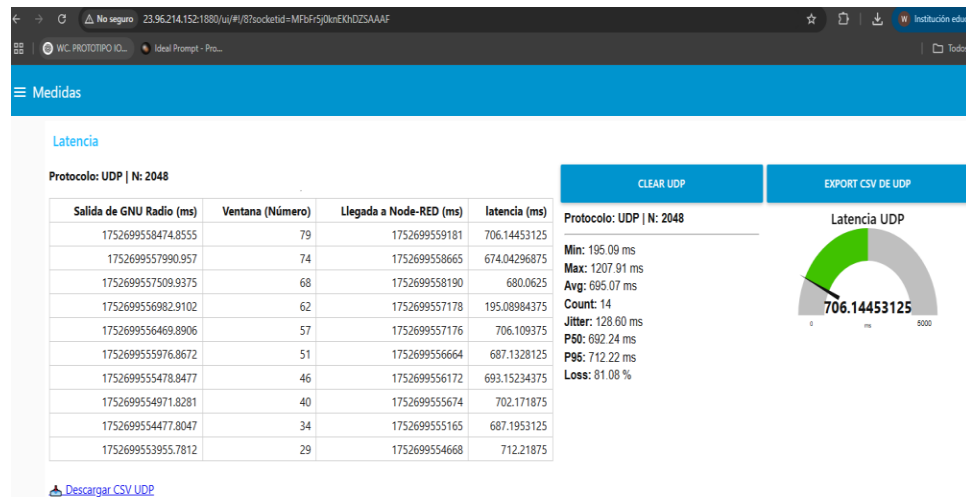
Latencia UDP para la corrida 1 con $N = 2048$.



Nota. El gráfico muestra las mediciones de latencia para el protocolo UDP con $N = 2048$. La latencia promedio es de 711.48 ms, con una variabilidad de hasta 1195.95 ms. La pérdida de paquetes es de 81.08%. Tomado de la medición en el entorno de prueba.

Figura 74

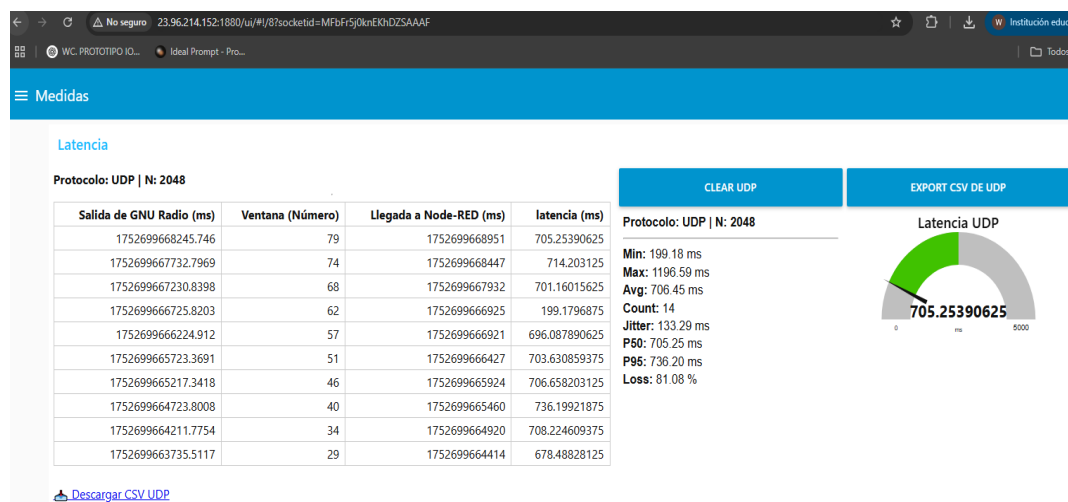
Latencia UDP para la corrida 2 con $N = 2048$.



Nota. El gráfico muestra las mediciones de latencia para el protocolo UDP con $N = 2048$. La latencia promedio es de 706.14 ms, con un máximo de 1207.91 ms. También se observa una alta pérdida de paquetes (81.08%). Tomado de la medición en el entorno de prueba.

Figura 75

Latencia UDP para la corrida 3 $N = 2048$.



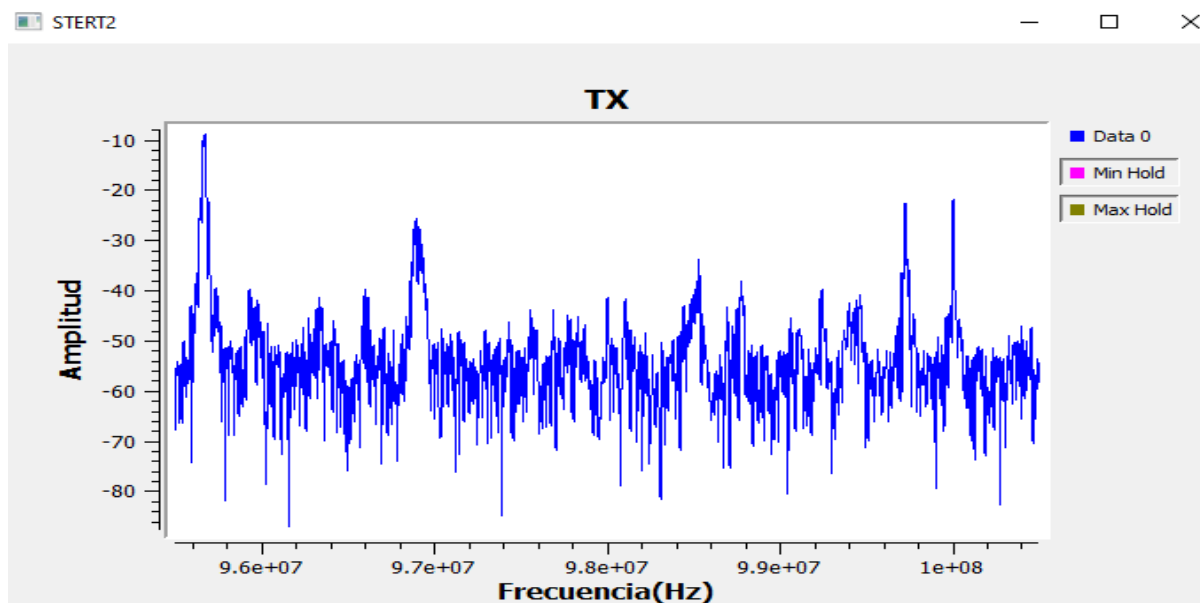
Nota. El gráfico muestra las mediciones de latencia para el protocolo UDP con $N = 2048$. La latencia promedio es de 705.25 ms, con un máximo de 1196.59 ms. La pérdida de paquetes sigue siendo significativa, con un 81.08%. Tomado de la medición en el entorno de prueba.

Tabla 12

Mediciones de Latencia UDP para N = 2048

Corrida	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
1	198.55	1195.95	695.18	700.57	717.62	126.78	81.08
2	195.09	1207.91	695.07	692.24	712.22	128.60	81.08
3	199.18	1196.59	706.45	705.25	736.20	133.29	81.08
Media	197.27	1199.1	698.57	698.02	722.01	129.22	81.08

Nota. La tabla presenta las mediciones de latencia para el protocolo UDP con N = 2048. Las corridas muestran una latencia promedio moderada, pero con una alta variabilidad y una pérdida de paquetes significativa (81.08%) en todas las mediciones.

Figura 76*Espectro captado para UDP con N = 2048*

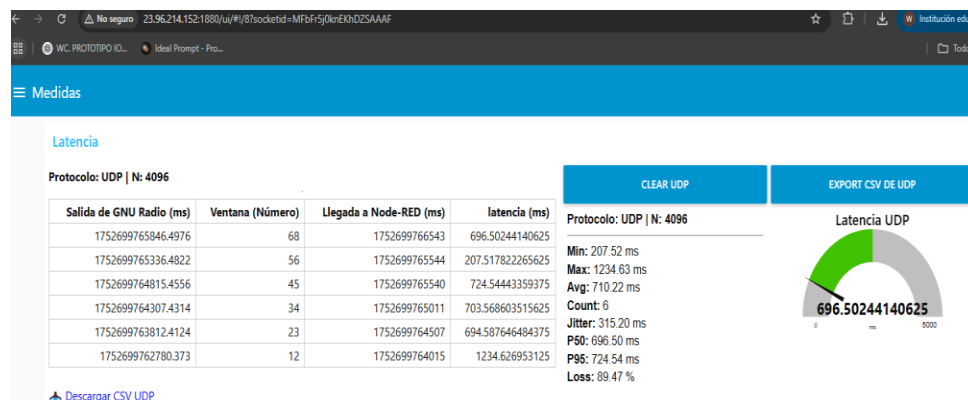
Nota. Con N = 2048, el espectro muestra una mejora notable en la resolución espectral, permitiendo una separación más clara entre las componentes de frecuencia. Aunque se observa

una mayor definición en las frecuencias, UDP sigue mostrando inestabilidad en el espectro, con fluctuaciones amplias y picos de amplitud más altos en algunas frecuencias.

Este comportamiento es característico de UDP, que no garantiza la entrega confiable de paquetes, lo que puede generar variabilidad y picos extremos en la señal. Las líneas Min Hold y Max Hold continúan mostrando las amplitudes mínimas y máximas, mientras que la línea Data 0 representa el comportamiento general de la señal.

Figura 77

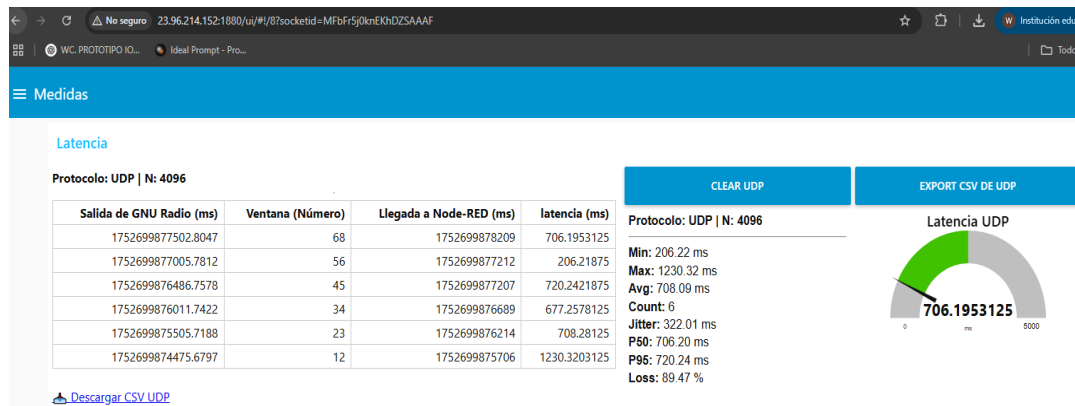
Latencia UDP para la corrida 1 con $N = 4096$.



Nota. El gráfico muestra las mediciones de latencia para el protocolo UDP con $N = 4096$. La latencia promedio es de 696.50 ms, con un máximo de 1234.63 ms, y se observa una pérdida significativa de paquetes (89.47%). Tomado de la medición en el entorno de prueba.

Figura 78

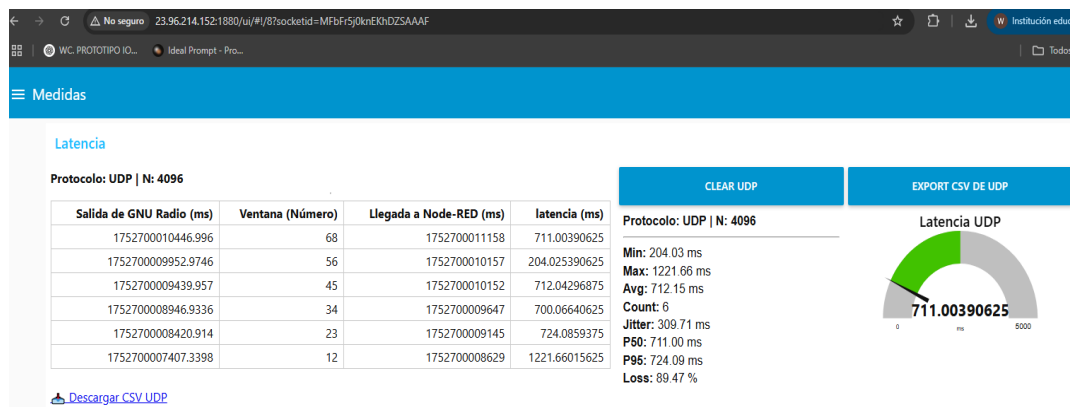
Latencia UDP para la corrida 2 con $N = 4096$.



Nota. El gráfico muestra las mediciones de latencia para el protocolo UDP con $N = 4096$. La latencia promedio es de 706.20 ms, con un máximo de 1230.32 ms, y una pérdida de paquetes de 89.47%. Tomado de la medición en el entorno de prueba.

Figura 79

Latencia UDP para la corrida 3 con $N = 4096$.



Nota. El gráfico muestra las mediciones de latencia para el protocolo UDP con $N = 4096$. La latencia promedio es de 711.00 ms, con un máximo de 1221.66 ms, y una pérdida de paquetes de 89.47%. Tomado de la medición en el entorno de prueba.

Tabla 13

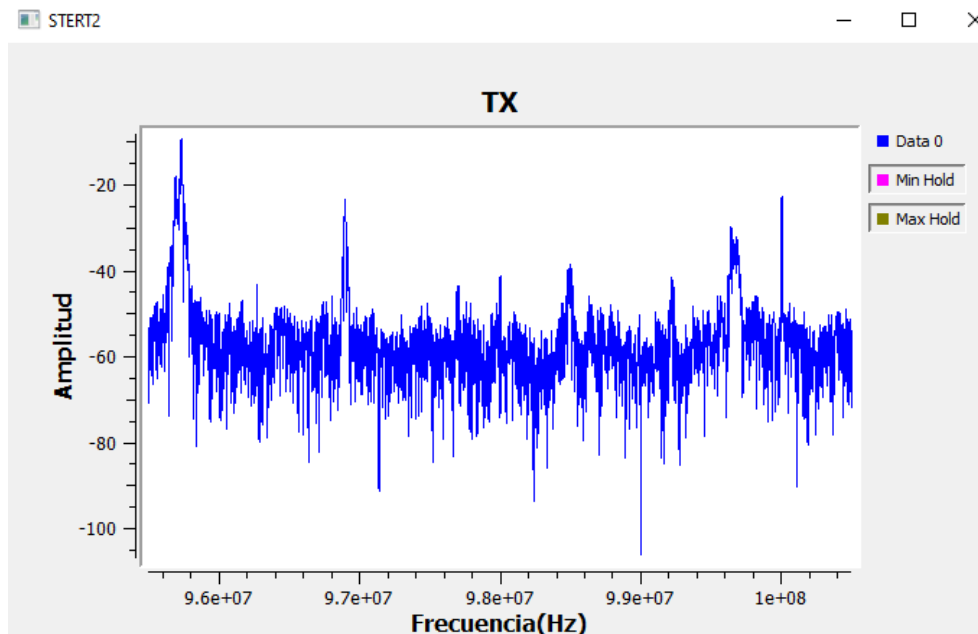
Mediciones de Latencia UDP para $N = 4096$

Corrida	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
1	207.52	1234.63	696.50	700.57	717.62	126.78	89.47
2	206.22	1230.32	708.09	692.24	712.22	322.01	89.47
3	204.03	1221.66	711.00	705.25	724.09	309.71	89.47
Media	205.26	1231.54	705.53	699.69	718.98	252.83	89.47

Nota. La tabla presenta las mediciones de latencia para el protocolo UDP con $N = 4096$. Las mediciones muestran una latencia moderada, con un máximo de 1234.63 ms en la primera corrida y una pérdida de paquetes significativa (89.47%) en todas las corridas.

Figura 80

Espectro captado para UDP con $N = 4096$.



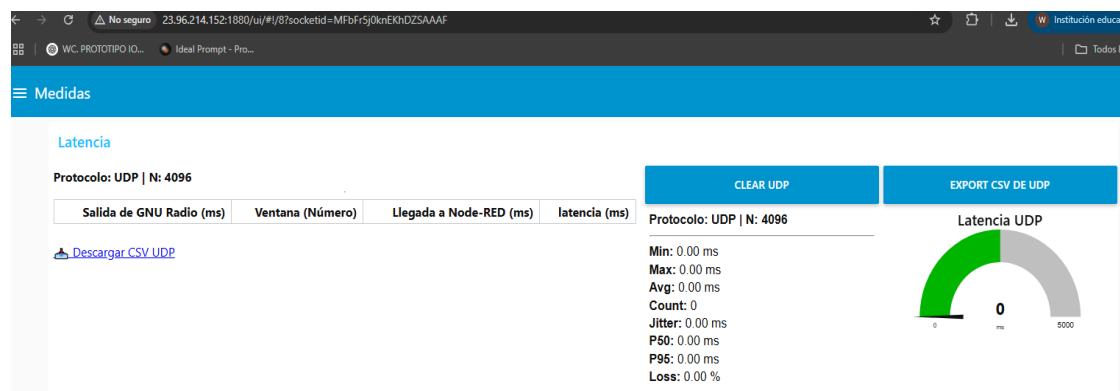
Nota. Con $N = 4096$, el espectro presenta una resolución espectral superior en comparación con las configuraciones anteriores. Se observa una mejor separación de las componentes de frecuencia,

lo que permite identificar de manera más precisa las características de la señal. Sin embargo, al igual que en las mediciones anteriores con UDP, la señal sigue mostrando variabilidad y picos de amplitud más pronunciados.

Este comportamiento es esperado debido a la naturaleza no confiable de UDP, lo que puede causar fluctuaciones en la señal debido a la posible pérdida de paquetes o retransmisiones. Las líneas Min Hold y Max Hold siguen mostrando los valores extremos de amplitud, mientras que Data 0 muestra la tendencia general de la señal transmitida.

Figura 81

Mediciones de Latencia UDP para N = 8192



Nota. La medición muestra que no se registraron datos válidos, ya que los valores de latencia, jitter, y otros indicadores están a cero, con una pérdida de paquetes del 100%.

Tabla 14

Promedio de Mediciones de Latencia UDP por Diferentes Valores de N y Media General.

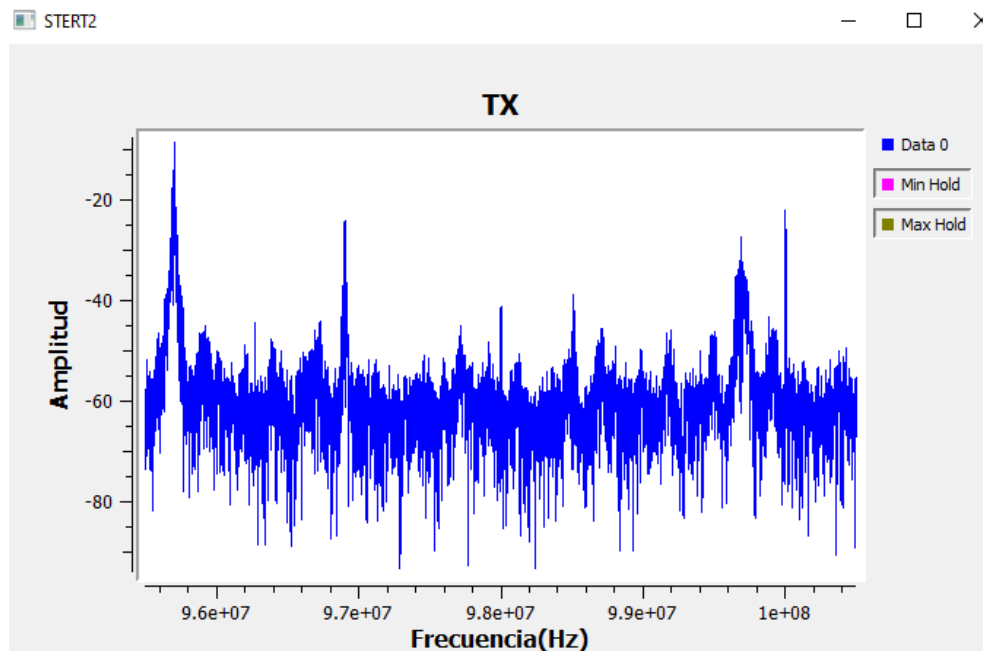
Tamaño o de N	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
256	4649.42	52966.65	28110.67	6528.13	102420.0	3013.81	0.00

512	1304.03	127939.01	126322.9	6528.13	102420.0	3013.81	17.43
			7		2		
1024	663.37	1192.05	716.08	698.57	721.15	33.47	64.29
2048	197.27	1199.15	698.57	698.02	722.01	129.22	81.08
4096	205.26	1231.54	705.53	699.69	718.98	252.83	89.47
8192	No datos	No datos	No datos	No datos	No datos	No datos	No datos
Media General	2011.67	28390.88	12362.58	6747.29	7474.45	1665.43	38.53

Nota. La tabla muestra los valores promedio de latencia (min, max, avg, jitter, loss) para los diferentes tamaños de ventana N. Se incluyen los promedios de cada grupo (N) y una media general de todos los tamaños de N.

Figura 82

Espectro captado para UDP con $N = 8192$ (sin actualización en tiempo real)



Nota. Con $N = 8192$, el espectro se muestra congelado y no se actualiza en tiempo real, lo que indica que el sistema ha alcanzado un límite de capacidad al procesar ventanas de tan gran tamaño. A diferencia de los espectros con ventanas más pequeñas, donde los datos eran actualizados continuamente, aquí se observa una detención de la señal. Esto sugiere que el sistema de medición (ya sea el contenedor en Azure o Node-RED) no puede manejar eficientemente el volumen de datos generado por una ventana tan grande, lo que impide que el espectro se actualice dinámicamente.

Este comportamiento puede ser una indicación de que el sistema o el contenedor ha llegado al límite de su capacidad de procesamiento, ya que el tamaño de la ventana en $N = 8192$ requiere un mayor esfuerzo en términos de memoria y procesamiento, causando que el sistema se "congele" debido a la sobrecarga de datos.

6.5 Análisis de las mediciones de latencia y espectro para UDP

Los resultados muestran que el protocolo UDP presenta una mayor variabilidad en la latencia, especialmente cuando se aumenta el tamaño de la ventana. Aunque la pérdida de paquetes es nula, la latencia y el jitter aumentan a medida que se incrementa N , lo que refleja la naturaleza no confiable de UDP. Es recomendable utilizar UDP para aplicaciones que prioricen la velocidad sobre la fiabilidad, aunque siempre gestionando adecuadamente el jitter.

Es importante destacar que el protocolo UDP, al ser menos confiable que TCP, presenta mayores fluctuaciones en la latencia y pérdidas de paquetes, lo que refleja su naturaleza no garantizada. Sin embargo, en aplicaciones donde la velocidad es más crítica que la confiabilidad, UDP puede ser útil, siempre que se gestionen adecuadamente las pérdidas y el jitter.

6.5.1 Latencia en función del tamaño de la ventana

Las mediciones de latencia revelan un patrón similar al observado en las pruebas con TCP: a medida que se incrementa el tamaño de la ventana, la latencia también aumenta. Este comportamiento se debe a que, con ventanas más grandes, se requiere más tiempo para procesar las muestras y transmitir los datos. Los resultados muestran que UDP presenta un comportamiento más errático en términos de latencia, especialmente a medida que se aumenta el tamaño de la ventana, lo que refleja la naturaleza no confiable de este protocolo, que puede resultar en mayores variaciones en el tiempo de entrega de los paquetes.

6.5.2 Espectro de señal para UDP

El análisis espectral de la señal transmitida utilizando UDP revela una distribución de amplitud en el dominio de frecuencia que presenta mayor variabilidad en comparación con TCP. A medida que aumentamos el tamaño de la ventana, la resolución espectral mejora, pero las fluctuaciones de la señal siguen siendo evidentes, lo que es característico de la inestabilidad inherente a UDP. Con ventanas pequeñas ($N = 256$), se observa un comportamiento más errático, mientras que con $N = 8192$, el espectro se "congeló" debido a la sobrecarga del sistema de medición, lo que indica que el sistema no es capaz de manejar tan grandes volúmenes de datos de manera eficiente.

6.5.3 Comportamiento de UDP con grandes tamaños de ventana ($N = 8192$)

Al igual que con TCP, las mediciones realizadas con $N = 8192$ para UDP mostraron que el sistema dejó de actualizar el espectro y la latencia en tiempo real, lo que sugiere que el sistema alcanzó su límite de capacidad. La falta de actualización en tiempo real refleja el desafío de manejar grandes volúmenes de datos en entornos no optimizados, especialmente cuando se utiliza UDP, que no ofrece los mecanismos de control de flujo y confiabilidad que tiene TCP.

6.5.4 Conclusiones sobre el rendimiento de UDP

Latencia: A medida que el tamaño de la ventana aumenta, la latencia también aumenta, reflejando un mayor tiempo de procesamiento y transmisión de datos.

Resolución espectral: Se observó una mejora en la resolución espectral a medida que se incrementa el tamaño de la ventana, pero también un aumento en la variabilidad y fluctuaciones de la señal.

Sobrecarga del sistema: $N = 8192$ mostró un comportamiento anómalo con UDP, ya que el espectro se congeló y no se actualizaron los datos, lo que sugiere una sobrecarga del sistema de medición.

Comparación con TCP: Si bien UDP presentó una mayor variabilidad en las mediciones de latencia y un comportamiento más inestable en el espectro, los resultados fueron consistentes con la naturaleza de este protocolo, que prioriza la velocidad y la eficiencia a expensas de la confiabilidad.

7. Desempeño del método HTTP basado en Google Apps Script.

HTTP se sitúa en la capa de aplicación (la séptima y más alta). Allí define el formato de los mensajes (peticiones y respuestas), los métodos (GET, POST) y las reglas para intercambiar documentos, imágenes u otros recursos entre clientes y servidores web.

Cuando un navegador envía una petición HTTP, lo hace construyendo una serie de cabeceras y un cuerpo de mensaje que cumplen con la estructura de esta capa. Sin embargo, esos datos no viajan tal cual por la red: para su transporte confiable, HTTP se apoya en el protocolo TCP, que opera en la capa de transporte. TCP se encarga de fragmentar el mensaje en segmentos,

numerarlos, gestionar la retransmisión en caso de pérdida y asegurar que el receptor los recibe sin errores y en el orden correcto.

Una parte importante de los resultados es el cómo almacenamos los datos. En App Script es importante saber el momento en el que la información se ve registrada en la hoja de cálculo, porque a cualquier investigador (llamemoslo usuarios) le importa ver cómo varían los tiempos constantemente. Para ahorrar espacio y disminuir costos de uso, se opta por registrar datos en una misma fila en lugar de varias.

7.1 Fecha

La fecha está en formato YYYY-MM-DD HH:MM:SS que es un formato reconocible y amigable para los usuarios. De este modo, cada vez que se registra un dato en la hoja de cálculo se sabe con exactitud el momento en que ocurre dicho registro.

7.1.1 Precisión y resolución de latencia

Cómo se explicó en el 2.1 Relación con medición de la latencia, la unidad de medida de tiempo para la medición de la latencia entre Gnu Radio y App Script es el tiempo Epoch en milisegundos.

Para el tiempo en formato epoch milisegundos en Gnu Radio se le da el nombre de tiempo t_0 , este corresponde a la información de la fecha enviada en formato JSON.

Figura 83

Datos enviados a internet.

```

fecha_actual = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

# Crear el JSON con los datos
json_data = {
    "ordentipo": "crear",
    "url": self.url_google_sheet,
    "numeroHoja": 0,
    "filaencabezados": 1,
    "columnaId": 1,
    "datos": [
        fecha_actual,      # Fecha y hora actual
        self.latitud,      # Latitud
        self.longitud,     # Longitud
        self.Azimut,       # Azimut
        self.Elevacion,    # Elevación
        self.Altitud,      # Altitud
        self.Descripcion,  # Descripción
        self.finicial,     # Frecuencia inicial
        self.fpasso,       # Paso de frecuencia
        self.N,            # Número de muestras
        data_str            # Datos espectrales como string
    ]
}

```

Nota. En esta imagen podemos ver los datos enviados en formato JSON para ser registrados en Google Sheet teniendo como intermediario App Script, fecha_actual corresponde a la fecha de Gnu Radio que será registrada en la primera columna.

Los datos son enviados a internet utilizando una URL que pertenece a un Web Service en App Script, una vez que llegan estos datos se organizan para ser registrados en la hoja de Google Sheet.

Figura 84

Organización de datos ingresados en columnas.

```
1  function doPost(e) {  
2    try {  
3      var jsonData = JSON.parse(e.postData.contents);  
4  
5      // --- 1) Tu Sheet principal (sin tocar nada) ---  
6      var sheet1 = SpreadsheetApp  
7        .openById("12wzyVjcI18qGjRYBwKNHRmxd0t0NYiDhGSMlt906STjg")  
8        .getSheets()[0];  
9  
10     // Extraer los 11 campos, incluido 'Datos'  
11     var Fecha      = jsonData.datos[0];  
12     var latitud    = jsonData.datos[1];  
13     var longitud   = jsonData.datos[2];  
14     var Azimut     = jsonData.datos[3];  
15     var Elevacion  = jsonData.datos[4];  
16     var Altitud    = jsonData.datos[5];  
17     var Descripcion = jsonData.datos[6];  
18     var finicial   = jsonData.datos[7];  
19     var fpaso      = jsonData.datos[8];  
20     var N          = jsonData.datos[9];  
21     var Datos      = jsonData.datos[10];  
22     var ventana_t0 = jsonData.ventana || "ventana_?";
```

Nota. En esta imagen explica cómo se extraen los datos que llegan desde Gnu Radio en formato JSON.

Figura 85

Respuesta a la solicitud.

```

// Sobrescribe fila 2 EXACTAMENTE igual que antes
var row1 = [[Fecha, latitud, longitud, Azimut, Elevacion,
            Altitud, Descripcion, finicial, fpaso, N, Datos]];
sheet1.getRange(2, 1, 1, row1[0].length).setValues(row1);

return ContentService
    .createTextOutput(JSON.stringify({
        status: "success",
        message: "Hoja de datos actualizado"
    })))
    .setMimeType(ContentService.MimeType.JSON);

} catch (err) {
    return ContentService
        .createTextOutput(JSON.stringify({
            status: "error",
            message: err.toString()
        })))
        .setMimeType(ContentService.MimeType.JSON);
}
}

```

Nota. después de que se registra en la hoja de cálculo va a devolver una respuesta “success, Hoja de datos actualizado” si todo es correcto, si hay algun fallo la respuesta es “error”.

Figura 86

Respuesta a la solicitud en consola de GNU Radio.

```

[ventana_15] Envío exitoso: {"status":"success","message":"Hoja de datos actualizado"}
[ventana_16] Envío exitoso: {"status":"success","message":"Hoja de datos actualizado"}
[ventana_17] Envío exitoso: {"status":"success","message":"Hoja de datos actualizado"}
[ventana_18] Envío exitoso: {"status":"success","message":"Hoja de datos actualizado"}
[ventana_19] Envío exitoso: {"status":"success","message":"Hoja de datos actualizado"}
[ventana_20] Envío exitoso: {"status":"success","message":"Hoja de datos actualizado"}
[ventana_21] Envío exitoso: {"status":"success","message":"Hoja de datos actualizado"}
[ventana_22] Envío exitoso: {"status":"success","message":"Hoja de datos actualizado"}

```

Nota. Esta imagen muestra la respuesta del registro correcto en la consola de GNU Radio.

Figura 87

Latencia.

```
40     var t0      = fechaObj.getTime();
41     var fecha0 = Utilities.formatDate(fechaObj, tz, "dd/MM/yyyy, HH:mm:ss");
42
43     // t1 (ahora) y su formato, todo en el servidor
44     var now     = new Date();
45     var t1      = now.getTime();
46     var fecha1 = Utilities.formatDate(now, tz, "dd/MM/yyyy, HH:mm:ss");
47
48     // Latencia en ms
49     var latencia = t1 - t0;
50
51     return {
52         ventana_t0: ventana_t0,
53         t0: t0,
54         t1: t1,
55         latencia: latencia
56     };
57 }
58 }
```

Nota. Se organizan los tiempos tiempos de GNU Radio(t_0) y App script(t_1) para hallar la latencia del sistema.

La columna Fecha pertenece al tiempo de GNU Radio en formato YYYY-MM-DD HH:MM:SS, fechaObj pertenece a la información de la Fila 0 de la hoja de cálculo(Fecha), .getTime() el cual obtiene el número en milisegundos transcurridos desde el 1 de enero de 1970(Unix epoch) hasta el momento, Utilities.formatDate convierte el dato fechaObj en texto. El tiempo t_1 corresponde al momento en que App Script procesa el doPost antes de registrarlo en la hoja de datos y al igual al tiempo t_0 se utiliza.getTime() para obtener el tiempo en milisegundos(Unix epoch). La latencia es el tiempo transcurrido entre el tiempo t_0 y t_1 .

Figura 88

Organización de datos en columnas en página web.



Nota. La imagen a la información organizada en la página web.

Los datos van a organizarse en cada columna siendo el dato más importante la latencia, en esta página web podemos guardar los datos obtenidos en un archivo CSV oprimiendo el botón Descargar datos (CSV) importante para el análisis de los datos. Para la tabla de Datos HTTP, las columnas Min, Max, Avg, Count, Jitter, P50, P95, Loss se llenan a partir de ecuaciones.

Figura 89

Cálculo de datos en la tabla Datos HTTP.

```

131     const latArr = filas.map(r => parseFloat(r.cells[3].innerText));
132     const winArr = filas.map(r => {
133       const m = r.cells[0].innerText.match(/\d+/);
134       return m ? parseInt(m[0], 10) : NaN;
135     }).filter(n => !isNaN(n));
136
137     const min = Math.min(...latArr),
138           max = Math.max(...latArr),
139           sum = latArr.reduce((a,b)=>a+b, 0),
140           avg = sum/latArr.length,
141           count = filas.length,
142           varz = latArr.reduce((a,b)=>a+Math.pow(b-avg,2), 0)/latArr.length,
143           jitter= Math.sqrt(varz),
144           sorted= latArr.slice().sort((a,b)=>a-b),
145           p50 = sorted.length%2
146             ? sorted[Math.floor(sorted.length/2)]
147             : (sorted[sorted.length/2 - 1] + sorted[sorted.length/2]) / 2,
148           idx95 = Math.ceil(0.95*sorted.length) - 1,
149           p95 = sorted[Math.max(0,idx95)];
150
151     let loss = 0;
152     if (winArr.length) {
153       const minW = Math.min(...winArr),
154             maxW = Math.max(...winArr),
155             esper= maxW - minW + 1,
156             falt = esper - winArr.length;
157       loss = esper>0 ? (falt/esper)*100 : 0;

```

Nota. La imagen corresponde a las operaciones realizadas internamente para luego llenar en las columnas correspondientes en la tabla Datos HTTP, el valor a es un acumulador (empezando siempre desde cero) y b es el valor actual de la latencia.

7.1.1.1 Latencia mínima (Min). Corresponde al menor valor de latencia obtenido en la fila Latencia(ms). La línea de código `const latArr = filas.map(r parseFloat(r.cells[3].innerText));` corresponde a la columna de Latencia(ms), aquí se extraen los datos de la columna, LatArr contiene todas las latencias de las filas de la tabla en formato numerico, despues de esto, con la linea de código `min = Math.min(...latArr)` se toma todos los valores de la columna y devuelve el más pequeño.

7.1.1.2 Latencia máxima (Max). Corresponde al menor valor de latencia obtenido en la fila Latencia(ms). al igual que con Latencia mínima, se utiliza `max = Math.max(...latArr)` para tomar todos los datos de la columna y devuelve el valor más grande de las latencias.

7.1.1.3 Latencia promedio(Avg). Corresponde al valor promedio de las latencias obtenidas en la fila Latencia(ms), para esto utilizamos la linea de codigo `sum = latArr.reduce((a,b)=>a+b,0)` para sumar todos los datos de latencias, luego se usa esta línea de código `avg = sum/latArr.length` que corresponde al dato sum(suma de todas las latencias) dividido en la cantidad de datos en la columna.

estas líneas de código se asemejan a la operación:

$$Avg = \frac{\sum_{i=1}^n}{n} ; n \text{ cantidad de ventanas}$$

7.1.1.4 Conteo (Count). Conte es la cantidad de ventanas(n) registradas en la tabla.

7.1.1.5 Jitter. El jitter es la variación en el retardo de la llegada de paquetes de datos en una red.(Network Jitter - Common Causes and Best Solutions | IR, n.d.) en el código se halla con la línea:

```
const varz = latArr.reduce((a, b) => a + Math.pow(b - avg, 2), 0) / latArr.length;
const jitter = Math.sqrt(varz);
```

La variable a es el acumulador, que empieza en cero en la función reduce(), b es el valor actual de la latencia en cada iteración, Math.pow(b - avg, 2) es la diferencia al cuadrado entre la latencia actual y promedio.

En cálculos matemáticos corresponde a esta fórmula:

$$Jitter = \sqrt{\frac{1}{n} \sum_{i=1}^n (xi - Avg)^2}$$

Xi corresponde a cada valor registrado en la columna latencia en la tabla

n corresponde a la cantidad de ventanas

Avg es la latencia promedio.

7.1.1.6 Percentiles (P50) y (P95). El percentil 50(P50) corresponde al valor del medio de los datos ordenados de menor a mayor, el percentil P95 es el valor de la posición que corresponde al 95%, es decir, de la cantidad total de los datos(n) lo multiplicamos por el 95% y el resultado me va a decir en qué posición se ubica el P95% y a que valor corresponde por supuesto ordenado de menor a mayor.

7.1.1.7 Paquetes perdidos (Loss). Los paquetes perdidos son aquellos que por alguna razón ya sea por congestión de internet o por alto flujo de datos en cantidades enormes el sistema no logre registrarlos a tiempos, en el caso del sistema decimos que hay paquete perdido cuando en la tabla se salta alguna ventana ya que cada una de las ventanas vienen etiquetadas.

Figura 90

Identificar porcentaje de paquetes perdidos.

```
150 let loss = 0;
151 if (winArr.length) {
152     const minW = Math.min(...winArr),
153           maxW = Math.max(...winArr),
154           esper = maxW - minW + 1,
155           falt = esper - winArr.length;
156     loss = esper > 0 ? (falt/esper)*100 : 0;
157 }
```

Nota. La imagen corresponde a las líneas de código que me identifican el porcentaje pérdida de paquetes a partir de la cantidad de ventanas no registradas.

let loss=0, corresponde al valor inicial que va almacenar el porcentaje de paquetes perdidos.

if (winArr.length) {, aquí se verifica si el arreglo que contiene las ventanas tiene elementos, si tiene elementos el código se ejecuta.

const minW = Math.min(...winArr), calcula el valor más pequeño del arreglo winArr.

const maxW = Math.max(...winArr), calcula el valor más grande dentro del arreglo winArr.

$\text{const esper} = \text{maxW} - \text{minW} + 1$, calcula el número de ventanas esperadas dentro del rango es decir, para n ventanas el rango es n .

$\text{const falt} = \text{esper} - \text{winArr.length}$; se calcula cuántas ventanas faltan (con respecto a las ventanas del arreglo winArr).

$\text{loss} = \text{esper} > 0 ? (\text{falt} / \text{esper}) * 100 : 0$; se calcula la pérdida de paquetes(loss) dividiendo las ventanas que faltan entre las ventanas esperadas multiplicado por cien.

7.1.2 Resultados para el protocolo HTTP

Para evaluar el rendimiento del sistema, se va a evaluar los resultados realizando tres intentos.

7.1.2.1 Resultados para $N = 256$.

Figura 91

Resultado del intento uno con $N=256$.

Ventana t0	t0 (Epoch)	t1 (Epoch)	Latencia (ms)
ventana_1	1752803260000	1752803263105	3105
ventana_2	1752803263000	1752803265968	2968
ventana_3	1752803266000	1752803269417	3417
ventana_4	1752803270000	1752803273557	3557
ventana_5	1752803273000	1752803276670	3670
ventana_6	1752803277000	1752803280870	3870
ventana_7	1752803281000	1752803284935	3935
ventana_8	1752803285000	1752803287774	2774
ventana_9	1752803292000	1752803295282	3282
ventana_10	1752803296000	1752803299497	3497
ventana_11	1752803299000	1752803302278	3278

Nota. Esta imagen corresponde a los resultados del primer intento para las muestras N igual a 256.

Figura 92

Resultado registrado en tabla del intento uno con N=256.

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
2774.00	3935.00	3395.73	11	344.90	3417.00	3935.00	0.00

Nota. Esta imagen corresponde a los resultados resumido para evaluar el sistema del primer intento para las muestras N igual a 256.

Figura 93

Resultado del intento dos con N=256.

Ventana t0	t0 (Epoch)	t1 (Epoch)	Latencia (ms)
ventana_1	1752803390000	1752803393749	3749
ventana_2	1752803394000	1752803396423	2423
ventana_3	1752803398000	1752803400755	2755
ventana_4	1752803401000	1752803403888	2888
ventana_5	1752803404000	1752803406856	2856
ventana_6	1752803407000	1752803410367	3367
ventana_7	1752803410000	1752803413499	3499
ventana_8	1752803415000	1752803418645	3645
ventana_9	1752803418000	1752803421120	3120
ventana_10	1752803421000	1752803423919	2919
ventana_11	1752803425000	1752803427868	2868

Nota. Esta imagen corresponde a los resultados del segundo intento para las muestras N igual a 256, presentando un mejor resultado con respecto al intento uno.

Figura 94

Resultado registrado en tabla del intento dos con N=256.

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
2423.00	3749.00	3099.00	11	395.14	2919.00	3749.00	0.00

Nota. Esta imagen corresponde a los resultados resumido para evaluar el sistema del segundo intento para las muestras N igual a 256.

Figura 95

Resultado del intento tres con N=256.

Ventana t0	t0 (Epoch)	t1 (Epoch)	Latencia (ms)
ventana_1	1752803643000	1752803646133	3133
ventana_2	1752803671000	1752803673906	2906
ventana_3	1752803678000	1752803680498	2498
ventana_4	1752803681000	1752803685632	4632
ventana_5	1752803697000	1752803700547	3547
ventana_6	1752803710000	1752803714175	4175
ventana_7	1752803713000	1752803717117	4117
ventana_8	1752803717000	1752803720257	3257
ventana_9	1752803730000	1752803733852	3852
ventana_10	1752803733000	1752803737242	4242
ventana_11	1752803738000	1752803741298	3298

Nota. Esta imagen corresponde a los resultados del tercer intento para las muestras N igual a 256, presentando una latencia máxima mayor a los anteriores dos resultados.

Figura 96

Resultado registrado en tabla del intento tres con N=256.

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
2498.00	4632.00	3605.18	11	622.61	3547.00	4632.00	0.00

Nota. Esta imagen corresponde a los resultados resumido para evaluar el sistema del tercer intento para las muestras N igual a 256.

Tabla 15

Métricas de latencia y jitter para los tres intentos N igual a 256.

Intento	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
1	2774	3935	3395.79	3417	3935	344.90	0.00%
2	2423	3749	3099	2919	3749	395.14	0.00%
3	2498	4632	3605.18	3547	4632	622.61	0.00%
Media	2565	4105.33	3366.65	3294.33	4105.33	454.216	0.00%

Nota. La Tabla 15 resume las métricas clave de cada ejecución (mínimo, máximo, promedio, percentiles P50 y P95, y jitter) y su promedio global.

De los resultados de la Tabla 15 podemos decir que el intento uno y dos tuvieron una conexión más estable con respecto al tercer intento, esto puede indicarme que el tercer intento tuvo más probabilidad de que se perdiera paquetes de los tres resultados.

7.1.2.2 Resultados para N = 512.

Figura 97

Resultado del intento uno con N=512.

Ventana t0	t0 (Epoch)	t1 (Epoch)	Latencia (ms)
ventana_1	1752806255000	1752806258575	3575
ventana_2	1752806258000	1752806261361	3361
ventana_3	1752806262000	1752806264471	2471
ventana_4	1752806264000	1752806267608	3608
ventana_5	1752806267000	1752806270529	3529
ventana_6	1752806270000	1752806273071	3071
ventana_7	1752806273000	1752806276901	3901
ventana_8	1752806278000	1752806281162	3162
ventana_9	1752806285000	1752806288536	3536
ventana_10	1752806288000	1752806291468	3468
ventana_11	1752806292000	1752806295266	3266

Nota. Esta imagen corresponde a los resultados del primer intento para las muestras N igual a 512.

Figura 98

Resultado registrado en tabla del intento uno con N=512.

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
2471.00	3901.00	3358.91	11	355.92	3468.00	3901.00	0.00

Nota. Esta imagen corresponde a los resultados resumido para evaluar el sistema del primer intento para las muestras N igual a 512.

Figura 99

Resultado del intento dos con N=512.

Ventana t0	t0 (Epoch)	t1 (Epoch)	Latencia (ms)
ventana_1	1752806381000	1752806384073	3073
ventana_2	1752806385000	1752806388300	3300
ventana_3	1752806388000	1752806392382	4382
ventana_4	1752806393000	1752806396064	3064
ventana_5	1752806396000	1752806398719	2719
ventana_6	1752806400000	1752806403185	3185
ventana_7	1752806404000	1752806407477	3477
ventana_8	1752806408000	1752806410570	2570
ventana_9	1752806410000	1752806413486	3486
ventana_10	1752806414000	1752806416743	2743
ventana_11	1752806416000	1752806419367	3367

Nota. Esta imagen corresponde a los resultados del segundo intento para las muestras N igual a 512, presentando un resultado más alto (mayor latencia) con respecto al intento uno.

Figura 100

Resultado registrado en tabla del intento dos con N=512.

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
2570.00	4382.00	3215.09	11	473.19	3185.00	4382.00	0.00

Nota. Esta imagen corresponde a los resultados resumido para evaluar el sistema del segundo intento para las muestras N igual a 512.

Figura 101

Resultado del intento tres con N=512.

Ventana t0	t0 (Epoch)	t1 (Epoch)	Latencia (ms)
ventana_1	1752806509000	1752806512980	3980
ventana_2	1752806514000	1752806517155	3155
ventana_3	1752806517000	1752806522222	5222
ventana_4	1752806522000	1752806525067	3067
ventana_5	1752806525000	1752806528285	3285
ventana_6	1752806528000	1752806530792	2792
ventana_7	1752806530000	1752806534058	4058
ventana_8	1752806535000	1752806537982	2982
ventana_9	1752806538000	1752806540174	2174
ventana_10	1752806541000	1752806544237	3237
ventana_11	1752806544000	1752806547083	3083

Nota. Esta imagen corresponde a los resultados del tercer intento para las muestras N igual a 512, este presenta unos valores de latencia mayores a los dos intentos anteriores.

Figura 102

Resultado registrado en tabla del intento tres con N=512.

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
2174.00	5222.00	3366.82	11	764.89	3155.00	5222.00	0.00

Nota. Esta imagen corresponde a los resultados resumido para evaluar el sistema del tercer intento para las muestras N igual a 512.

Tabla 16

Métricas de latencia y jitter para los tres intentos N=512.

Corrida	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
1	2471	3901	3358.91	3468	3901	355.92	0.00%
2	2570	4382	3215.09	3185	4382	473.19	0.00%
3	2174	5222	3366.82	3155	5222	764.89	0.00%
Media	2405	4501.66	3313.606	3269.33	4501.66	531.33	0.00%

Nota. La Tabla 16 resume las métricas clave de cada ejecución (mínimo, máximo, promedio, percentiles P50 y P95, y jitter) y su promedio global.

De los resultados obtenidos podemos ver que el tercer intento demostró ser el de peor conexión estable de los tres. si vemos esta tabla con respecto a la Tabla 15 podemos ver que los resultados de la tabla 8 tuvieron resultados muy parecidos excepto en el jitter, esto demuestra que el jitter en N igual a 512 le cuesta más tener una conexión estable en comparación con los resultados del jitter en N igual a 256.

7.1.2.3 Resultados para N = 1024.

Figura 103

Resultado del intento uno con N=1024.

Ventana t0	t0 (Epoch)	t1 (Epoch)	Latencia (ms)
ventana_1	1752853399000	1752853401517	2517
ventana_2	1752853402000	1752853404552	2552
ventana_3	1752853404000	1752853406782	2782
ventana_4	1752853406000	1752853409412	3412
ventana_5	1752853409000	1752853412806	3806
ventana_6	1752853412000	1752853415040	3040
ventana_7	1752853415000	1752853419121	4121
ventana_8	1752853420000	1752853422600	2600
ventana_9	1752853423000	1752853425171	2171
ventana_10	1752853425000	1752853428360	3360
ventana_11	1752853428000	1752853431369	3369

Nota. Esta imagen corresponde a los resultados del primer intento para las muestras N igual a 1024.

Figura 104

Resultados registrados en la tabla del intento uno con N=1024.

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
2171.00	4121.00	3066.36	11	573.96	3040.00	4121.00	0.00

Nota. Esta imagen corresponde a los resultados resumido para evaluar el sistema del primer intento para las muestras N igual a 1024.

Figura 105

Resultado del intento dos con N=1024.

Ventana t0	t0 (Epoch)	t1 (Epoch)	Latencia (ms)
ventana_1	1752852869000	1752852873221	4221
ventana_2	1752852873000	1752852877360	4360
ventana_3	1752852877000	1752852880560	3560
ventana_4	1752852880000	1752852884457	4457
ventana_5	1752852883000	1752852887410	4410
ventana_6	1752852887000	1752852890130	3130
ventana_7	1752852899000	1752852902098	3098
ventana_8	1752852904000	1752852909392	5392
ventana_9	1752852908000	1752852912247	4247
ventana_10	1752852912000	1752852915608	3608
ventana_11	1752852915000	1752852918818	3818

Nota. Esta imagen corresponde a los resultados del segundo intento para las muestras N igual a 1024.

Figura 106

Resultados registrados en la tabla del intento dos con N=1024.

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
3098.00	5392.00	4027.36	11	639.37	4221.00	5392.00	0.00

Nota. Esta imagen corresponde a los resultados resumido para evaluar el sistema del segundo intento para las muestras N igual a 1024.

Figura 107

Resultado del intento tres con N=1024.

Ventana t0	t0 (Epoch)	t1 (Epoch)	Latencia (ms)
ventana_1	1752853119000	1752853121570	2570
ventana_2	1752853121000	1752853123964	2964
ventana_3	1752853124000	1752853126576	2576
ventana_4	1752853126000	1752853128968	2968
ventana_5	1752853129000	1752853131292	2292
ventana_6	1752853131000	1752853133739	2739
ventana_7	1752853134000	1752853136684	2684
ventana_8	1752853136000	1752853139811	3811
ventana_9	1752853140000	1752853142451	2451
ventana_10	1752853143000	1752853145049	2049
ventana_11	1752853146000	1752853149470	3470

Nota. Esta imagen corresponde a los resultados del tercer intento para las muestras N igual a 1024.

Figura 108

Resultados registrados en la tabla del intento tres con N=1024.

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
2049.00	3811.00	2779.45	11	484.83	2684.00	3811.00	0.00

Nota. Esta imagen corresponde a los resultados resumido para evaluar el sistema del tercer intento para las muestras N igual a 1024.

Tabla 17

Métricas de latencia y jitter para tres intentos con N=1024.

Corrida	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
1	2171	4121	3066.36	3040	4121	573.96	0.00%
2	3098	5392	4027.36	4221	5392	639.37	0.00%
3	2049	3811	2779.45	2684	3811	484.83	0.00%
Media	2439.33	4441.33	3291.056	3315	4441.33	566.053	0.00%

Nota. La Tabla 17 resume las métricas clave de cada ejecución (mínimo, máximo, promedio, percentiles P50 y P95, y jitter) y su promedio global.

Los resultados me indican que en el segundo intento estuvieron latencias más altas que los otros dos intentos, comparando con la Tablas 15 y Tabla 16, podemos ver que el Jitter en N igual a 1024 es más alto, por lo tanto, en N igual a 1024 le cuesta más tener una conexión estable en comparación a N igual 256 y 512.

7.1.2.4 Resultados para N = 2048.

Figura 109

Resultado del intento uno con N=2048.

Ventana t0	t0 (Epoch)	t1 (Epoch)	Latencia (ms)
ventana_1	1752875048000	1752875051687	3687
ventana_2	1752875051000	1752875054573	3573
ventana_3	1752875055000	1752875058630	3630
ventana_4	1752875058000	1752875060828	2828
ventana_5	1752875061000	1752875063941	2941
ventana_6	1752875066000	1752875068536	2536
ventana_7	1752875069000	1752875071397	2397
ventana_8	1752875072000	1752875074493	2493
ventana_9	1752875075000	1752875076960	1960
ventana_10	1752875079000	1752875082165	3165
ventana_11	1752875083000	1752875085929	2929

Nota. Esta imagen corresponde a los resultados del primer intento para las muestras N igual a 2048.

Figura 110

Resultados registrados en la tabla del intento uno con N=2048.

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
1960.00	3687.00	2921.73	11	531.95	2929.00	3687.00	0.00

Nota. Esta imagen corresponde a los resultados resumido para evaluar el sistema del primer intento para las muestras N igual a 2048.

Figura 111

Resultado del intento dos con N=2048.

Ventana t0	t0 (Epoch)	t1 (Epoch)	Latencia (ms)
ventana_1	1752875151000	1752875153756	2756
ventana_2	1752875154000	1752875156232	2232
ventana_3	1752875156000	1752875158901	2901
ventana_4	1752875159000	1752875161402	2402
ventana_5	1752875162000	1752875164557	2557
ventana_6	1752875165000	1752875166953	1953
ventana_7	1752875168000	1752875170700	2700
ventana_8	1752875171000	1752875173934	2934
ventana_9	1752875174000	1752875176738	2738
ventana_10	1752875178000	1752875180396	2396
ventana_11	1752875181000	1752875183201	2201

Nota. Esta imagen corresponde a los resultados del segundo intento para las muestras N igual a 2048.

Figura 112

Resultados registrados en la tabla del intento dos con N=2048.

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
1953.00	2934.00	2524.55	11	299.82	2557.00	2934.00	0.00

Nota. Esta imagen corresponde a los resultados resumido para evaluar el sistema del segundo intento para las muestras N igual a 2048.

Figura 113

Resultado del intento tres con N=2048.

Ventana t0	t0 (Epoch)	t1 (Epoch)	Latencia (ms)
ventana_1	1752875236000	1752875238922	2922
ventana_2	1752875239000	1752875242216	3216
ventana_3	1752875242000	1752875244955	2955
ventana_4	1752875246000	1752875248509	2509
ventana_5	1752875249000	1752875251704	2704
ventana_6	1752875252000	1752875254408	2408
ventana_7	1752875255000	1752875256834	1834
ventana_8	1752875257000	1752875260137	3137
ventana_9	1752875261000	1752875264965	3965
ventana_10	1752875266000	1752875268309	2309
ventana_11	1752875268000	1752875271258	3258

Nota. Esta imagen corresponde a los resultados del tercer intento para las muestras N igual a 2048.

Figura 114

Resultados registrados en la tabla del intento tres con N=2048.

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
1834.00	3965.00	2837.91	11	547.60	2922.00	3965.00	0.00

Nota. Esta imagen corresponde a los resultados resumido para evaluar el sistema del segundo intento para las muestras N igual a 2048.

Tabla 18

Métricas de latencia y jitter para tres intentos con N=2048.

Corrida	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
1	1960	3687	2921.73	2929	3687	531.95	0.00%
2	1953	2934	2524.55	2557	2934	299.82	0.00%
3	1834	3965	2837.91	2922	3965	547.60	0.00%
Media	1915.66	3528.66	2761.396	2802.66	3525.33	459.79	0.00%

Nota. La Tabla 18 resume las métricas clave de cada ejecución (mínimo, máximo, promedio, percentiles P50 y P95, y jitter) y su promedio global.

Para los resultados N igual a 2048 tuvieron mejores resultados de latencia que los resultados N igual a 1024, al igual que el Jitter, que es el más bajo de todas las muestras anteriores (intento 2).

7.1.2.5 Resultados para N = 4096.

Figura 115

Resultado del intento uno con N=4096.

Ventana t0	t0 (Epoch)	t1 (Epoch)	Latencia (ms)
ventana_1	1752876451000	1752876453834	2834
ventana_2	1752876455000	1752876457921	2921
ventana_3	1752876458000	1752876461278	3278
ventana_4	1752876461000	1752876463828	2828
ventana_5	1752876464000	1752876466749	2749
ventana_6	1752876467000	1752876469995	2995
ventana_7	1752876471000	1752876473291	2291
ventana_8	1752876475000	1752876478345	3345
ventana_9	1752876478000	1752876480929	2929
ventana_10	1752876481000	1752876484094	3094
ventana_11	1752876484000	1752876487230	3230

Nota. Esta imagen corresponde a los resultados del primer intento para las muestras N igual a 4096.

Figura 116

Resultados registrados en la tabla del intento uno con N=4096.

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
2291.00	3345.00	2954.00	11	281.05	2929.00	3345.00	0.00

Nota. Esta imagen corresponde a los resultados resumido para evaluar el sistema del primer intento para las muestras N igual a 4096.

Figura 117

Resultado del intento dos con N=4096.

Ventana t0	t0 (Epoch)	t1 (Epoch)	Latencia (ms)
ventana_1	1752876627000	1752876629702	2702
ventana_2	1752876630000	1752876632593	2593
ventana_3	1752876633000	1752876635752	2752
ventana_4	1752876636000	1752876638415	2415
ventana_5	1752876639000	1752876641868	2868
ventana_6	1752876642000	1752876644512	2512
ventana_7	1752876645000	1752876647441	2441
ventana_8	1752876648000	1752876650773	2773
ventana_9	1752876651000	1752876653918	2918
ventana_10	1752876656000	1752876658319	2319
ventana_11	1752876658000	1752876661007	3007

Nota. Esta imagen corresponde a los resultados del segundo intento para las muestras N igual a 4096.

Figura 118

Resultados registrados en la tabla del intento dos con N=4096

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
2319.00	3007.00	2663.64	11	214.00	2702.00	3007.00	0.00

Nota. Esta imagen corresponde a los resultados resumido para evaluar el sistema del segundo intento para las muestras N igual a 4096.

Figura 119

Resultado del intento tres con N=4096.

Ventana t0	t0 (Epoch)	t1 (Epoch)	Latencia (ms)
ventana_1	1752876849000	1752876851902	2902
ventana_2	1752876853000	1752876855793	2793
ventana_3	1752876856000	1752876859084	3084
ventana_4	1752876859000	1752876861824	2824
ventana_5	1752876862000	1752876864700	2700
ventana_6	1752876865000	1752876867993	2993
ventana_7	1752876868000	1752876872096	4096
ventana_8	1752876872000	1752876875154	3154
ventana_9	1752876875000	1752876878624	3624
ventana_10	1752876878000	1752876881137	3137
ventana_11	1752876881000	1752876884279	3279

Nota. Esta imagen corresponde a los resultados del tercer intento para las muestras N igual a 4096.

Figura 120

Resultados registrados en la tabla del intento tres con N=4096

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
2700.00	4096.00	3144.18	11	388.80	3084.00	4096.00	0.00

Nota. Esta imagen corresponde a los resultados resumido para evaluar el sistema del tercer intento para las muestras N igual a 4096.

Tabla 19

Métricas de latencia y jitter para tres intentos con N=4096.

Corrida	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
1	2291	3345	2954	2929	3345	281.05	0.00%
2	2319	3007	2663.64	2702	3007	214	0.00%
3	2700	4096	3144.18	3084	4096	388.80	0.00%
Media	2436.66	3482.66	2920.606	2905	3482.66	294.616	0.00%

Nota. La Tabla 19 resume las métricas clave de cada ejecución (mínimo, máximo, promedio, percentiles P50 y P95, y jitter) y su promedio global.

Con los resultados N igual a 4096 se ve que la latencia no necesariamente se ve demasiado afectada por las variaciones de las N muestras como se pensaba en un principio, por el contrario, vemos que con N igual a 4096 dio como resultado el Jitter más bajo con respecto a las demás N, lo que se ve con los resultados es que todo se ve afectado por la congestión en la red.

Tabla 20

Comparación global de latencias para HTTP con N = 256, 512, 1024, 2048, 4096, 8192.

Tamaño de N	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	loss (%)
256	2565	4105.33	3366.65	3294.33	4105.33	454.216	0.00%
512	2405	4501.66	3313.606	3269.33	4501.66	531.33	0.00%
1024	2439.33	4441.33	3291.056	3315	4441.33	566.053	0.00%
2048	1915.66	3528.66	2761.396	2802.66	3525.33	459.79	0.00%
4096	2436.66	3482.66	2920.606	2905	3482.66	294.616	0.00%

8192	No datos	No datos	No datos	No datos	No datos	No datos	No datos
Media General	2352.33	4011.928	3130.662	3117.264	4011.262	461.201	0.00%
			8				

Nota. La tabla muestra los valores promedio de latencia para el protocolo HTTP con diferentes tamaños de ventana (N). También incluye la media general calculada de todas las mediciones de N, excluyendo el caso de N = 8192, ya que no hay datos disponibles.

Cómo podemos ver los resultados fueron buenos para ser un protocolo no dedicado al envío de datos en tiempo real, el Jitter que tan estable es la conexión da como resultado valores bajos por lo tanto es una conexión en general estable. se confirma que las N muestras mayores no resultan en latencias y Jitters altos, por el contrario, en la N más alta de los resultados tuvo el Jitter más bajo, lo que me indica que si da como resultado latencias altas es por otros motivos como la congestión en la red y fallos en conectividad.

Para hacer demostración de lo anterior mencionado se hizo una prueba con más ventanas con el fin de comprobar de que el Jitter no aumenta demasiado con un N igual a 4096 qué es el máximo de muestras que permite el sistema, el resultado fue el siguiente.

Figura 121

Resultado del intento con el máximo de ventanas N=4096.

Min	Max	Avg	Count	Jitter	P50	P95	Loss (%)
1865.00	7003.00	2857.18	201	646.88	2751.00	3838.00	0.00

Nota. Esta imagen corresponde a los resultados con un numero de ventanas igual a 201 para las muestras N igual a 4096.

como se ve en la Figura 121, el jitter no se elevó demasiado dando un resultado similar al Jitter de N igual 1024, con esto se comprueba de que la latencia y el Jitter aumenta es por otros factores como la red y no por el tamaño de las muestras.

7.2 Conclusión comparativa

El protocolo HTTP, que opera en la capa de aplicación del modelo OSI, ha sido evaluado en términos de latencia y estabilidad mediante el uso de Google Apps Script como intermediario para registrar los datos en una hoja de cálculo de Google Sheets. Durante las pruebas, se varió el tamaño de la ventana de muestras (N) y se realizaron varios intentos para analizar su impacto sobre las métricas clave, como la latencia, el jitter, y la pérdida de paquetes.

7.2.1 Latencia y Estabilidad de la Conexión.

Los resultados demuestran que, a pesar de que HTTP no está diseñado específicamente para la transmisión de datos en tiempo real, los valores de latencia obtenidos son razonablemente buenos, especialmente cuando se compara con otros protocolos más dedicados. En las pruebas, se evidenció que el protocolo HTTP, aun con la variación en el tamaño de la ventana de muestras (N), mantuvo una latencia relativamente estable. Los intentos con tamaños N más grandes (2048 y 4096) no resultaron en latencias significativamente más altas, como podría haberse esperado inicialmente, sino que incluso mostraron un jitter más bajo en comparación con tamaños de N menores.

7.2.2 Jitter y Variación en los Retardos

El jitter, que mide la variabilidad en la latencia entre los paquetes, se mantuvo sorprendentemente bajo incluso en los intentos con tamaños de muestra mayores. En particular, cuando el tamaño de la ventana aumentó a 4096 muestras, el jitter fue el más bajo registrado en comparación con N = 256, 512 y 1024. Este comportamiento sugiere que, si bien el protocolo

HTTP puede generar latencias elevadas bajo ciertas condiciones, la congestión de la red y otros factores externos tienen un impacto mucho más significativo en la estabilidad de la conexión que el tamaño de la muestra en sí mismo.

7.2.3 Pérdida de Paquetes

Un aspecto importante de la evaluación fue la baja tasa de pérdida de paquetes, que se mantuvo en un 0% en todas las pruebas realizadas. Esto indica que, aunque HTTP no es la opción más eficiente para el envío de datos en tiempo real, su desempeño fue adecuado para el propósito del experimento, con una entrega confiable de la mayoría de los paquetes.

7.2.4 Análisis Comparativo de las Muestras

Al comparar los resultados entre las diferentes muestras de tamaño N, se observa que los intentos con tamaños N más grandes presentaron una mayor estabilidad en cuanto a jitter y latencia, a pesar de las expectativas de que el aumento de N podría causar una mayor congestión y retraso. Los valores promedio de latencia para cada tamaño de muestra fueron consistentes y no reflejaron una degradación significativa del rendimiento con tamaños más grandes. Este resultado subraya la robustez del protocolo HTTP en condiciones controladas.

7.2.5 Factores Externos y Congestión de Red

Los resultados sugieren que factores externos, como la congestión de la red y fallos en la conectividad, tienen un impacto mucho mayor en el rendimiento que el tamaño de la ventana de muestras utilizada. Las variaciones de latencia y jitter observadas entre los intentos parecen estar más relacionadas con fluctuaciones de la red que con el propio protocolo HTTP o el tamaño de los paquetes.

7.2.6 Conclusión Final

En general, el protocolo HTTP, utilizado con Google Apps Script como intermediario, demostró ser una opción funcional para la medición y el registro de datos con una latencia aceptable y un jitter bajo, incluso con tamaños de muestra más grandes. La estabilidad de la conexión es relativamente buena, con la congestión de la red como el principal factor que afecta la latencia. Si bien HTTP no es el protocolo más adecuado para aplicaciones en tiempo real, en este contexto específico mostró un desempeño sorprendentemente confiable. La evidencia obtenida sugiere que para proyectos con requerimientos de latencia y estabilidad en condiciones controladas, HTTP puede ser una solución válida, aunque siempre será recomendable evaluar alternativas como UDP o TCP para aplicaciones más sensibles a la latencia.