



# PROYECTO CC8

**José Fernando Chalo Sajquín – 22002407**

**Alejandra Nazareth Sosa Carrillo – 22002246**

**María Regina Rivas Girón - 21005971**

## Tabla de contenido

1.	Descripción General del Proyecto .....	1
2.	Objetivos.....	2
2.1	Objetivo General.....	2
2.2	Objetivos Específicos .....	2
3.	Arquitectura General del proyecto .....	3
3.1	Board principal: NuttX en BeagleBone Black .....	3
3.2	Board funcional: NuttX en ESP32-S3 .....	4
4.	Configuraciones de Nuttx.....	7
5.1	Configuración para BeagleBone Black (AM335x) .....	7
5.1.1	Configuración base e importación del código .....	7
5.1.2	Configuración del sistema y drivers .....	8
5.1.3	Verificación y ausencia de RAW sockets .....	8
5.1.4	Habilitación de UART y corrección de driver .....	9
5.1.5	Ajuste en configuración de relojes AM335x .....	9
5.1.6	Integración de aplicaciones personalizadas .....	9
5.2	Configuración para ESP32-S3.....	11
5.2.1	Instalación del entorno de compilación .....	11
5.	Seguimiento de investigación.....	14
5.1	Verificación inicial de capacidades de red en Nuttx.....	14
5.2	Cambio hacia comunicación por UART.....	16
5.3	Pruebas y estabilización de la comunicación .....	16
6.	Resultados.....	18
6.1	Resultados en BeagleBone Black .....	18
6.2	Resultados en ESP32-S3 .....	19
7.	Conclusiones.....	20
8.	Referencias .....	21

## **1. Descripción General del Proyecto**

Este proyecto consiste en el desarrollo de una aplicación de comunicación sobre NuttX, explorando desde las capas inferiores del modelo TCP/IP hasta la interacción directa con el hardware, la plataforma principal de estudio fue la BeagleBone Black, seleccionada para analizar el comportamiento del sistema operativo en tiempo real en un entorno donde gran parte del soporte debe ser configurado, verificado y depurado manualmente.

El objetivo central fue construir un mecanismo de comunicación configurable en tiempo de ejecución capaz de solicitar parámetros como dirección IP y puerto, y de establecer un canal entre dispositivos utilizando la interfaz de sockets de NuttX, esto permitió profundizar en la creación, envío y recepción de datos a nivel de aplicación, así como en el manejo de estructuras internas del stack de red.

Durante el desarrollo se comprobó que en versiones recientes de NuttX, la BeagleBone Black no cuenta con soporte funcional para el subsistema Ethernet, lo que imposibilita el uso de sockets convencionales o Raw Sockets sobre la interfaz física, esta limitación condujo a una investigación del stack de NuttX, su configuración interna, las opciones disponibles en menuconfig y los requerimientos reales del sistema para activar loopback, drivers y dispositivos de red.

Como resultado de esta exploración se documentaron fallos, configuraciones incompletas, ajustes en controladores, pruebas en loopback y finalmente la transición a un canal serial como vía de comunicación alternativa, utilizando UART0 y herramientas externas para establecer un cliente y un servidor operando sobre NuttX, para demostrar la parte funcional del proyecto se empleó una ESP32-S3, un board con soporte completo para NuttX y capacidad Wi-Fi.

## **2. Objetivos**

### **2.1 Objetivo General**

Desarrollar un sistema de comunicación basado en NuttX que permita el intercambio de datos entre dispositivos mediante parámetros dinámicos, investigando el comportamiento del sistema operativo sobre la BeagleBone Black y demostrando la funcionalidad completa en la ESP32-S3.

### **2.2 Objetivos Específicos**

- Analizar el soporte real de NuttX en la BeagleBone Black, evaluando drivers disponibles, configuración del stack de red y limitaciones presentes en la implementación del subsistema Ethernet.
- Investigar la viabilidad de utilizar Raw Sockets en NuttX, comprendiendo su funcionamiento interno, los requisitos del sistema y las causas de los errores obtenidos durante las pruebas.
- Establecer un canal de comunicación alternativo en la BeagleBone Black mediante interfaces disponibles (como UART).
- Desarrollar la parte funcional del sistema en un board con soporte completo de NuttX, utilizando la ESP32-S3 para establecer comunicación por Wi-Fi.

### **3. Arquitectura General del proyecto**

#### **3.1 Board principal: NuttX en BeagleBone Black**

La BeagleBone Black se estableció como la plataforma principal del proyecto debido a que sobre ella se debía integrar NuttX y validar el funcionamiento de las capas inferiores del modelo TCP/IP, sin embargo, durante la implementación se identificó que el port oficial de NuttX para AM335x aún no incluye soporte completo para el controlador Ethernet CPSW, lo cual impide la creación de la interfaz eth0 y, en consecuencia, el uso de sockets TCP/UDP tradicionales.

Ante esta limitación, la arquitectura del proyecto evolucionó hacia un enfoque híbrido:

- Capa de red simulada mediante Loopback**

Se habilitaron las opciones de Networking, Loopback Device e IP Loopback, permitiendo probar comunicación basada en sockets dentro de la misma BeagleBone Black sin requerir hardware Ethernet funcional.

- Análisis de RAW sockets y verificación del .config**

Se inspeccionó el soporte real del build de NuttX 12.11.0, confirmando que la opción de RAW Sockets AF\_INET no está disponible para esta plataforma, descartando la posibilidad de implementar el laboratorio mediante paquetes Ethernet reales.

- Transición a comunicación por UART**

Al no contar con un stack TCP/IP operativo, se implementó una segunda capa de comunicación utilizando:

- UART0 para el acceso a NSH mediante FTDI
- Un canal serial adicional para la comunicación host de ida y vuelta hacia la BeagleBone Black

Se desarrollaron nuevamente los programas cliente y servidor adaptados al entorno UART y se resolvieron errores de driver, incluyendo la corrección del handler de interrupción en am335x\_serial.c.

- **Integración mediante aplicaciones en NuttX**

Los programas se incorporaron como aplicaciones internas con su propio Kconfig, Makefile y registro en el menú de NuttX, lo que permitió ejecutarlos directamente desde NSH.

### **3.2 Board funcional: NuttX en ESP32-S3**

La ESP32-S3 se utilizó como plataforma funcional para validar las capacidades reales del stack de red en NuttX, particularmente en lo referente a comunicación Wi-Fi y uso de sockets estándar, a diferencia de la BeagleBone Black, esta placa sí cuenta con un sistema completamente operativo del subsistema de red, lo cual permitió ejecutar aplicaciones que en la Beagle eran inviables.

#### **Configuración base del sistema**

Para esta placa se empleó la configuración oficial esp32s3-devkit:wifi, la cual integra:

- Controlador Wi-Fi habilitado y operativo.
- Stack TCP/IP completo.
- DHCP, DNS y herramientas de red básicas.

Adicionalmente se ajustaron parámetros de memoria y tipo de chip para asegurar compatibilidad con el modelo específico utilizado (ESP32-S3 N8R8), y también se habilitó explícitamente la opción de RAW sockets, ya que no viene activa en la configuración por defecto.

## Capacidades verificadas en ESP32-S3

- **Sockets TCP/UDP completamente funcionales**

Se validó la comunicación entre cliente y servidor en modo loopback, lo cual funcionó correctamente y de forma estable.

```
nsh> ifconfig
lo      Link encap:Local Loopback at RUNNING mtu 1518
          inet addr:127.0.0.1 DRaddr:127.0.0.1 Mask:255.0.0.0

wlan0   Link encap:Ethernet Hwaddr f0:9e:9e:21:fc:88 at RUNNING mtu 1500
          inet addr:192.168.1.216 DRaddr:192.168.1.1 Mask:255.255.255.0

nsh> labl_server &
labl_server [6:10]
nsh> === Servidor UDP NuttX ===
Escuchando en puerto 5004
Esperando mensaje del cliente.....
nsh> labl_client
== Cliente UDP NuttX ==
Servidor: 127.0.0.1:5004
Ingrese operación (ej: 3 + 2) o EXIT: 3 + 2
Envíando mensaje al servidor.....
Recibiendo mensaje del cliente.....
Mensaje recibido: 3 + 2
Respuesta enviada: 5
Esperando mensaje del cliente.....
Respuesta del servidor: 5
Ingrese operación (ej: 3 + 2) o EXIT: 
```

**Figura 1:** Verificación de loopback activo.

- **Conectividad Wi-Fi real**

La placa estableció conexión mediante:

- DHCP (obtención de dirección IP).
- Ping a dominios externos (validando resolución DNS y conectividad real).
- Escaneo de redes usando la aplicación wapi.

```
nsh> ping 8.8.8.8
PING 8.8.8.8 56 bytes of data
56 bytes from 8.8.8.8: icmp_seq=0 time=60.0 ms
56 bytes from 8.8.8.8: icmp_seq=1 time=50.0 ms
56 bytes from 8.8.8.8: icmp_seq=2 time=40.0 ms
56 bytes from 8.8.8.8: icmp_seq=3 time=40.0 ms
56 bytes from 8.8.8.8: icmp_seq=4 time=40.0 ms
56 bytes from 8.8.8.8: icmp_seq=5 time=50.0 ms
56 bytes from 8.8.8.8: icmp_seq=6 time=40.0 ms
56 bytes from 8.8.8.8: icmp_seq=7 time=40.0 ms
56 bytes from 8.8.8.8: icmp_seq=8 time=40.0 ms
56 bytes from 8.8.8.8: icmp_seq=9 time=50.0 ms
10 packets transmitted, 10 received, 0% packet loss, time 10100 ms
rtt min/avg/max/mdev = 40.000/45.000/60.000/6.708 ms
```

**Figura 2:** Verificación de ping a otros dominios.

bssid / frequency	/ signal level	/ encode	/ ssid
90:f8:91:dc:60:50	2462	-91	0800 CLARO1_DC604C
48:4e:fc:9f:d1:7d	2462	-90	0800 TIIGO-B5C5
e0:88:5d:8f:6b:bc	2462	-90	0800 UNE_46B1
84:01:12:e6:14:ea	2462	-89	0800 Casa_taracena
50:0f:f5:bd:21:c9	2437	-89	0800 REPETIDOR
c0:25:67:e5:70:49	2437	-86	0800 Myornet
f8:63:d9:ac:ea:49	2412	-86	0800 SagOro
f8:63:d9:9c:fc:0e	2412	-85	0800 TIIGO-9609
74:3a:ef:b5:78:db	2462	-83	0800 CLARO1_B578D7
74:3a:ef:1c:9a:c2	2462	-78	0800 CLARO1_1C9ABE
74:3a:ef:8d:ec:6f	2437	-77	0800 CLARO1_8DEC6B
18:34:af:b2:54:b5	2462	-76	0800 CLARO1_8254B2
90:f8:91:dd:52:18	2437	-75	0800 CLARO1_DD5214
90:93:5a:12:d8:26	2437	-74	0800 TIIGO-E696
74:3a:ef:8d:13:3f	2437	-64	0800 CLARO1_8D133B
18:34:af:83:7e:04	2462	-24	0800 CLARO1_837E01

**Figura 3:** Escaneo de redes con la herramienta wapi.

Estas pruebas confirmaron que la implementación Wi-Fi en NuttX sobre ESP32-S3 es completamente funcional y operativa.

- **Comunicación entre dispositivos externos**

Se realizaron pruebas cliente-servidor entre:

- Cliente ejecutándose en NuttX (ESP32-S3).
- Servidor ejecutándose en una laptop externa

La transmisión fue exitosa, confirmando que la placa puede interactuar con dispositivos externos a través de la red local.

## 4. Configuraciones de Nuttx

### 5.1 Configuración para BeagleBone Black (AM335x)

#### 5.1.1 Configuración base e importación del código

El entorno de trabajo se inicializó descargando los repositorios oficiales de NuttX:

```
mkdir nuttxspace
```

```
cd nuttxspace
```

```
git clone https://github.com/apache/nuttx.git nuttx
```

```
git clone https://github.com/apache/nuttx-apps apps
```

Se cargó la configuración base de la BeagleBone Black:

```
tools/configure.sh boards/arm/am335x/beaglebone-black/configs/nsh
```

Las configuraciones se realizaron con:

```
make menuconfig
```

Y la compilación con soporte multinúcleo:

```
make -j$(nproc)
```

### 5.1.2 Configuración del sistema y drivers

Para habilitar el stack de red y permitir pruebas locales:

- Networking support \*
- Internet Protocol (AF\_INET) \*
- Loopback device support
- IP loopback support

```
#  
# Link layer support  
#  
CONFIG_NET_ETHERNET=y  
# CONFIG_NET_CELLULAR is not set  
CONFIG_NET_LOOPBACK=y  
CONFIG_NET_IPLOOPBACK=y  
CONFIG_NET_LOOPBACK_PKTSIZE=0  
# CONFIG_NET_MBIM is not set  
CONFIG_NET_SLIP=y  
CONFIG_NET_SLIP_NINTERFACES=1  
CONFIG_NETDEV_LATEINIT=y
```

**Figura 4:** Macros esperadas en .config

- Worker thread support \*
- High-priority kernel worker thread \*
- Low-priority kernel worker thread \*

### 5.1.3 Verificación y ausencia de RAW sockets

Se buscó explícitamente soporte para RAW sockets AF\_INET, sin embargo, el CONFIG\_NET\_RAW no aparece en el build, por lo que la plataforma no soporta RAW sockets, y fue confirmado al intentar crear un socket, obteniendo el error 93.

#### 5.1.4 Habilitación de UART y corrección de driver

Al habilitar UART1, la compilación fallaba debido a un manejador de interrupciones faltante, por lo que se comentó la línea que causaba el error para permitir que el sistema compile correctamente sin afectar el uso básico del UART.

#### 5.1.5 Ajuste en configuración de relojes AM335x

Durante una compilación previa se detectó un error en la inicialización del clock, por lo que se modificó el archivo `am335x_clockconfig.c`

Sustituyendo:

`CM PER L4LS CLKSTCTRL UART GCLK`

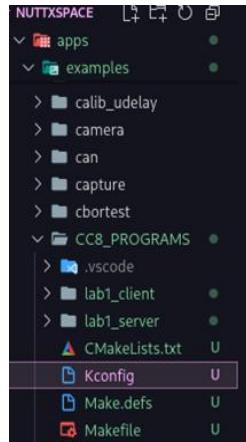
Por:

`CM PER L4LS CLKSTCTRL UART GFCLK`

Esta corrección permite la inicialización adecuada del periférico UART.

#### 5.1.6 Integración de aplicaciones personalizadas

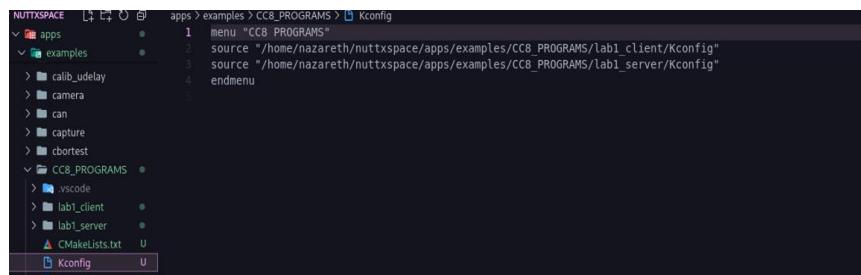
Las aplicaciones se agregaron en `apps/examples`



**Figura 5:** Aplicación agregada en apps/examples

Cada aplicación incluye:

- Un Kconfig
- Un Makefile
- Código fuente (.c)



**Figura 6:** Archivo Kconfig

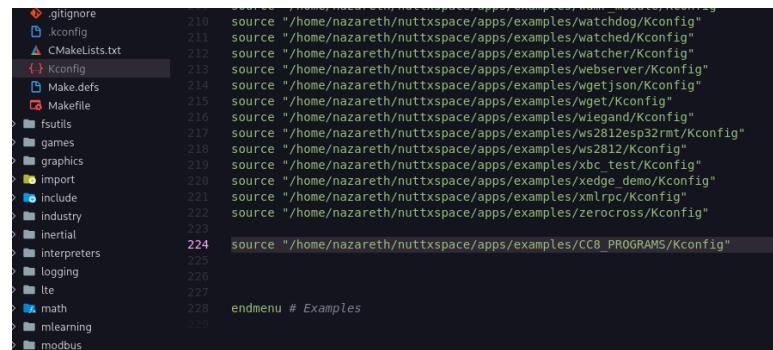
```

apps > examples > CC8_PROGRAMS > lab1_client > Makefile
1 include $(APPDIR)/Makefile.defs
2
3 # Nombre del programa
4 PROGNAME = lab1_client
5 PRIORITY = SCHED_PRIORITY_DEFAULT
6 STACKSIZE = 2048
7 MODULE = $(CONFIG_EXAMPLES_HELLO_NUTTX)
8
9 # Archivos fuente
10 MAINSRC = lab1_client.c
11
12 include $(APPDIR)/Application.mk
13

```

**Figura 7:** Archivo Makefile

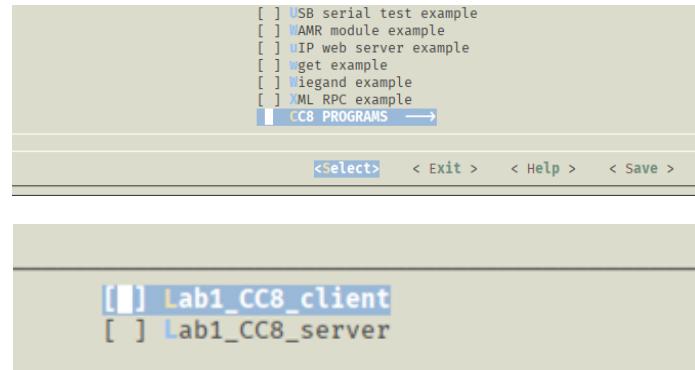
Igualmente, se agrega la ruta de la carpeta agregada en el archivo Kconfig de Examples



```
210 source "/home/nazareth/nuttxspace/apps/examples/watchdog/Kconfig"
211 source "/home/nazareth/nuttxspace/apps/examples/watched/Kconfig"
212 source "/home/nazareth/nuttxspace/apps/examples/watcher/Kconfig"
213 source "/home/nazareth/nuttxspace/apps/examples/webserver/Kconfig"
214 source "/home/nazareth/nuttxspace/apps/examples/wgetjson/Kconfig"
215 source "/home/nazareth/nuttxspace/apps/examples/wget/Kconfig"
216 source "/home/nazareth/nuttxspace/apps/examples/wiegand/Kconfig"
217 source "/home/nazareth/nuttxspace/apps/examples/ws2812esp32rtt/Kconfig"
218 source "/home/nazareth/nuttxspace/apps/examples/ws2812/Kconfig"
219 source "/home/nazareth/nuttxspace/apps/examples/xb test/Kconfig"
220 source "/home/nazareth/nuttxspace/apps/examples/xedg demo/Kconfig"
221 source "/home/nazareth/nuttxspace/apps/examples/xmlrpc/Kconfig"
222 source "/home/nazareth/nuttxspace/apps/examples/zerocross/Kconfig"
223
224 source "/home/nazareth/nuttxspace/apps/examples/CC8_PROGRAMS/Kconfig"
225
226
227
228 endmenu # Examples
229
```

**Figura 8:** Archivo Kconfig de la carpeta Examples

Luego se valida la visibilidad de la aplicación en menuconfig:



**Figura 9:** Aplicaciones agregadas a menuconfig.

## 5.2 Configuración para ESP32-S3

### 5.2.1 Instalación del entorno de compilación

Repositorios base:

```
mkdir ~/nuttxspace && cd ~/nuttxspace  
git clone https://github.com/apache/incubator-nuttx.git nuttx  
git clone https://github.com/apache/incubator-nuttx-apps.git apps  
git clone https://bitbucket.org/nuttx/tools.git
```

Herramientas de configuración:

```
cd tools/kconfig-frontends  
./configure --enable-mconf  
make  
sudo make install  
sudo ldconfig
```

Dependencias del sistema:

```
sudo apt-get install kconfig-frontends automake bison build-essential flex gperf git  
libncurses5-dev libtool libusb-dev libusb-1.0.0-dev pkg-config
```

5.2.2 Toolchain para ESP32-S3

Se instaló la toolchain:

```
curl https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8_2_0-esp-2020r2-linux-amd64.tar.gz |  
tar -xz  
sudo mv xtensa-esp32-elf/ /opt/xtensa/  
export PATH=$PATH:/opt/xtensa/xtensa-esp32-elf/bin  
pip3 install esptool
```

### 5.2.3 Configuración base de Nuttx con WiFi

Se empleó la configuración de referencia:

```
./tools/configure.sh esp32s3-devkit:wifi
```

### 5.2.4 Compilación y carga

Compilación:

```
make -j$(nproc)
```

Carga del sistema:

```
make download ESPTOOL_PORT=/dev/ttyACM0 ESPTOOL_BAUD=115200
```

Conexión vía minicom:

```
sudo minicom -D /dev/ttyACM0 -b 115200
```

## 5. Seguimiento de investigación

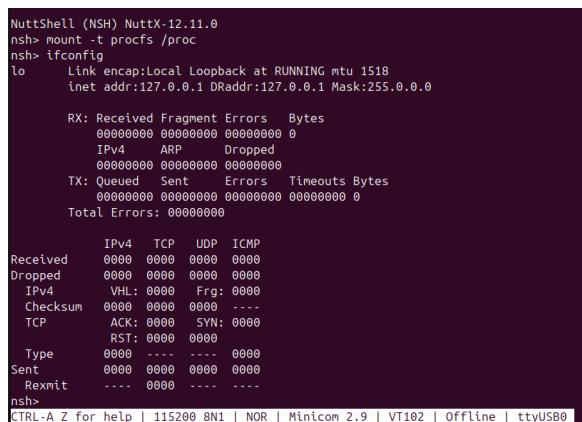
Esta sección documenta el proceso técnico seguido para habilitar la comunicación entre la computadora host y la BeagleBone Black ejecutando NuttX, por lo que incluye la verificación del estado real del sistema, los errores encontrados, las pruebas realizadas y las soluciones aplicadas, el propósito es dejar un registro claro y reproducible del camino seguido durante la investigación.

### 5.1 Verificación inicial de capacidades de red en Nuttx

El primer paso fue determinar si el build de NuttX disponible para la BeagleBone Black permitía la comunicación mediante sockets AF\_INET y, específicamente, RAW sockets, ya que estos eran necesarios para la propuesta inicial del laboratorio.

Durante la revisión de configuración se confirmó que el sistema incluía:

- Soporte general de red.
- Stack TCP/IP habilitado.
- Interfaz de loopback operativa.
- Work queues del kernel funcionando correctamente.

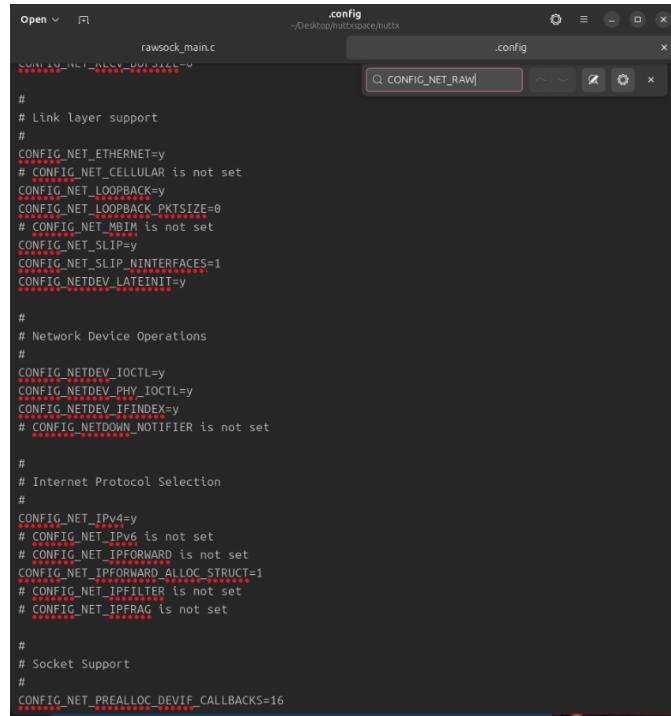


```
NuttShell (NSH) NuttX-12.11.0
nsh> mount -t procfs /proc
nsh> ifconfig
lo    Link encap:Local Loopback at RUNNING mtu 1518
      inet addr:127.0.0.1 Bcast:127.0.0.1 Mask:255.0.0.0
             inet6 addr: ::/128 Scope:Host
                 inet6 mcast: ff02::1/16 Scope:Link
      RX: Received Fragment Errors Bytes
          00000000 00000000 00000000 0
          IPv4   ARP   Dropped
          00000000 00000000 00000000
      TX: Queued Sent Errors Timeouts Bytes
          00000000 00000000 00000000 00000000 0
          Total Errors: 00000000

          IPv4   TCP   UDP   ICMP
Received  0000  0000  0000  0000
Dropped   0000  0000  0000  0000
      IPv4   VHL: 0000  Trg: 0000
      Checksum 0000  0000  0000  ----
      TCP    ACK: 0000  SYN: 0000
          RST: 0000  0000
      Type   0000  ----  0000
      Sent   0000  0000  0000  0000
      ReTxmt 0000  ----  ----
nsh>
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.9 | VT102 | Offline | ttyUSB0
```

Figura 10: Verificación de loopback activo.

Sin embargo, al inspeccionar el archivo.config, y realizar una prueba directa de creación de sockets, se verificó que el build no incluía soporte para RAW sockets



The screenshot shows a terminal window with two tabs open: 'rawsock\_main.c' and '.config'. The '.config' tab is active and displays the NuttX configuration file. A search bar at the top right contains the query 'Q CONFIG\_NET\_RAW'. The search results highlight several lines related to network raw socket support:

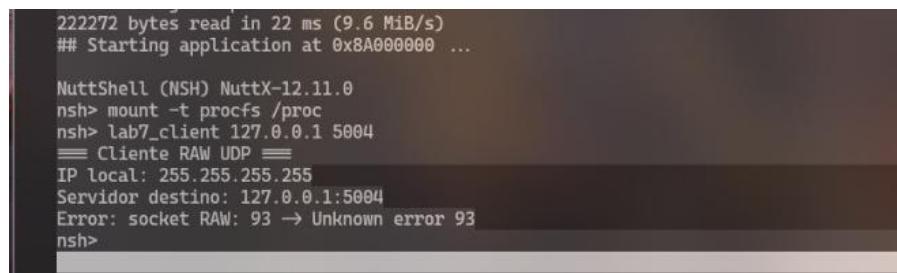
```
# Link layer support
#
# CONFIG_NET_ETHERNET=y
# CONFIG_NET_CELLULAR is not set
CONFIG_NET_LOOPBACK=y
CONFIG_NET_LOOPBACK_PKTSIZE=0
# CONFIG_NET_MBTM is not set
CONFIG_NET_SLIP=y
CONFIG_NET_SLIP_NINTERFACES=1
CONFIG_NETDEV_LATEINIT=y

#
# Network Device Operations
#
CONFIG_NETDEV_IOCTL=y
CONFIG_NETDEV_PHY_IOCTL=y
CONFIG_NETDEV_IFINDEX=y
# CONFIG_NETDOWN_NOTIFIER is not set

#
# Internet Protocol Selection
#
CONFIG_NET_IPv4=y
# CONFIG_NET_IPv6 is not set
# CONFIG_NET_ICMPv6 is not set
CONFIG_NET_IFFORWARD_ALLOC_STRUCT=1
# CONFIG_NET_IPFILTER is not set
# CONFIG_NET_IPFRAG is not set

#
# Socket Support
#
CONFIG_NET_PREAMLOC_DEVIF_CALLBACKS=16
```

Figura 11: Búsqueda de CONFIG\_NET\_RAW en .config.



The screenshot shows a terminal window running the NuttShell (NSH) on NuttX. The user has mounted the procfs filesystem and connected to a client at 127.0.0.1:5004. They attempt to create a RAW socket but receive an error message:

```
222272 bytes read in 22 ms (9.6 MiB/s)
## Starting application at 0x8A000000 ...

NuttShell (NSH) NuttX-12.11.0
nsh> mount -t procfs /proc
nsh> lab7_client 127.0.0.1 5004
== Cliente RAW ==
IP local: 255.255.255.255
Servidor destino: 127.0.0.1:5004
Error: socket RAW: 93 → Unknown error 93
nsh>
```

Figura 12: Error 93 returned al intentar crear un socket RAW.

El resultado determinó que la BeagleBone Black, bajo esta versión y configuración de NuttX, no podía ejecutar comunicación basada en RAW sockets. Esto imposibilitó continuar con el enfoque planteado inicialmente y obligó a replantear el mecanismo de comunicación.

## 5.2 Cambio hacia comunicación por UART

Debido a la ausencia de soporte para RAW sockets, se optó por migrar la comunicación hacia UART utilizando un módulo USB–Serial para establecer un enlace directo con la computadora host.

Durante este proceso surgió un error en la compilación al habilitar UART1, puesto que un manejador de interrupciones no estaba implementado en el archivo correspondiente.

```
CC: socket/getpeername.c chip/am335x_serial.c:198:4: error: 'struct up_dev_s' has no member named 'handler'  
198 |     .handler      = uart1_interrupt,  
|     ^-----  
CC: ctype/lib_ctype.c chip/am335x_serial.c:198:21: error: 'uart1_interrupt' undeclared here (not in a function); did you mean 'uart_interrupt'?  
198 |     .handler      = uart1_interrupt,  
|     ^-----  
|           uart_interrupt
```

**Figura 13:** Error de compilación

La solución consistió en comentar la línea involucrada, lo cual permitió completar el build sin afectar el funcionamiento básico del puerto serie.

## 5.3 Pruebas y estabilización de la comunicación

Con el enlace UART habilitado, se inició una serie de pruebas de transmisión y recepción utilizando NuttShell en la BeagleBone Black a través de minicom y un cliente serial en la computadora host. Durante esta etapa se observaron problemas recurrentes:

- Recepción de caracteres inválidos o corruptos.
- Datos residuales en el buffer de entrada.
- Bloqueos al intentar abrir el mismo puerto desde dos procesos (acceso exclusivo).
- Cambios en la asignación de puertos ttyUSB\* al reconnectar el módulo FTDI.

Para resolver estos problemas se aplicaron las siguientes medidas:

- Limpieza explícita del buffer antes de procesar nuevos datos.
- Separación estricta de puertos:
  - Un puerto dedicado al entorno NuttX (servidor).
  - Un puerto destinado para el cliente serial en la PC.
- Implementación de una regla udev para asignar un nombre persistente al dispositivo FTDI, evitando variaciones entre reconexiones.

Tras estas correcciones, la comunicación UART alcanzó un funcionamiento estable, con intercambio claro, reproducible y sin errores entre ambos extremos.

## 6. Resultados

### 6.1 Resultados en BeagleBone Black

```
== Cliente UART ==
Usando puerto /dev/ttyUSB0 @ 115200
Ingrese operación (ej: 3 + 2) o EXIT: 3 + 3
Respuesta del servidor: 6

Ingrese operación (ej: 3 + 2) o EXIT: 599 + 101
Respuesta del servidor: 700

Ingrese operación (ej: 3 + 2) o EXIT: 10 / 2
Respuesta del servidor: 5

Ingrese operación (ej: 3 + 2) o EXIT: 100 / 2
Respuesta del servidor: 50

Ingrese operación (ej: 3 + 2) o EXIT: 5 * 4
Respuesta del servidor: 20

Ingrese operación (ej: 3 + 2) o EXIT: 8 * 8
Respuesta del servidor: 64

Ingrese operación (ej: 3 + 2) o EXIT: 100 - 1
Respuesta del servidor: 99

Ingrese operación (ej: 3 + 2) o EXIT: 3 - 2
Respuesta del servidor: 1

Ingrese operación (ej: 3 + 2) o EXIT: exit
```

**Figura 14:** Ejecución del cliente UART desde la computadora host.

```
## Starting application at 0x8A000000 ...

NuttShell (NSH) NuttX-12.11.0
nsh> lab_serial_server
== Servidor UART NuttX ==
Escuchando en /dev/ttyS1 @ 115200 baud
Recibido: 3 + 3
Recibido: 599 + 101
Recibido: 10 / 2
Recibido: 100 / 2
Recibido: 5 * 4
Recibido: 8 * 8
Recibido: 100 - 1
Recibido: 3 - 2
Recibido: exit
nsh> [REDACTED]
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.9 | VT102 | Offline | ttyUSB1
```

**Figura 15:** Servidor UART ejecutándose en NuttX desde BeagleBone Black.

## 6.2 Resultados en ESP32-S3

```
nsh> client_udp 192.168.1.10 5004  
===== Cliente UDP Multi-Servidor =====  
Total de servidores: 1  
Servidor #1 => 192.168.1.10:5004  
Ingrese operación (EXIT para salir): 9 - 2  
Respuesta de 192.168.1.21:5004 => 7  
Ingrese operación (EXIT para salir): EXIT  
Cliente terminado.
```

Figura 16: Cliente corriendo en NuttX sobre ESP32-S3.

```
▲ ~/Labs_CC8/Lab7 ⌂ main ◉ !? > make run-server  
java Server  
Servidor escuchando en UDP puerto: 5004  
> 192.168.1.216 client [2025/11/26 00:22:16] UDP: 9 - 2  
< 192.168.1.10 server [2025/11/26 00:22:16] UDP: 7
```

Figura 17: Servidor ejecutado en laptop.

## 7. Conclusiones

El desarrollo del proyecto permitió comprender de manera directa las capacidades reales y las limitaciones del port de NuttX para la BeagleBone Black, a lo largo del proceso se evidenciaron fallos de soporte en el subsistema Ethernet, ausencia de RAW sockets, errores en drivers y configuraciones incompletas, lo que obligó a replantear la arquitectura inicial y adoptar UART como medio principal de comunicación.

Cada obstáculo proporcionó información valiosa sobre la estructura interna del sistema operativo, su ciclo de compilación y los requisitos necesarios para activar dispositivos y servicios en plataformas con soporte parcial.

De forma complementaria, el uso de la ESP32-S3 permitió validar el comportamiento completo del stack TCP/IP de NuttX y confirmar la funcionalidad del diseño de sockets originalmente planteado, incluyendo conectividad Wi-Fi, loopback y comunicación con dispositivos externos.

En conjunto, el trabajo realizado no solo documenta un proceso de investigación reproducible, sino que también establece una base técnica clara para futuros desarrollos, migraciones o ampliaciones dentro del ecosistema NuttX y su integración con hardware embebido.

## 8. Referencias

- Monteiro, S. (1 de Diciembre de 2020). *Getting Started with ESP32 and NuttX*. Obtenido de Medium: <https://medium.com/the-esp-journal/getting-started-with-esp32-and-nuttx-fd3e1a3d182c>
- The Apache Software Foundation. (2020). *Configuration Settings*. Obtenido de NuttX: [https://nuttx.apache.org.translate.goog/docs/10.0.0/components/nsh/config.html?\\_x\\_tr\\_sl=auto&\\_x\\_tr\\_tl=es&\\_x\\_tr\\_hl=es-419](https://nuttx.apache.org.translate.goog/docs/10.0.0/components/nsh/config.html?_x_tr_sl=auto&_x_tr_tl=es&_x_tr_hl=es-419)
- The Apache Software Foundation. (2023). “Raw” packet socket support. Obtenido de NuttX: <https://nuttx.apache.org/docs/latest/components/net/pkt.html>
- The Apache Software Foundation. (2023). *beaglebone-black*. Obtenido de NuttX: <https://nuttx.apache.org/docs/latest/platforms/arm/am335x/boards/beaglebone-black/index.html>
- The Apache Software Foundation. (2023). *Commands*. Obtenido de NuttX: <https://nuttx.apache.org/docs/latest/applications/nsh/commands.html>
- The Apache Software Foundation. (2023). *ESP32S3-DevKit*. Obtenido de NuttX: <https://nuttx.apache.org/docs/latest/platforms/xtensa/esp32s3/boards/esp32s3-devkit/index.html#>
- The Apache Software Foundation. (2023). *Network Interfaces*. Obtenido de Nuttx: [https://nuttx.apache.org/docs/latest/reference/user/11\\_network.html](https://nuttx.apache.org/docs/latest/reference/user/11_network.html)