Proyecto ARQ

IIE, Departamento de Telecomunicaciones

Grupo **ARTES**

Martín Randall
Paola Romero

Orientador:
Víctor González-Barbone

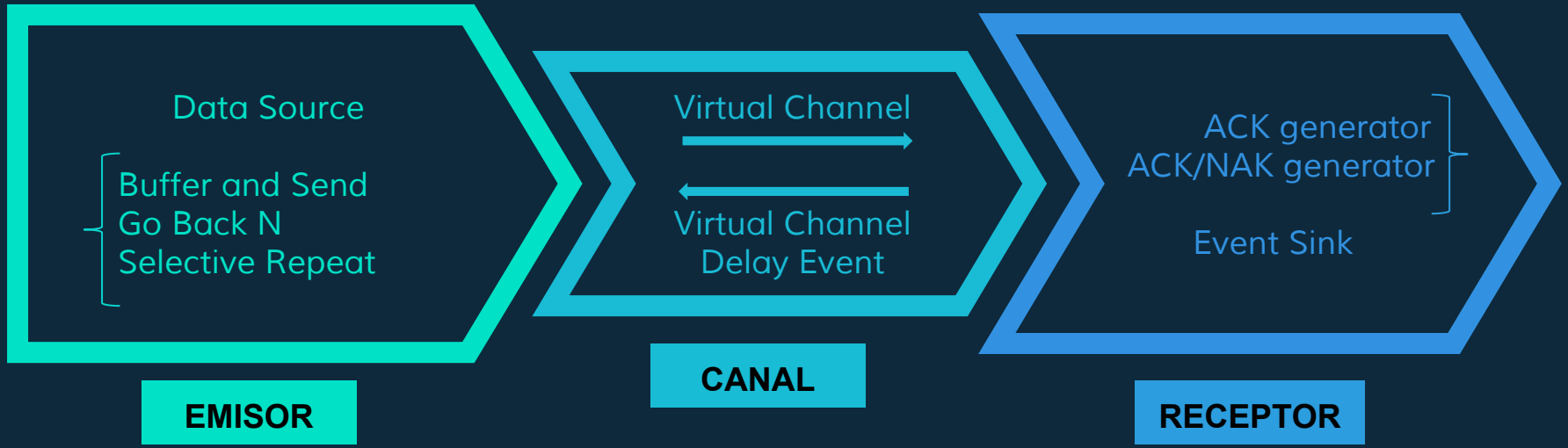
Propósito

- Evaluar el conjunto de bloques disponibles en gwn y colaborar con el desarrollo de una librería apropiada.
- Estudiar el método de transmisión de datos Automatic Repeat Request (ARQ)
- Implementar sus variantes en ejemplos de GRC
- Comprobar su funcionamiento y carencias

Protocolos estudiados e implementados

Modelado del sistema

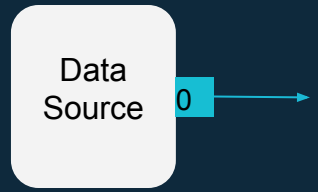
Los protocolos mencionados son algunos de los mecanismos que se utilizan en capa 2 para el control de errores. Nuestra implementación se centra en asegurar que los eventos enviados se reciben en su totalidad y en el orden correcto.



1 stop-and-wait

Bloques implementados

data_source



Bloque generador de eventos de tipo 'Data'. Cada evento tiene un identificador único (campo payload) que se utilizará como número de secuencia en etapas posteriores. Primera etapa de transmisión.

PARÁMETROS

time_out

tiempo de generación de eventos o tiempo de generación máximo.

rand

setea generación de eventos periódica o aleatoria (entre 0 y time_out)

retry

cantidad de eventos a producir

MODALIDADES DE FUNCIONAMIENTO

Generación de eventos periódica cada tiempo time_out → rand=False

Generación aleatoria de eventos → rand=True

El tiempo de generación de un evento corresponderá a un número aleatorio sorteado entre 0 y time_out.

PRUEBA

data_source -> meessage_debug

buff_and_snd

Envía el evento recibido y lo guarda, esperando por su correspondiente ACK antes de enviar el siguiente. Si pasado determinado tiempo no se recibe el ACK, se retransmite el evento.

Segunda etapa de transmisión.

PARÁMETROS

time_out

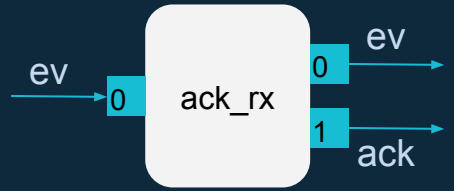
tiempo de expiración. Pasado este tiempo se reenvía el evento correspondiente

buff_tam

tamaño del buffer de almacenamiento

Se utiliza para almacenar los eventos que llegan, para no tener pérdidas significativas

ack_rx



Primera etapa de recepción.

Envía el evento recibido por su puerto 0 y emite el reconocimiento correspondiente por el puerto 1.

El ACK es un evento de tipo 'TimerACKTout' y lleva la información del número de secuencia del evento recibido.

PRUEBA

```
data_source   ->  ack_rx   ->   event_sink1
                  ack_rx   ->   event_sink2
```

delay _ ev

Introduce un retardo temporal que toma un valor aleatorio en determinado rango

PARÁMETROS

delay_min

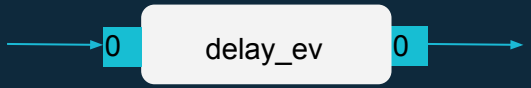
mínimo retardo a introducirse

delay_add

define el máximo valor de retardo.

El retardo temporal será de un valor aleaotorio en el rango:

[delay_min, delay_min+delay_add]



Para no introducir retardo:
delay_min=0, delay_add=0

Para introducir un retardo fijo
delay_add=0 y delay_min=X

virtual _ channel *by Vagonbar*

Bloque disponible en gwn.

Simula las pérdidas en un canal. Se introduce una probabilidad de pérdida que define si un evento pasa o no por el canal mediante el sorteo de un número aleatorio.

PARÁMETRO

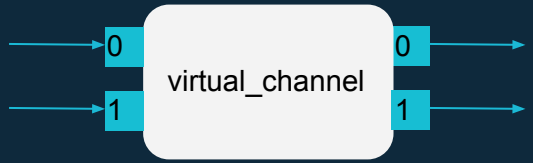
prob_loss

probabilidad de pérdida

ENTRADAS / SALIDAS

Puerto 0 → Eventos

Puerto 1 → PDUs



event _ sink *by Vagonbar*

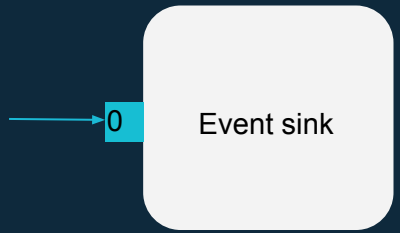
Bloque disponible en gwn.

Receptor de eventos. Muestra en pantalla información sobre el evento recibido.

INFORMACIÓN DEL EVENTO

--- blkname, received ev: ev.nickname
payload: ev.payload

--- blkname, received ev: ev.nickname
seq nr: ev.ev_dc['seq_nr']



Protocolo stop-and-wait

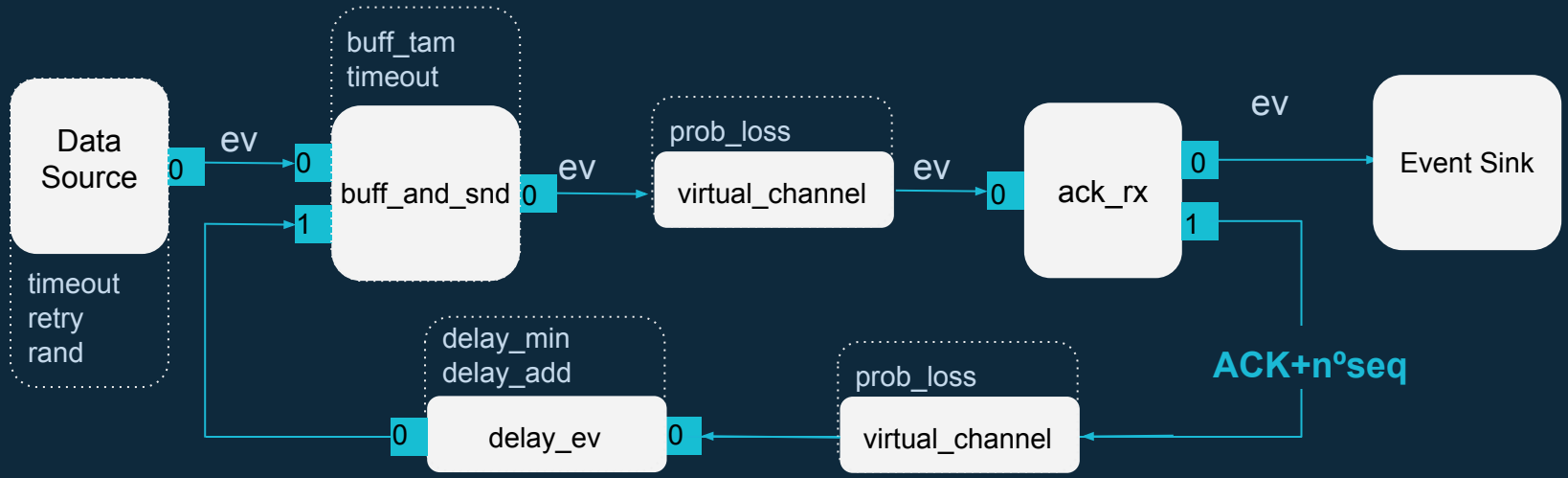


DIAGRAMA DE PRUEBA

2

go-back-N

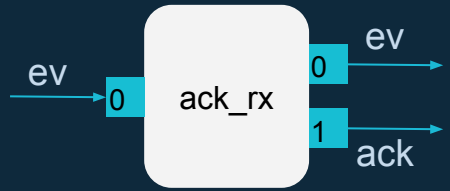
Bloques implementados

ack_rx (modificación)

Envía el evento recibido por su puerto 0 y el correspondiente ACK por el puerto 1.

Se realizaron modificaciones a la versión anterior para adaptarlo a este protocolo.

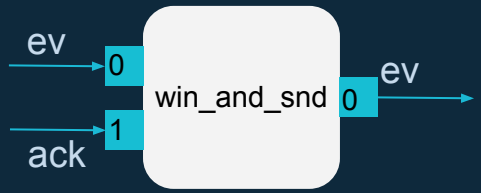
- → Evento repetido: sólo se reenvía el ACK con el número de secuencia correspondiente.
- → Evento nuevo: se verifica que el número de secuencia del evento que llega sea el esperado. Si es correcto, se envía el evento por el puerto 0 y el ACK por el puerto 1.



win_and_snd

A medida que llegan datos, se envían y se guardan en la ventana de emisor. Si se llena la ventana, se guardan en un buffer. Si este último se llena, se descartan los eventos.

Se inicia un timeout por cada evento enviado de la ventana de transmisión. Se podrán tener hasta win_tam timeouts activos.



PARÁMETROS

win_tam

tamaño de ventana de transmisión

buff_tam

tamaño del buffer de almacenamiento

time_out

tiempo de expiración. Pasado este tiempo, se reenvía evento correspondiente

Protocolo go-back-N

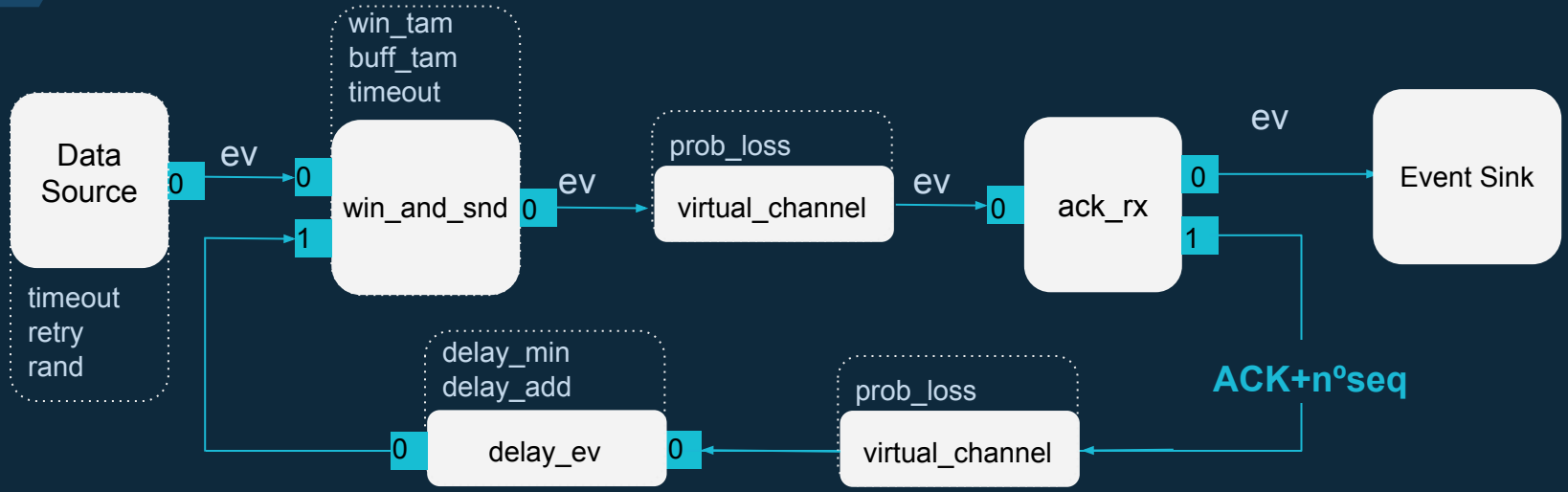


DIAGRAMA DE PRUEBA

3

Selective Repeat

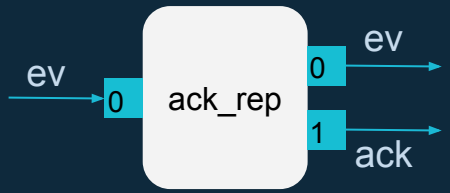
Bloques implementados

ack_rep

Cumple la función de ventana de recepción y se encarga de enviar los ACK de los eventos que llegan correctamente y los NAK de aquellos que faltan. Además se encarga de transmitir en orden, por su puerto 0, los eventos recibidos.

Se utilizaron dos listas de python:

- ➔ buffer: almacena eventos que llegan al bloque pero que aún no están listos para ser enviados en orden.
- ➔ bool: lista de booleanos que tiene una correspondencia biunívoca con buffer. Se utiliza para identificar eventos no recibidos y enviar el NAK correspondiente.



PARÁMETRO

win_tam

tamaño de ventana de recepción

sel_rep

A medida que llegan los eventos, se envían, se guardan en la ventana de emisor y se inicia un timeout. Cuando se llena la ventana, los eventos nuevos se guardan en el buffer y si éste se llena, se descartan.

Diccionario	
payload	nº timeout
	0
	1
	2
	...

	loob
0	False
1	True
2	False
.	.
.	.
.	.
.	.
win_tam	False

Indica si el timeout i está disponible

Número de timeout i

Protocolo selective repeat

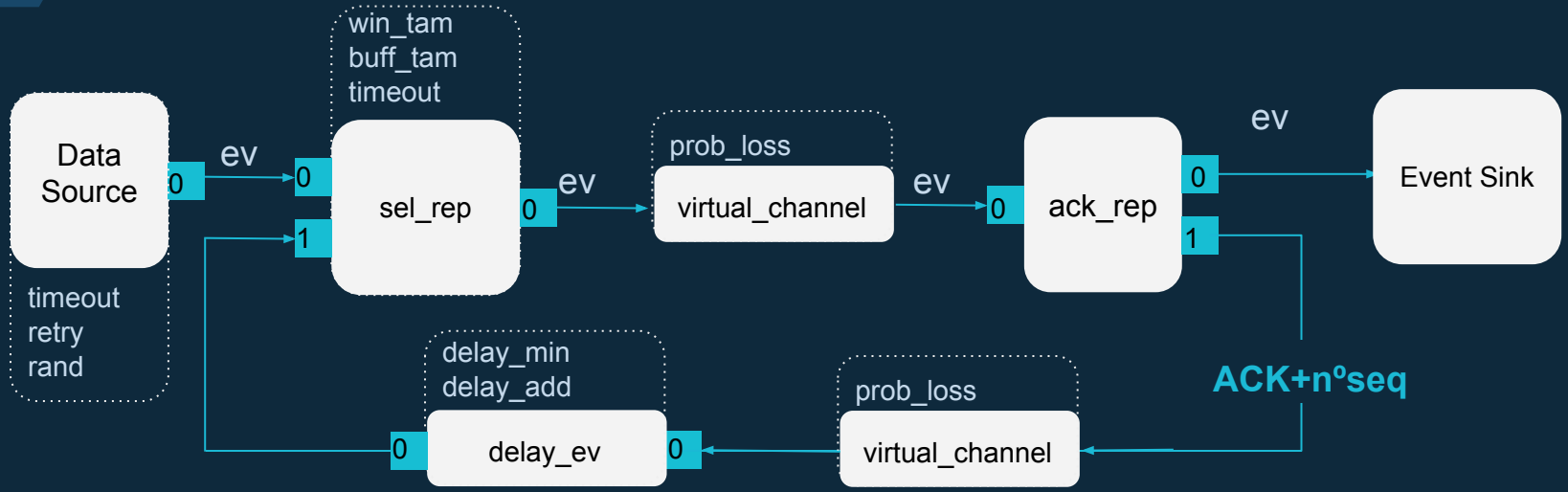


DIAGRAMA DE PRUEBA

Implementaciones

Se realizó la implementación para los 3 protocolos estudiados, utilizando los dispositivos USRP B 100. Para esto tuvimos que codificar los PDU a mensajes, y modularlos utilizando QPSK.

Los bloques de modulación y demodulación, así como los conversores de PDU a eventos fueron desarrollados por Víctor Gónzalez, trabajando nosotros la interconexión de los mismos y la adaptación de nuestros bloques.

La adaptación consistió en que en la construcción original, los números de secuencia se enviaban en la carga útil, cuando realmente esta no debe ser modificada, sino simplemente transportada. Ahora los mensajes incluyen un diccionario que los vincula al número de secuencia, liberando la carga útil para su uso correcto.

A continuación mostraremos la implementación concreta del protocolo Selective Repeat.

Selective Repeat

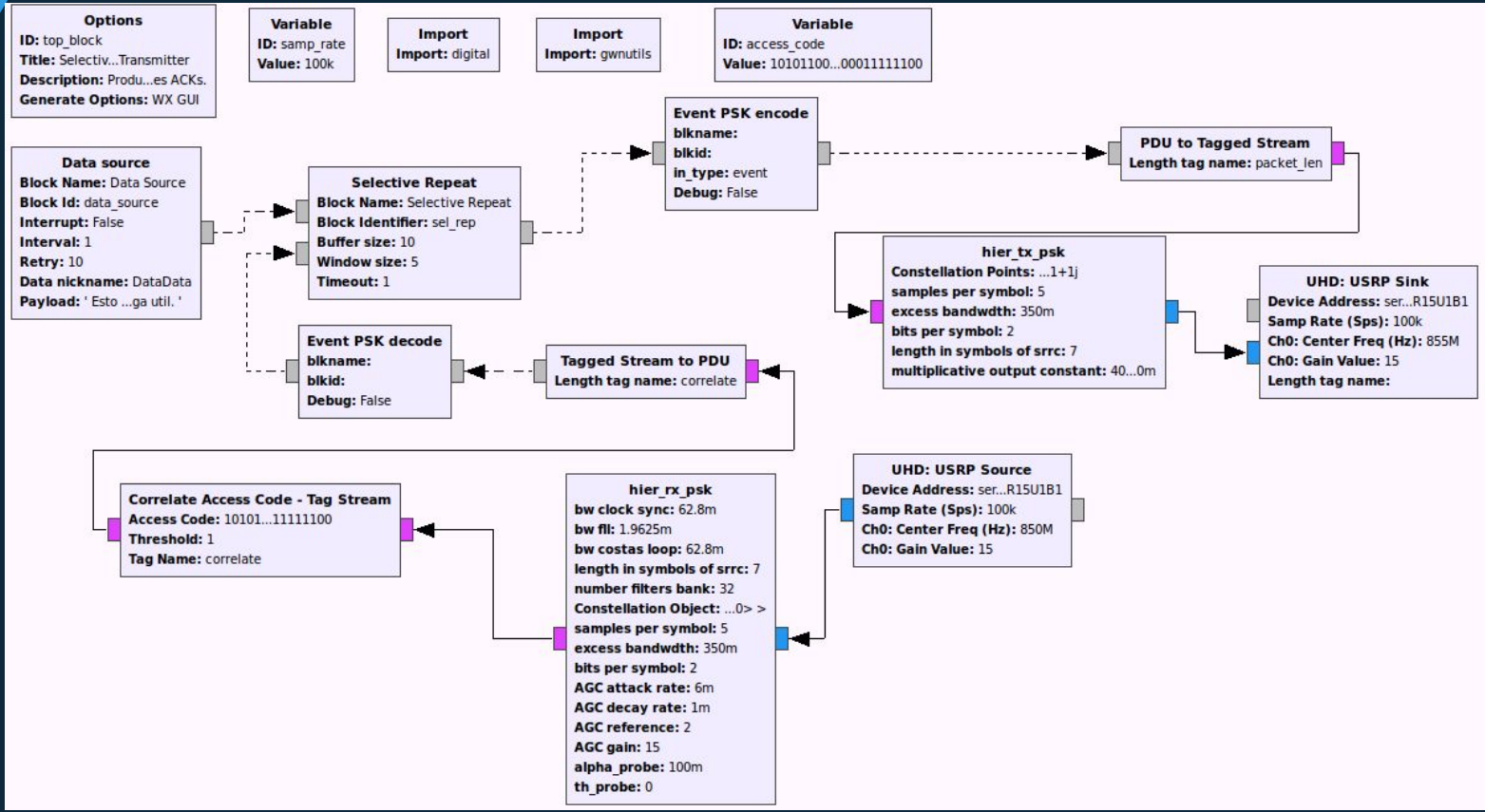
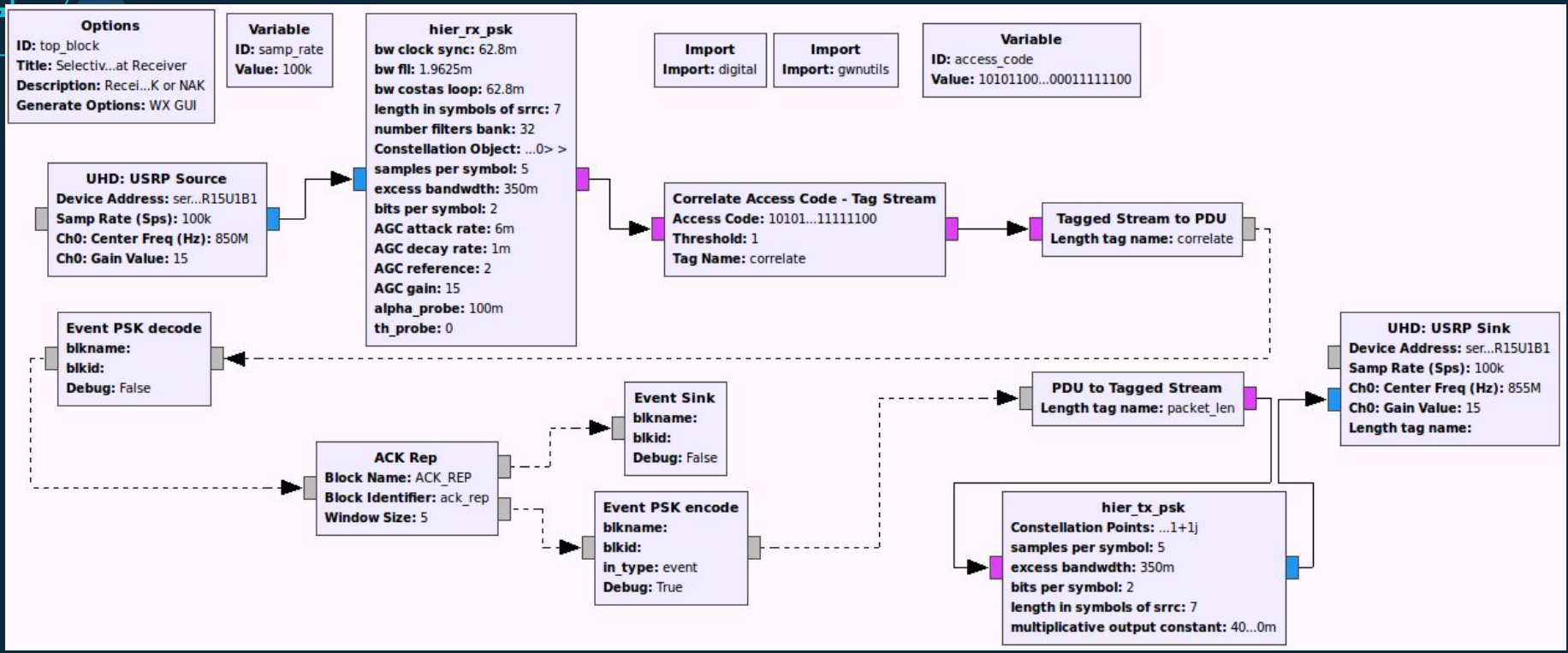


Diagrama en GRC del Transmisor del Selective Repeat

Selective Repeat

Diagrama en GRC del Receptor del Selective Repeat



Resultado de la prueba

Transmitiendo a las frecuencias de 850 y 855 MHz, estando los USRP a una distancia de dos metros, pudimos recibir correctamente la totalidad de los mensajes enviados, así como los ACKs de confirmación.

Vale la pena aclarar que no se realizó una elección específica de las antenas para la frecuencia utilizada (estando estas en el límite, en un caso de 850 a 6500 MHz, y en el otro de 900 a 2600 MHz).

```
   Data 2 repeated, resending ACK to Source.
UU   Sending events to sink from 4 to 4
   Sending ACK corresponding to event number 4
--- ev_sink, received ev: DataData, seq nr: 4
    payload: ' Esto es la carga util. This is the payload. C'est la charge utile. E a carga util. '
U   Data 2 repeated, resending ACK to Source.
U   Data 2 repeated, resending ACK to Source.
U   Sending events to sink from 5 to 5
   Sending ACK corresponding to event number 5
--- ev_sink, received ev: DataData, seq nr: 5
    payload: ' Esto es la carga util. This is the payload. C'est la charge utile. E a carga util. '
   Invalid CRC, discarding packet
U   Invalid CRC, discarding packet
   Invalid CRC, discarding packet
 Event 8 is in reception window  // Last event arrived to sink is 5
   Sending ACK corresponding to event number 8
   Sending NAK corresponding to event number 6
   Sending NAK corresponding to event number 7
   Invalid CRC, discarding packet
U   Data 2 repeated, resending ACK to Source.
   Invalid CRC, discarding packet
U Event 7 is in reception window  // Last event arrived to sink is 5
   Sending ACK corresponding to event number 7
U   Invalid CRC, discarding packet
   Invalid CRC, discarding packet
   Invalid CRC, discarding packet
 Event 10 is in reception window  // Last event arrived to sink is 5
   Sending ACK corresponding to event number 10
   Sending NAK corresponding to event number 9
 Event 9 is in reception window  // Last event arrived to sink is 5
   Sending ACK corresponding to event number 9
   Data 2 repeated, resending ACK to Source.
   Sending events to sink from 6 to 10
--- ev_sink, received ev: DataData, seq nr: 6
    payload: ' Esto es la carga util. This is the payload. C'est la charge utile. E a carga util. '
--- ev_sink, received ev: DataData, seq nr: 7
    payload: ' Esto es la carga util. This is the payload. C'est la charge utile. E a carga util. '
--- ev_sink, received ev: DataData, seq nr: 8
    payload: ' Esto es la carga util. This is the payload. C'est la charge utile. E a carga util. '
   Sending ACK corresponding to event number 6
--- ev_sink, received ev: DataData, seq nr: 9
    payload: ' Esto es la carga util. This is the payload. C'est la charge utile. E a carga util. '
--- ev_sink, received ev: DataData, seq nr: 10
```

Resultado en recepción

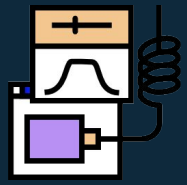
Resultado de la prueba

```
U   Event 9, type DataData arrived to Sel_Rep. Sending event, saving in Win and starting timeout!

 / / Message 8 lost, resending from Sel_Rep DataData \ \

 / / Message 7 lost, resending from Sel_Rep DataData \ \
(8, 1)
(9, 2)
(7, 0)
   ACK correctly received from message 9, type DataData

U(8,1
(7,0

 / / NAK arrived! Message 7 appears to be lost, resending from Sel_Rep DataData
(8,1
(7,0

 / / NAK arrived! Message 8 appears to be lost, resending from Sel_Rep DataData
(8, 1)
(7, 0)
   ACK correctly received from message 8, type DataData

U(7, 0)
   ACK correctly received from message 7, type DataData

   Win empty, event: 10, type DataData arrived to Sel_Rep. Sending event, saving in Win and starting timeout!
U
 / / Message 10 lost, resending from Sel_Rep DataData \ \
U
 / / Message 10 lost, resending from Sel_Rep DataData \ \
U(10, 0)
   ACK correctly received from message 10, type DataData
```

Resultado en transmisión

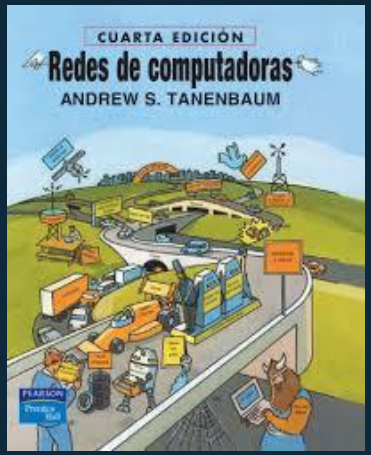
Herramientas

python™



github
SOCIAL CODING

GNURadio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM



UNIX®

¡Gracias!

¿Preguntas?


...Y DÉJENME DECIRLES QUE HAN TOMADO LA DECISIÓN CORRECTA AL ELEGIRME.
SE NOTA QUE SON GENTE QUE SABE LO QUE NECESITA.