Facultade de Informática
UNIVERSIDADE DA CORUÑA

DISEÑO DE REDES

Práctica 4 - 802.11 MAC DCF

Estudante: Tomé Maseda Dorado
Estudante: Jorge Álvarez Cabado

A Coruña, November de 2021.

Índice Xeral

Índice de Figuras

Introducción

En esta práctica se estudiará en profundidad el funcionamiento del protocolo de capa 2 ***IEEE 802.11 DCF (Distributed Coordination Function)*** con y sin uso de paquetes RTS/CTS de señalización.

Se estudiará un escenario en el que hay dos nodos emisores y un nodo receptor, para cada caso, se estudiará lo que ocurre cuando existen nodos ocultos (obstáculo entre los dos nodos emisores) y cuándo no.

Capítulo 1

Ficheros utilizados en la simulación

1.1 Ficheros .ini

```
[General]
network = HiddenNode
sim-time-limit = 5s
seed-set = 11606
record-eventlog = true

**.numApps = 1
**.arp.typename="GlobalArp"

**.radio.displayCommunicationRange = true
**.visualizer.*.displayRoutes = true
**.visualizer.*.displayLinks = true

**.sender1.app[0].typename = "UdpBasicApp"
**.sender1.app[0].destAddresses = "receiver"
**.sender1.app[0].startTime = 0s
**.sender1.app[0].destPort = 5001
**.sender1.app[0].sendInterval = exponential(3ms)

**.sender2.app[0].typename = "UdpBasicApp"
**.sender2.app[0].destAddresses = "receiver"
**.sender2.app[0].startTime = 0s
**.sender2.app[0].destPort = 5001
**.sender2.app[0].sendInterval = truncnormal(3ms, 1ms)

**.receiver.app[0].typename = "UdpSink"
**.receiver.app[0].localPort = 5001

**.messageLength = 500B

```

```
**.wlan[*].bitrate = 9Mbps

[Config _Wall]
*.physicalEnvironment.config = xml("<environment> \
<object position='min 420 170 0' orientation='0 0 0' shape='cuboid
    10 200 40' \
material='concrete'/> \
</environment>")
*.radioMedium.obstacleLoss.typename = "IdealObstacleLoss"

[Config _RTS]
**.wlan[*].mac.dcf.rtsPolicy.rtsThreshold = 100B

[Config No_Wall_No_Rts]

[Config Wall_No_Rts]
extends=_Wall

[Config Wall_Rts]
extends=_Wall,_RTS

[Config No_Wall_Rts]
extends=_RTS
```

Listing 1.1: Fichero omnetpp.ini

1.2 Ficheros .ned

```
import inet.visualizer.contract.IIntegratedVisualizer;
import inet.networklayer.configurator.ipv4.Ipv4NetworkConfigurator;
import
    inet.physicallayer.ieee80211.packetlevel.Ieee80211ScalarRadioMedium;
import inet.environment.common.PhysicalEnvironment;
import inet.node.inet.AdhocHost;


network HiddenNode
{
    parameters:
        @display("bgb=641,420");
    submodules:
        visualizer: <default("IntegratedCanvasVisualizer")> like
    IIntegratedVisualizer if hasVisualizer() {
                @display("p=77,260");
        }
```

```
16          configurator: Ipv4NetworkConfigurator {
17              @display("p=77,180");
18          }
19          radioMedium: Ieee80211ScalarRadioMedium {
20              @display("p=77,100");
21          }
22          physicalEnvironment: PhysicalEnvironment {
23              @display("p=77,340");
24          }
25          sender1: AdhocHost {
26              @display("p=270,320");
27          }
28          sender2: AdhocHost {
29
30              @display("p=570,320");
31          }
32          receiver: AdhocHost {
33            @display("p=420,60");
34          }
35  }
```

Listing 1.2: Fichero HiddenNode.ned

Capítulo 2

Preguntas acerca de la simulación

2.1 Pregunta 1

Simule el escenario 1 y obtenga gráficas en función del tiempo de:

- **Paquetes descartados por receiver por recibirse incorrectamente (packetDropIncorrectlyReceived). Esto se corresponde con las colisiones entre los nodos sender1 y sender2.**

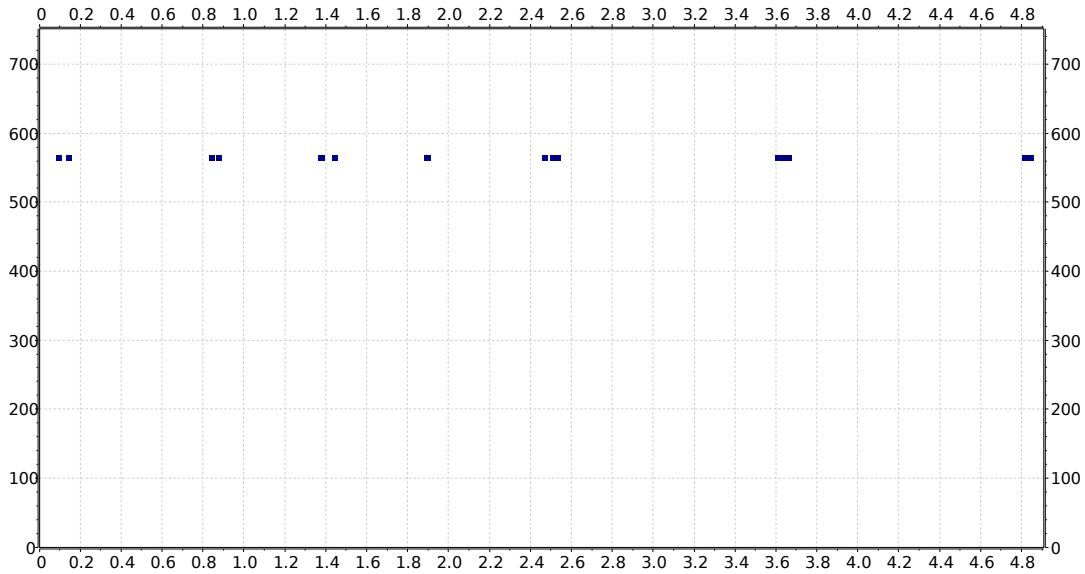

Figura 2.1: Paquetes recibidos incorrectamente por *receiver* (B) vs Tiempo (s)

No hay muchas colisiones, esto se debe a que los nodos *sender1* y *sender2* son visibles el uno para el otro, entonces, cuando un nodo va a transmitir si ve que el medio está ocupado (el otro nodo está transmitiendo), espera. De todas formas, sí que se producen

algunas colisiones, ¿Por qué? Porque si los dos nodos ven el medio libre e intentan transmitir a la vez, se producen colisiones.

- **Contention window (cw) y backoff (backoffSlots) en el nodo sender1.**

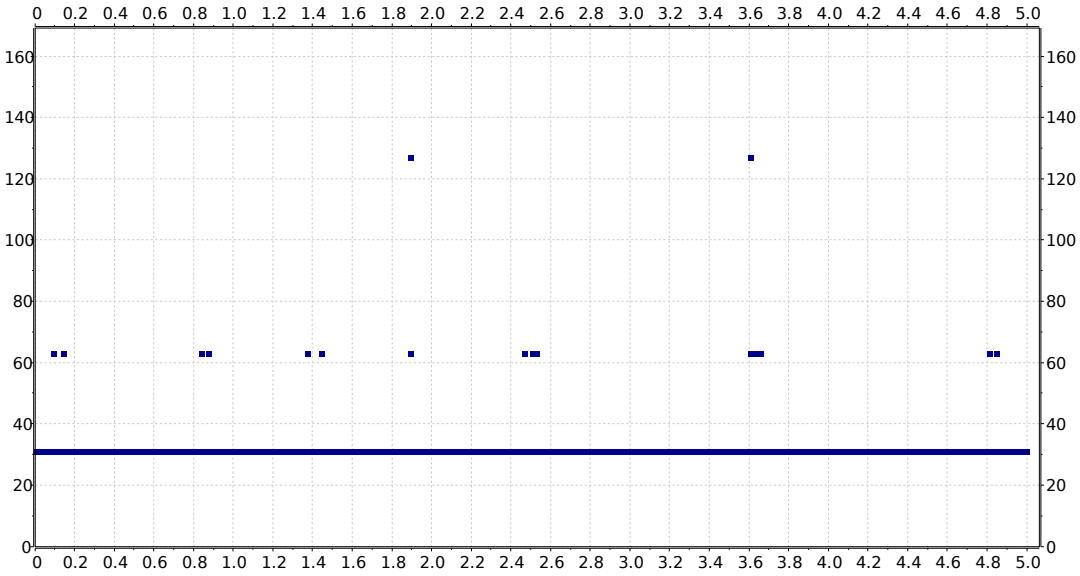

Figura 2.2: *Contention Window* del nodo *sender1* (ms) vs Tiempo (s)

La *Contention Window* (CW) es el máximo tiempo que esperará un nodo para retransmitir un paquete que no se recibió correctamente debido a una colisión, es decir, si hay una colisión, el nodo esperará un tiempo aleatorio (*backoff*) entre 0 y CW ms.

Cada vez que se intenta retransmitir un paquete (debido a una colisión), el CW se duplica, si nos fijamos en la gráfica y la comparamos con la gráfica anterior (Figura 2.1), vemos que en los mismos instantes de tiempo en los que hay paquetes recibidos incorrectamente (colisiones) el tamaño de la CW se duplica, en caso de haber varias colisiones consecutivas se duplica varias veces (como ocurre en t=1.9s aproximadamente). Por otro lado, cuando un paquete se transmite correctamente, la CW vuelve a su tamaño original (CwMin).

Viendo la gráfica podemos ver que la CW tiene un CwMin=31, el CwMax no lo sabemos pero por defecto siempre es 1023. La CW, en este caso, va variando su tamaño de 31 (CwMin) a 62 (1 colisión) y a 124 (2 colisiones consecutivas), por tanto, también sabemos que no ocurren más de 2 colisiones consecutivas en toda la simulación.

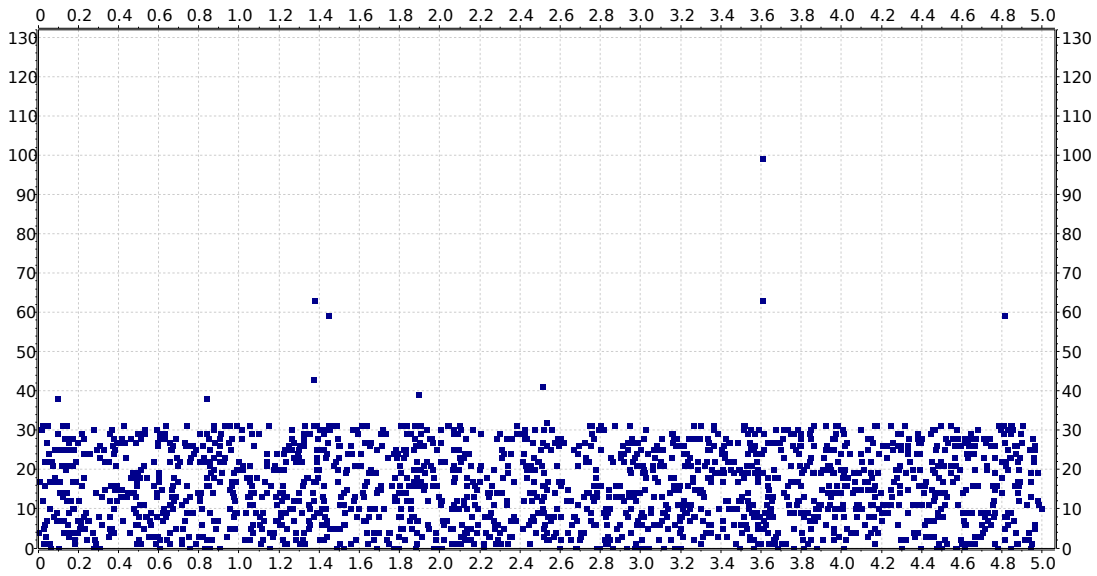

Figura 2.3: Backoff del nodo *sender1* (ms) vs Tiempo (s)

El *backoff* es análogo a la CW, ya que es el tiempo que esperará el nodo para retransmitir un paquete que, como explicamos previamente, es un tiempo aleatorio entre 0 y CW. Por tanto, vemos que la mayoría del tiempo el *backoff* toma valores entre 0 y 31, en los instantes en los que se produce una colisión toma valores entre 0 y 62 y en los instantes en los que se producen dos colisiones consecutivas toma valores entre 0 y 124.

- **Retardo extremo a extremo (endToEndDelay) medido en el receiver. Escala: ms (por defecto está en segundos, por lo tanto: botón derecho → Apply → Other... → multiply-by → 1000).**

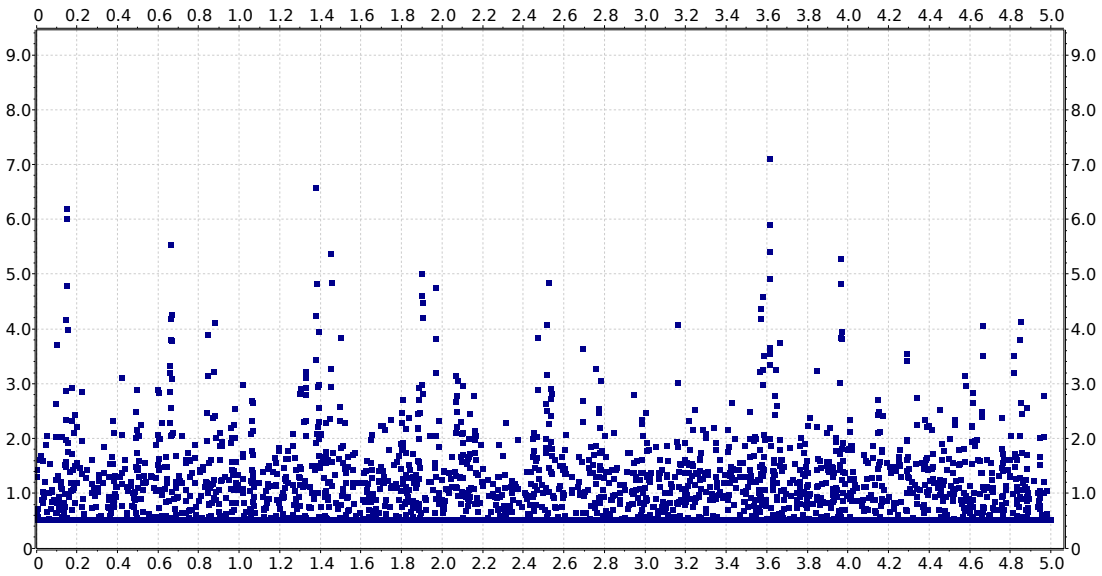

Figura 2.4: Retardo extremo a extremo. Tiempo que tarda un paquete en llegar al nodo *receiver* (ms) vs Tiempo (s)

En este caso el retardo extremo a extremo va a depender principalmente del *backoff*, lo que quiere decir que dependerá de las colisiones. El nodo emisor intentará enviar y, si se produce una colisión, tendrá que esperar un *backoff* para retransmitir, todo ese tiempo es retardo, tiempo que espera el paquete en la cola MAC para poder ser enviado y llegar al nodo *receiver*. Si observamos la gráfica, vemos que los picos de retardo se producen principalmente en los instantes de tiempo en los que aumenta el *backoff*. También aumenta el retardo en otros instantes de tiempo, esto se puede deber a otros problemas de la red como retardos a nivel de capa 1 (errores de transmisión e interferencias).

2.2 Pregunta 2

Obtenga un diagrama de secuencia de la simulación durante una colisión. Para simplificar el diagrama filtre por tipo NED → inet.AdhocHost, elimine los self-message y las dependencias y ordene los nodos verticalmente de forma que el receiver se encuentre entre los dos senders. Muestre el intercambio de paquetes entre los nodos durante la colisión y lo que ocurre tras la colisión (reenvío). A la vista del diagrama, ¿por qué ocurren las colisiones?

Observando el log, vemos que la primera colisión ocurre en t=0.096805983205s, por tanto, simularemos hasta t=0.1s.

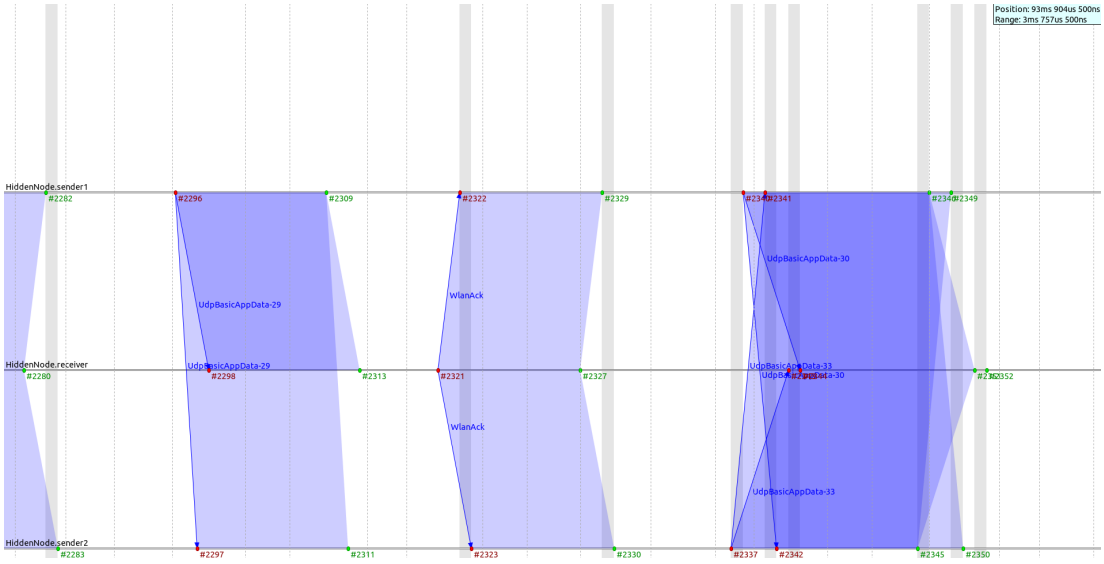

Figura 2.5: Diagrama de secuencia de la colisión producida en t=0.096805983205s

Viendo el diagrama podemos ver exactamente lo que ocurre, en el evento #2296 ***sender1*** envía un datagrama UDP a ***receiver***, después, en el evento #2321 ***receiver*** confirma la recepción del datagrama con un *WlanAck*, por tanto, el medio vuelve a estar libre. Ahora tanto ***sender1*** como ***sender2*** envían un datagrama UDP casi a la vez (eventos #2337 y #2340), lo que está pasando es que primero envía ***sender2*** (evento #2337) y antes de que a ***sender1*** le llegue el datagrama enviado por ***sender2*** (evento #2341), este envía un datagrama (evento #2340), si ***sender1*** hubiese recibido el datagrama UDP de ***sender2*** antes de enviar el suyo, no enviaría nada porque detectaría que el medio está ocupado. Como los dos nodos emisores envían un datagrama UDP a la vez, se produce una colisión en ***receiver*** (evento #2344 - instante en el que recibe el datagrama de ***sender1***) y los paquetes son descartados (hay que retransmitir).

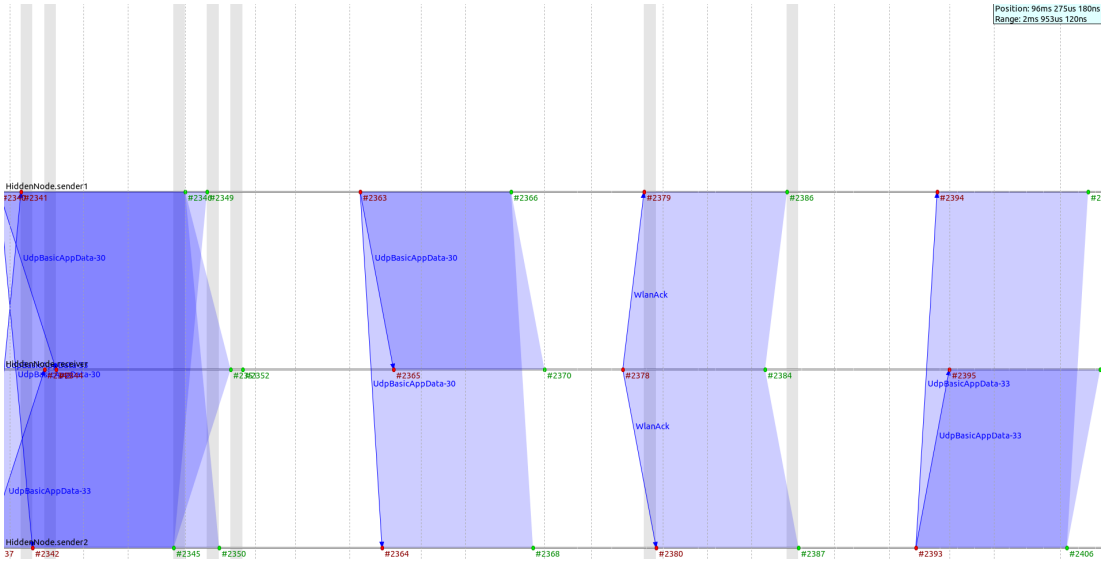

Figura 2.6: Diagrama de secuencia del reenvío de paquetes tras la colisión producida en t=0.096805983205s

Una vez producida la colisión, los dos nodos emisores lo detectan porque no reciben ningún *WlanAck*. Al detectar la colisión los dos nodos doblan su CW de 31ms a 62ms y esperan un *backoff* aleatorio entre 0ms y 62ms, en este caso, el *backoff* de ***sender1*** es más pequeño, entonces transmite él primero, ***receiver*** responde con un *WlanAck* y ***sender2*** envía tras esperar el tiempo de *backoff* residual que le quedaba por esperar, que en este caso sería:
bo1 - *backoff* de *sender1*
bo2 - *backoff* de *sender2*
br2 - *backoff* residual de *sender2*

br2 = *bo2* - *bo1*

Es decir, al *backoff* que tenía inicialmente ***sender2***, se le resta el *backoff* de ***sender1***, porque ya esperó ese tiempo antes (cuando finalmente transmitió ***sender1***).

2.3 Pregunta 3

Simule ahora el escenario 2 y obtenga las 4 gráficas del apartado 1. ¿Qué se observa? ¿Qué diferencias hay con el escenario 1 en cada una?

Ahora el número de colisiones crece muchísimo, esto se debe a que ahora hay un obstáculo entre los dos nodos emisores, *sender1* y *sender2*, ninguno de los dos recibirá los paquetes que envía el otro y, consecuentemente, ninguno de los dos sabrá cuando el medio está ocupado,

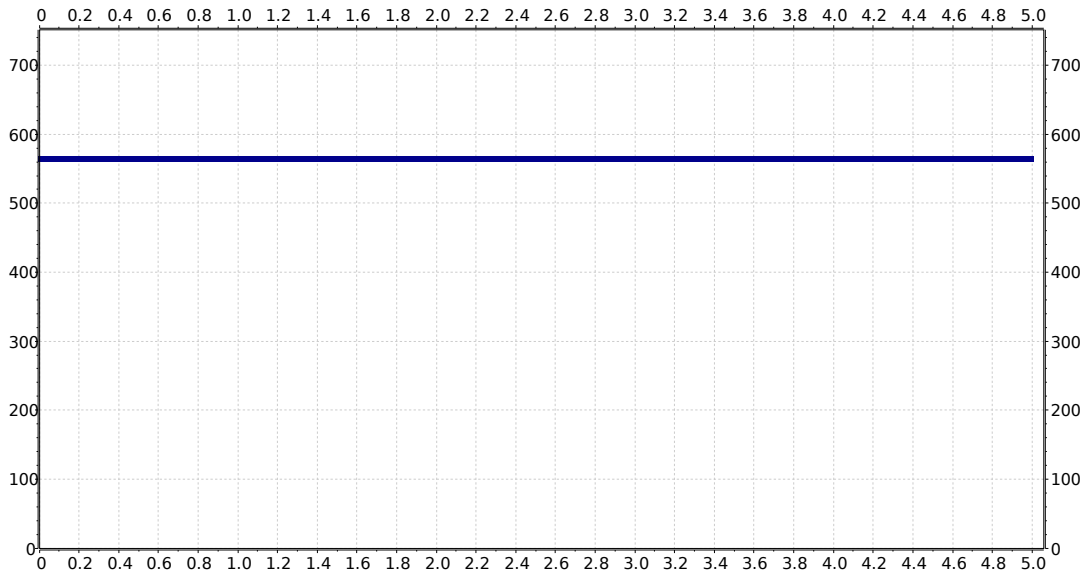

Figura 2.7: Paquetes recibidos incorrectamente por *receiver* (B) vs Tiempo (s)

por tanto, los dos nodos enviarán siempre sin esperar y se estarán produciendo colisiones durante toda la simulación.

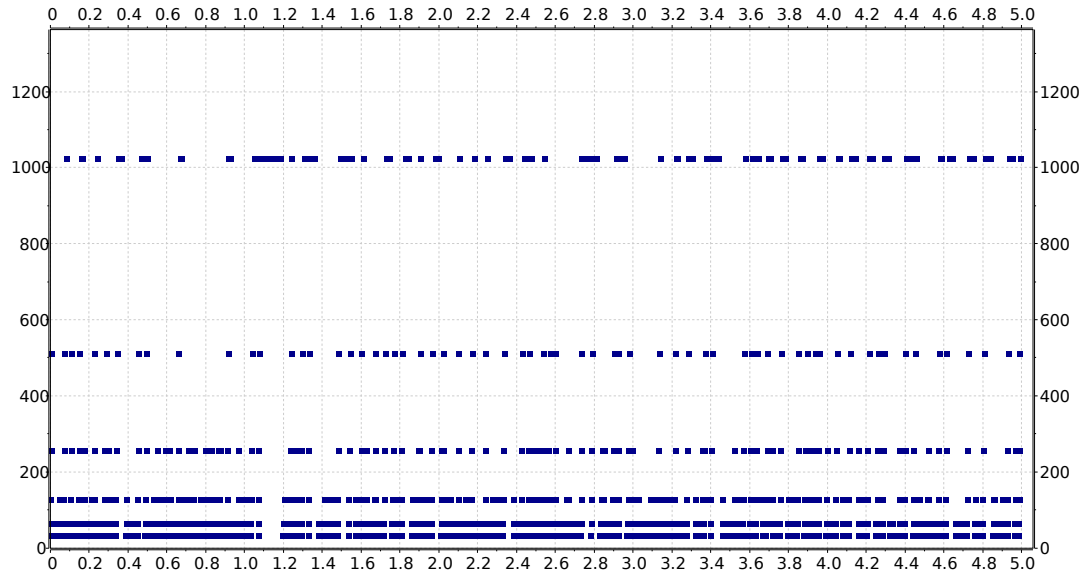

Figura 2.8: *Contention Window* del nodo *sender1* (ms) vs Tiempo (s)

Ahora la *Contention Window* estará en un bucle en el que irá duplicando su duración hasta CwMax (1023ms) y una vez alcance este tamaño se mantendrá ahí hasta que, aleatoriamente, coincida que un paquete se envía exitosamente y su duración se reinicie a CwMin (31ms). Es

decir la duración de CW estará en un bucle de este estilo:

31ms-62ms-124ms-248ms-496ms-1023ms

ACLARACIÓN: Como se ve en la gráfica, en la práctica, no es exactamente así, además, puede coincidir que se envíe un paquete con éxito antes de que la CW llegué a una duración de 1023ms (CwMax), pero la mayor parte del tiempo se producirá un comportamiento similar al descrito.

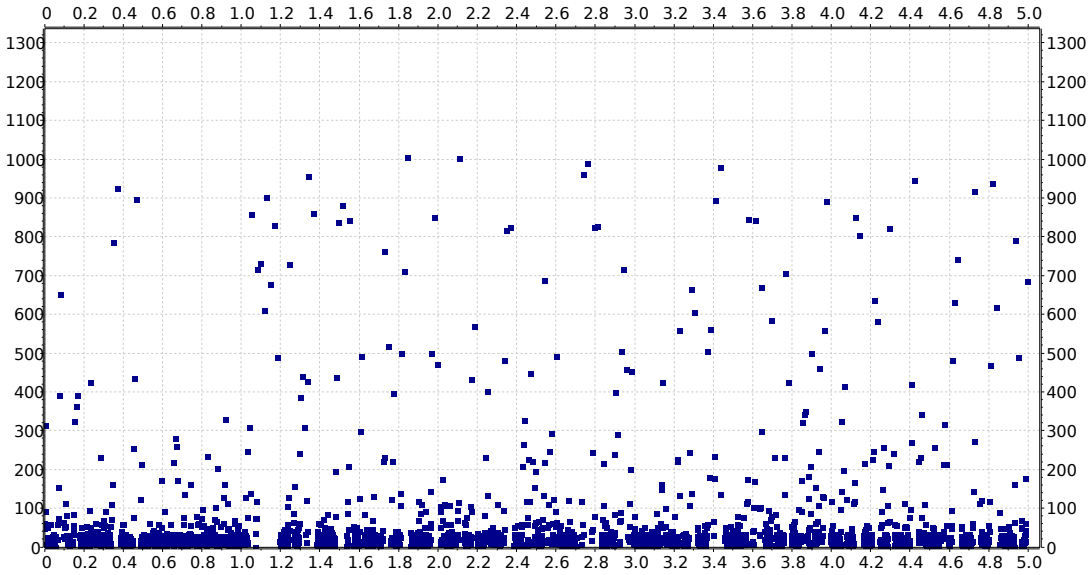

Figura 2.9: Backoff del nodo *sender1* (ms) vs Tiempo (s)

De nuevo el *backoff* es análogo a la *Contention Window*, ahora como el tamaño de la CW varía constantemente, el *backoff* ya no tomará la mayor parte de sus valores entre 0ms y 31ms, sino que sus valores estarán más repartidos entre 0ms y 1023ms.

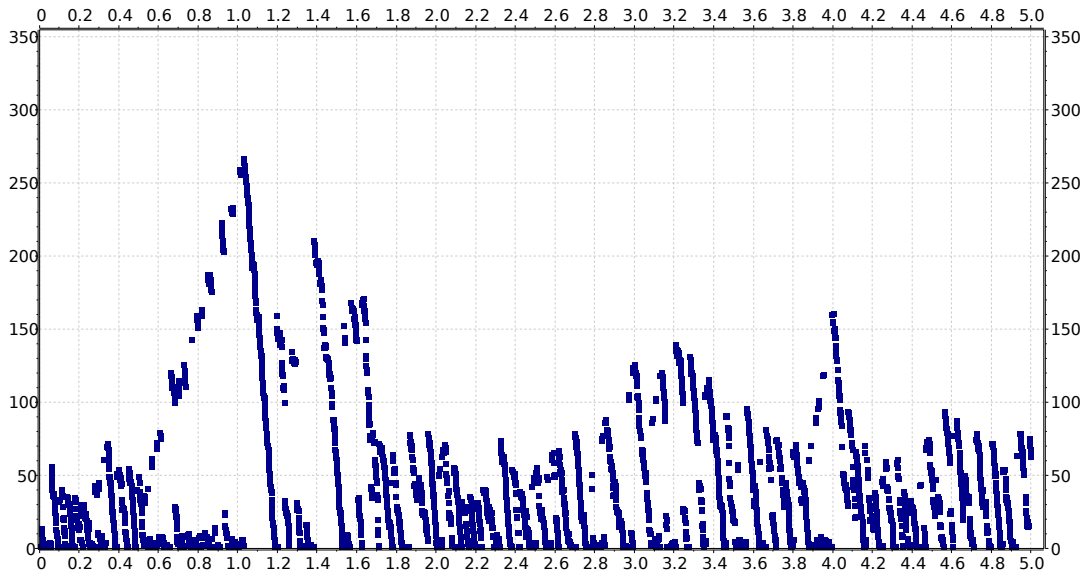

Figura 2.10: Retardo extremo a extremo. Tiempo que tarda un paquete en llegar al nodo *receiver* (ms) vs Tiempo (s)

Proporcionalmente al *backoff* y la CW, el retardo también aumenta muchísimo, antes no llegaba a superar los 8ms y ahora incluso llega a superar los 250ms. Podemos apreciar un pico bastante grande sobre t=1ms, que si nos fijamos coincide con un periodo de tiempo en el que la CW se mantiene en 1023ms (Figura 2.8), es decir, muchos intentos de retransmisión consecutivos sin éxito, además de unos *backoffs* muy grandes, lo que provoca que el retardo crezca desmesuradamente.

2.4 Pregunta 4

Obtenga un diagrama de secuencia de una colisión ocurrida en el escenario 2. ¿Qué diferencia hay con respecto al escenario 1? ¿Por qué ocurren ahora las colisiones?

Se simula de nuevo hasta t=0.1s, aunque en este caso podríamos simular mucho menos, ya que se empiezan a producir colisiones desde que se inicia la simulación.

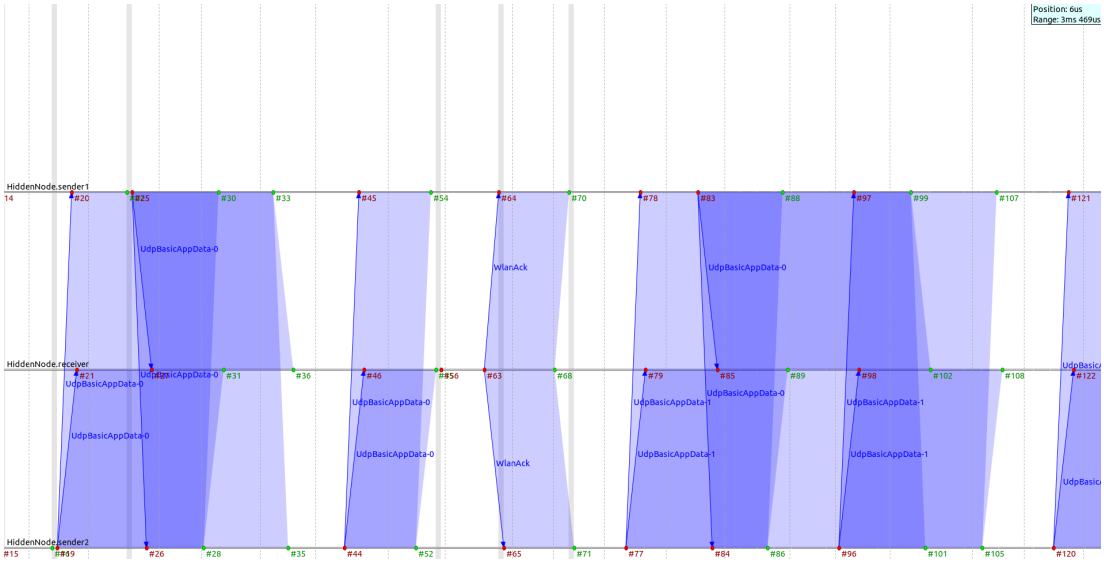

Figura 2.11: Diagrama de secuencia de la colisión producida en t=0.004s (aproximadamente)

Efectivamente, nada más comenzar la simulación ya se produce una colisión, ¿Qué está pasando en este caso? Ahora ***sender2*** envía un datagrama UDP primero, después, ***sender1***, aunque haya pasado el tiempo necesario para que reciba el datagrama UDP de ***sender2***, no lo recibe porque hay un obstáculo entre los dos nodos, entonces, envía su datagrama UDP igualmente, provocando una colisión.

En el diagrama de secuencia se ve como si cada nodo emisor recibiese los paquetes del otro nodo emisor, pero no se reciben realmente, por ejemplo, el evento #20 sería el instante de tiempo en el que ***sender1*** debería haber recibido el datagrama UDP de ***sender2***, sabiendo así que el medio está ocupado y esperando para poder transmitir. Sin embargo, no recibe ese datagrama, por eso transmite de todas formas un poco más tarde en el evento #25, provocando una colisión, esto estará ocurriendo constantemente durante toda la simulación.

2.5 Pregunta 5

Muestre y explique los siguientes valores para los dos escenarios anteriores:

- **Número de paquetes UDP que bajan de la capa de aplicación (packetSent) en cada nodo sender.**

HiddenNode.sender1.app[0]	packetSent:vector(packetBytes)	1691
HiddenNode.sender2.app[0]	packetSent:vector(packetBytes)	1664

Figura 2.12: Número de paquetes UDP que bajan de la capa de aplicación (escenario 1)

sender1: 1691 paquetes | *sender2*: 1664 paquetes

Paquetes UDP que bajan de la capa de aplicación (escenario 1) = 3355 paquetes

HiddenNode.sender1.app[0]	packetSent:vector(packetBytes)	1737
HiddenNode.sender2.app[0]	packetSent:vector(packetBytes)	1649

Figura 2.13: Número de paquetes UDP que bajan de la capa de aplicación (escenario 2)

sender1: 1737 paquetes | *sender2*: 1649 paquetes

Paquetes UDP que bajan de la capa de aplicación (escenario 2) = 3386 paquetes

- **Número de paquetes extraídos de cola MAC para ser transmitidos (packetPopped) en cada sender.**

HiddenNode.sender1.wlan[0].	packetPopped:vector(packetBytes)	1691
HiddenNode.sender2.wlan[0].	packetPopped:vector(packetBytes)	1664

Figura 2.14: Número de paquetes extraídos de cola MAC para ser transmitidos (escenario 1)

sender1: 1691 paquetes | *sender2*: 1664 paquetes

Paquetes extraídos de la cola MAC (escenario 1) = 3355 paquetes

HiddenNode.sender1.wlan[0].mac.dcf.channelAccess.pendingQueue	packetPopped:vector(packetBytes)	1728
HiddenNode.sender2.wlan[0].mac.dcf.channelAccess.pendingQueue	packetPopped:vector(packetBytes)	1630

Figura 2.15: Número de paquetes extraídos de cola MAC para ser transmitidos (escenario 2)

sender1: 1728 paquetes | *sender2*: 1630 paquetes

Paquetes extraídos de la cola MAC (escenario 2) = 3358 paquetes

- **Número de transmisiones de capa MAC (packetSentToLower) en cada sender.**

HiddenNode.sender1.wlan[0].	packetSentToLower:vector(packetBytes)	1711
HiddenNode.sender2.wlan[0].	packetSentToLower:vector(packetBytes)	1684

Figura 2.16: Número de transmisiones de capa MAC (escenario 1)

sender1: 1711 transmisiones | *sender2*: 1684 transmisiones

Transmisiones de capa MAC (escenario 1) = 3395 transmisiones

HiddenNode.sender2.wlan[0].mac	packetSentToLower:vector(packetBytes)	2464
HiddenNode.sender1.wlan[0].mac	packetSentToLower:vector(packetBytes)	2560

Figura 2.17: Número de transmisiones de capa MAC (escenario 2)

sender1: 2464 transmisiones | *sender2*: 2560 transmisiones

Transmisiones de capa MAC (escenario 2) = 5024 transmisiones

- **Número de retransmisiones de capa MAC (packetSentToLower – packetPopped) en cada sender.**

  Escenario 1

  sender1: 1711 - 1691 = 20 retransmisiones

  sender2: 1684 - 1664 = 20 retransmisiones

  Escenario 2

  sender1: 2464 - 1728 = 736 retransmisiones

  sender2: 2560 - 1630 = 930 retransmisiones

- **Número de paquetes UDP recibidos por receiver.**

HiddenNode.receiver.udp	packetReceived:vector(packetBytes)	3355

Figura 2.18: Número de paquetes UDP recibidos por *receiver* (escenario 2)

Paquetes recibidos (escenario 1) = 3355 paquetes

HiddenNode.receiver.udp	packetReceived:vector(packetBytes)	3293

Figura 2.19: Número de paquetes UDP recibidos por *receiver* (escenario 2)

Paquetes recibidos (escenario 2) = 3293 paquetes

En el escenario (1) se enviaron (a nivel de aplicación) 3355 paquetes, de los cuales se recibieron 3355 paquetes, es decir, no se perdió ningún paquete. Esto tiene lógica, porque si miramos el valor *packetDropIncorrectlyReceived* en el *receiver* es igual a 20, es decir, ocurrieron 20 colisiones a lo largo de toda la simulación, que coincide con el número de retransmisiones de cada nodo emisor, es decir, hubo 20 colisiones y esas 20 veces, los dos nodos emisores retransmitieron el paquete perdido correctamente, por tanto, no se pierde ningún paquete.

En el escenario (2) si que hay más valores que, a priori, pueden parecer extraños. En primer lugar, el número de paquetes UDP enviados es igual a 3386 paquetes, mientras que el número de paquetes extraídos de la cola MAC son 3358 paquetes, ¿Qué pasa con los otros 28 paquetes? Seguramente, estos paquetes se hayan dropeado en la cola MAC, al alcanzar el número máximo de retransmisiones fallidas *retryLimit* (por defecto 7), esto significaría que esos 28 paquetes se intentaron retransmitir 7 veces sin éxito, por tanto, se descartaron.

En segundo lugar, el número de retransmisiones de los dos nodos emisores no coincide, cuando en un escenario habitual debería ser el mismo, ¿Por qué? Como los nodos emisores no se escuchan entre ellos, puede ocurrir que tras una colisión, uno de los nodos retransmita y, cuando el segundo vaya a retransmitir también, el primer nodo ya esté transmitiendo un nuevo paquete, provocando una colisión de nuevo, haciendo que el segundo nodo tenga que retransmitir otra vez el mismo paquete.

En el escenario (2) se pierden 3386 - 3293 = 93 paquetes. Si observamos las retransmisiones ahora, como cada nodo retransmite un número de veces distinto, veamos las retransmisiones medias, (736+930)/2 = 833 retransmisiones de media. Por otro lado, observando el valor *packetDropIncorrectlyReceived* en el *receiver* vemos que es igual a 893, es decir, hubo 893 colisiones.

Si ahora observamos todos estos datos en conjunto, el número de paquetes perdidos en el escenario (2) tiene bastante lógica, durante la simulación hubo 893 colisiones, de las cuales se retransmitieron 833 de media, es decir, 60 paquetes perdidos (aproximadamente), si recordamos que antes dijimos que también se habían perdido 28 paquetes en las colas MAC de los nodos emisores, nos sale un total de 88 paquetes perdidos, que prácticamente coincide con los 93 paquetes perdidos calculados previamente (son números aproximados).

Por otro lado, comparando los dos escenarios, en el escenario (2) se pierden más paquetes que en el escenario (1) y el número de retransmisiones es mucho mayor (833 frente a 20 retransmisiones). Esto es lógico porque el número de colisiones en el escenario (2) es muy superior, debido al obstáculo que impide que los nodos emisores se escuchen mutuamente.

2.6 Pregunta 6

Simule el escenario 3 y muestre las mismas gráficas que en la cuestión 1. ¿Qué se observa ahora en comparación con el escenario 2?

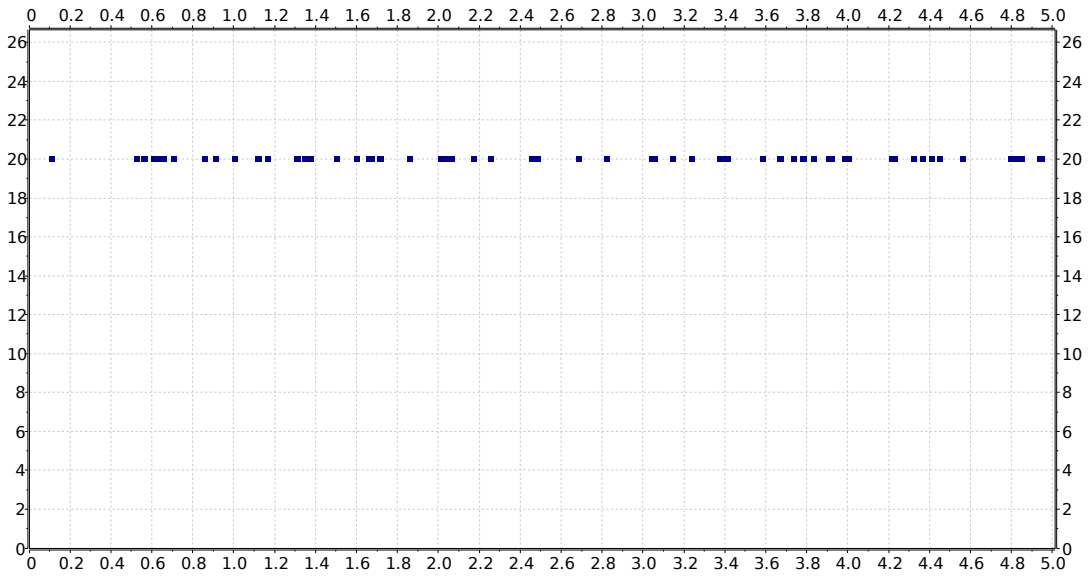

Figura 2.20: Paquetes recibidos incorrectamente por *receiver* (B) vs Tiempo (s)

Ahora el número de colisiones disminuye, sigue habiendo un obstáculo, pero el uso de RTS/CTS ayuda a solucionar el problema del *"Nodo oculto"* que provoca el obstáculo, aunque como podemos ver, no lo soluciona por completo ni mucho menos.

Si nos fijamos en los escenarios anteriores (Figuras 2.1 y 2.7) el tamaño de los paquetes que colisionaban era de 500B (paquetes UDP), ahora los paquetes que colisionan son de 20B (paquetes RTS), esto hace que el coste de retransmisión sea mucho menor. Nunca colisionará un paquete UDP porque siempre se envía un paquete RTS previamente para reservar el medio.

Como el número de colisiones es mucho menor, ahora la CW crece muchísimo menos que en el escenario (2), la mayor parte del tiempo varía entre 31ms y 62ms, alcanzando los 124ms en instantes de tiempo concretos, es decir, no se llegan a producir más de 2 colisiones consecutivas en toda la simulación, cuando en el escenario (2), el número de colisiones consecutivas llegó a superar las 7.

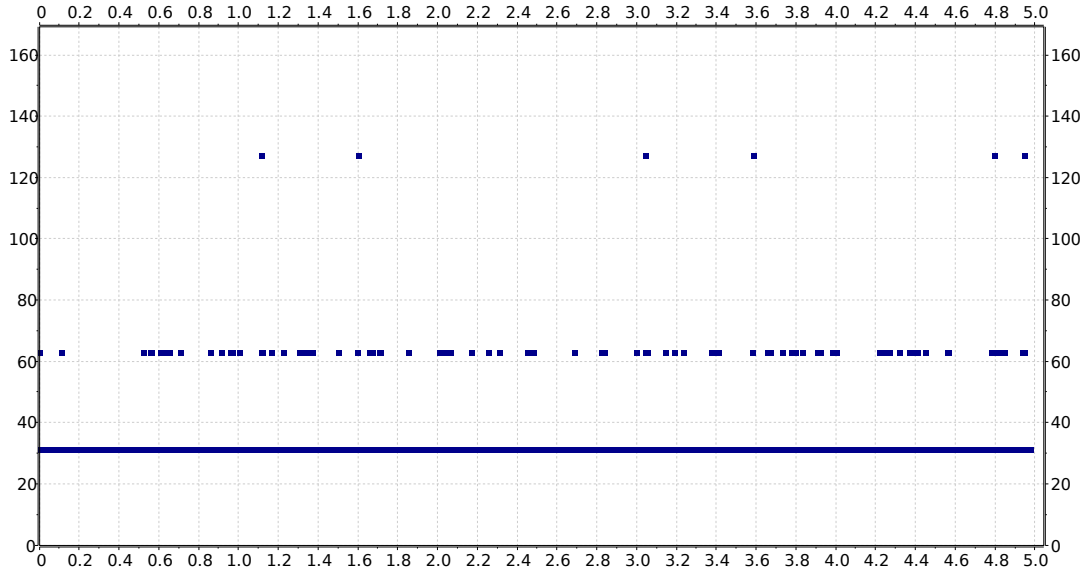

Figura 2.21: *Contention Window* del nodo *sender1* (ms) vs Tiempo (s)

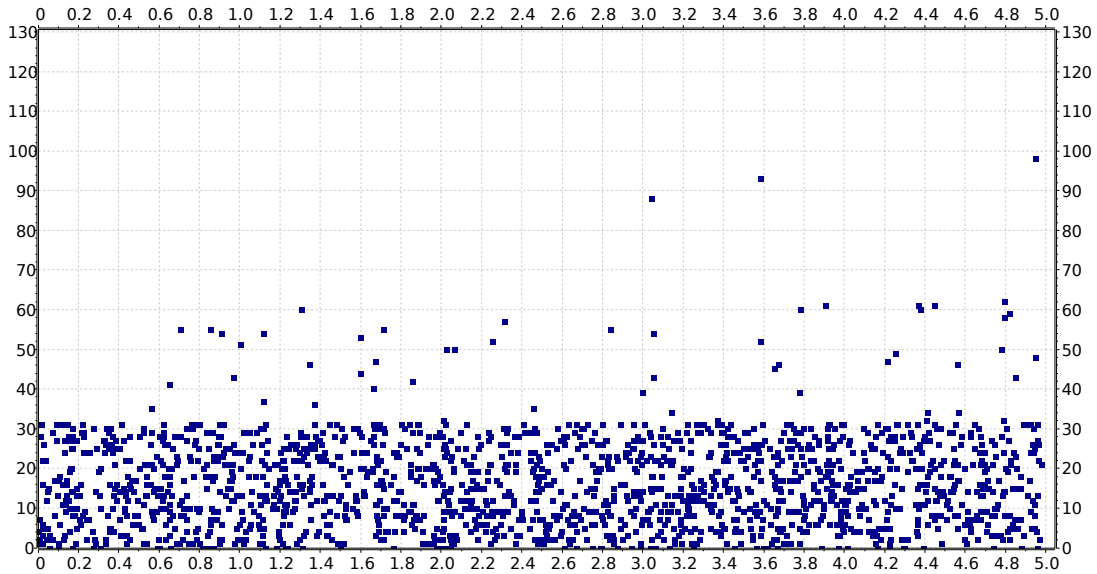

Figura 2.22: Backoff del nodo *sender1* (ms) vs Tiempo (s)

El *backoff*, como siempre, análogamente a la CW, varía la mayor parte del tiempo entre 0ms y 31ms, tomando también bastantes valores entre 31ms y 62ms y algún valor (aunque muy pocos) entre 62ms y 124ms.

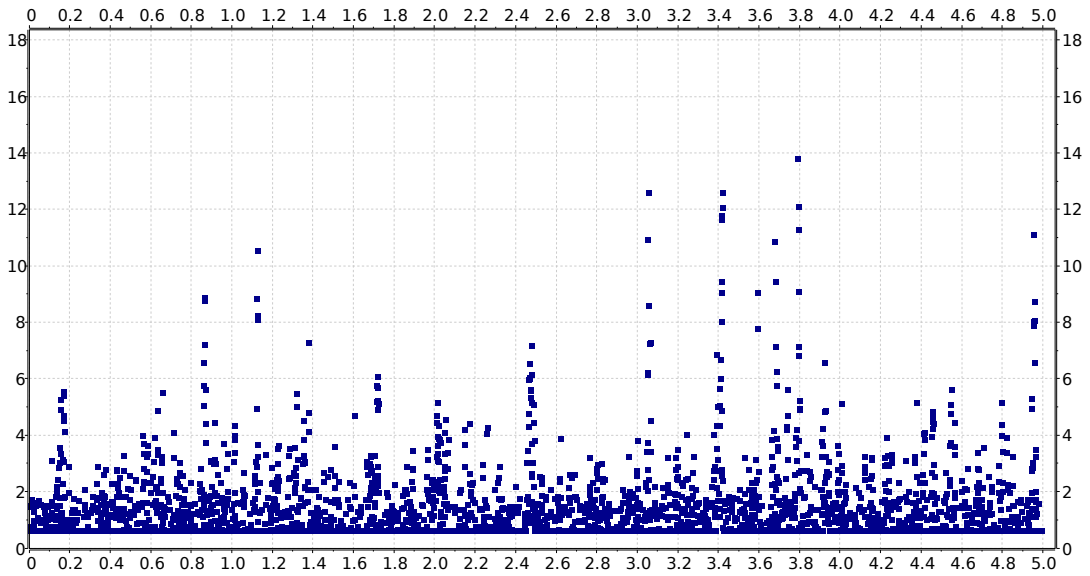

Figura 2.23: Retardo extremo a extremo. Tiempo que tarda un paquete en llegar al nodo *receiver* (ms) vs Tiempo (s)

A la par que la CW y el *backoff*, el retardo disminuye mucho, cuando en el escenario (2) llegaba a superar los 250ms, ahora no supera los 14ms en ningún momento, sigue siendo un retardo mayor que en el escenario (1), porque el uso de RTS/CTS no soluciona por completo el problema del *"nodo oculto"*, si lo solucionase, el retardo debería ser similar, porque estaríamos ante un escenario prácticamente igual (aunque no exactamente, veremos más sobre esto en la Sección 2.8 página 26).

2.7 Pregunta 7

Obtenga un diagrama de secuencia de una colisión en el escenario 3. ¿Cuándo ocurren ahora las colisiones? ¿Deberían ser más, menos o las mismas (en media) que en el escenario 2? ¿Por qué?

La primera colisión se produce en t=0.110598388609s, por tanto, simulamos hasta t=0.12s (aproximadamente).

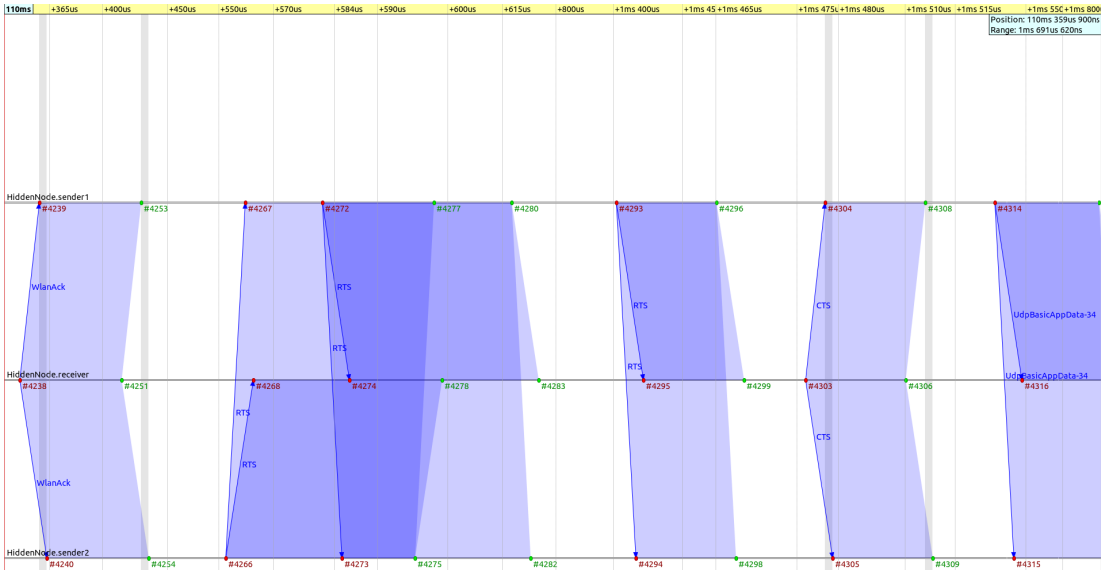

Figura 2.24: Diagrama de secuencia de la colisión producida en t=0.110598388609s

Ahora las colisiones ocurren cuando los dos nodos emisores envían un paquete RTS lo suficientemente rápido como para que no de tiempo a que el nodo ***receiver*** envíe un paquete CTS en medio, respondiendo al primero que haya enviado su paquete RTS. En este caso, ***sender2***, envía un paquete RTS para reservar el medio, ***sender1*** no escucha a ***sender2***, entonces piensa que el medio está libre porque el nodo ***receiver*** aún no ha respondido al paquete RTS de ***sender2*** con un paquete CTS, sería a partir de ese momento cuando ***sender1*** sabría que el medio está ocupado, pero como transmite su paquete RTS antes de que esto ocurra, el nodo ***receiver*** recibe dos paquetes RTS a la vez y se produce una colisión.

Respecto al escenario (2), ahora las colisiones deberían ser **menos** de media, ¿Por qué? En el escenario (2) el **tiempo potencial de colisión**, es decir, el tiempo durante el que puede ocurrir una colisión, era el tiempo transcurrido desde que un nodo emisor ocupa el medio hasta que lo deja libre, porque el otro nodo emisor no sabría en ningún momento si el medio está libre u ocupado, debido al obstáculo. Ahora, el **tiempo potencial de colisión**, es el tiempo que pasa desde que un nodo emisor (el que reserva el medio) envía su paquete RTS hasta que el otro nodo emisor recibe el paquete CTS de *receiver* indicando que el medio está ocupado. El **tiempo potencial de colisión** es infinitamente más grande en el escenario (2), por tanto, debería haber menos colisiones de media en el escenario (3), y se cumple, como hemos visto previamente en las gráficas (Figura 2.20).

2.8 Pregunta 8

Simule el escenario 4 y compare los resultados con los del escenario 1. ¿Se obtiene algún beneficio al usar RTS/CTS en un escenario sin nodos ocultos? ¿Cuál es el retardo medio en comparación con el escenario 1?

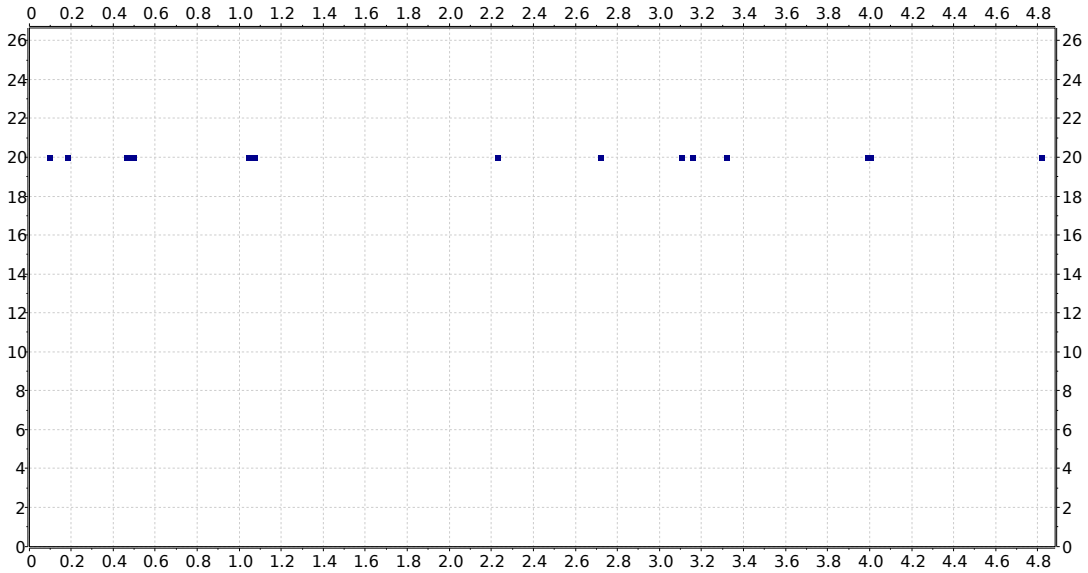

Figura 2.25: Paquetes recibidos incorrectamente por *receiver* (B) vs Tiempo (s)

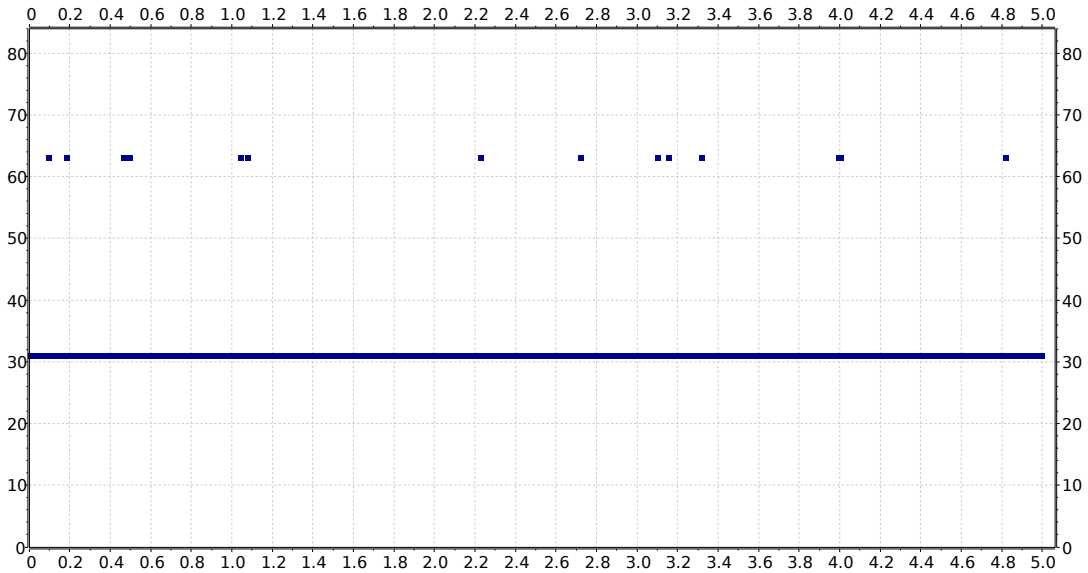

Figura 2.26: *Contention Window* del nodo *sender1* (ms) vs Tiempo (s)

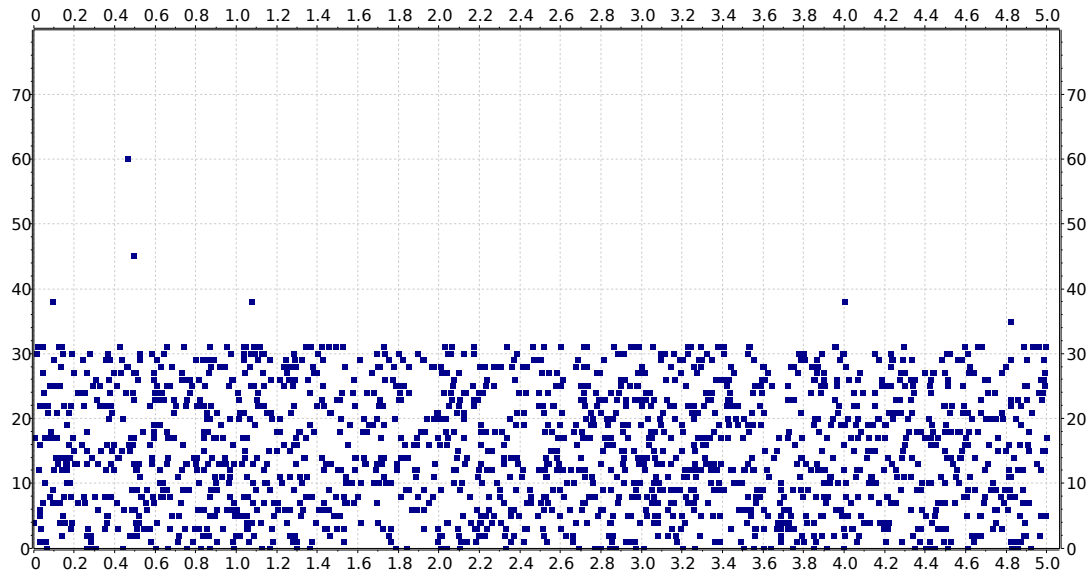

Figura 2.27: Backoff del nodo *sender1* (ms) vs Tiempo (s)

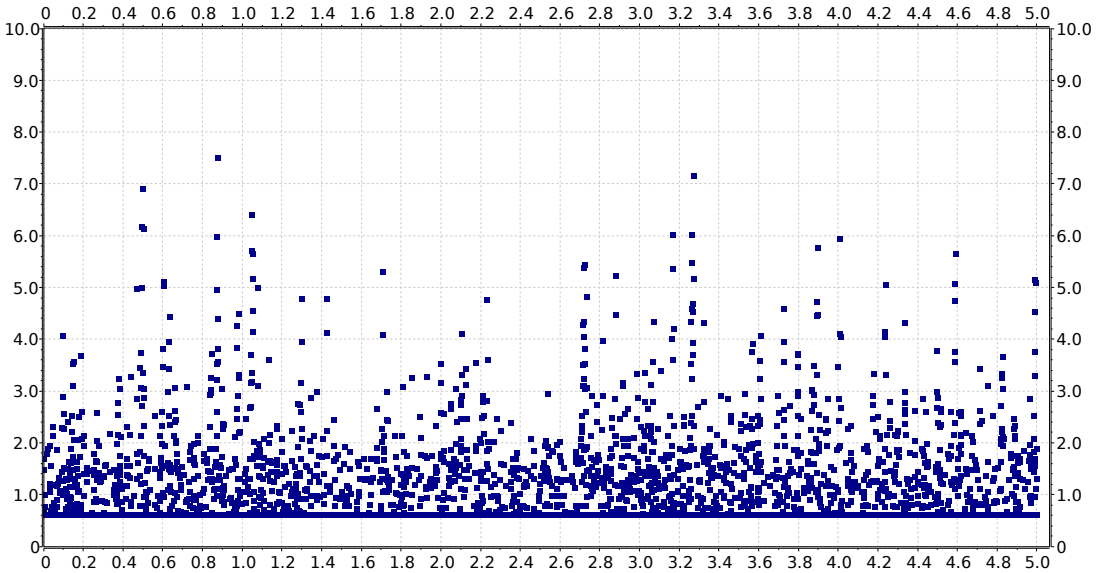

Figura 2.28: Retardo extremo a extremo. Tiempo que tarda un paquete en llegar al nodo *receiver* (ms) vs Tiempo (s)

Las gráficas son excesivamente similares, la única diferencia apreciable, es que en el escenario (4) la CW nunca alcanza una duración de 124ms, es decir, no se produce más de una colisión consecutiva nunca, mientras que en el escenario (1) si, pero esto es irrelevante, en escenarios como estos, en los que no hay un nodo oculto, que se produzcan dos colisiones consecutivas depende en gran parte del *backoff* y este es un número aleatorio.

Los resultados parecen casi idénticos, un poco mejores en el escenario (4), aunque muy poco. Viendo esto, podriamos decir que sí merece la pena usar RTS/CTS o que da lo mismo, pero antes veamos el retardo medio en cada escenario.

HiddenNode.receiver.app[0]	endToEndDelay:vector	9.801946296101342E-4

Figura 2.29: Retardo extremo a extremo medio en el escenario (1)

HiddenNode.receiver.app[0]	endToEndDelay:vector	0.0012078019376425993

Figura 2.30: Retardo extremo a extremo medio en el escenario (4)

Retardo medio escenario (1) = 0.0009801946296101342s

Retardo medio escenario (4) = 0.0012078019376425993s

En el escenario (4) hay 0.22ms más de retardo medio, a priori, puede parecer muy poco, pero lo importante es la razón de que se produzca este retardo, ya que hay más retardo a pesar de haber menos colisiones (*packetDropIncorrectlyReceived* = 16).

El retardo del escenario (4) es provocado por el envío de paquetes RTS/CTS, cada vez que queremos enviar un paquete UDP hay que enviar un paquete RTS también, esto en una simulación de 5 segundos provoca una diferencia mínima, pero en un escenario real sin nodos ocultos **no merece la pena usar RTS/CTS**, los resultados en cuanto al número de colisiones y los *backoffs* será similar, sin embargo, el retardo aumentará.

2.9 Pregunta 9

Obtenga un diagrama de secuencia de una colisión en el escenario 4. ¿Cuándo ocurren las colisiones? ¿Deberían ser más, menos o las mismas (en media) que en el escenario 1? ¿Por qué?

Sobre t=0.1s se produce una colisión, por tanto, simulamos hasta t=0.2s (aproximadamente).

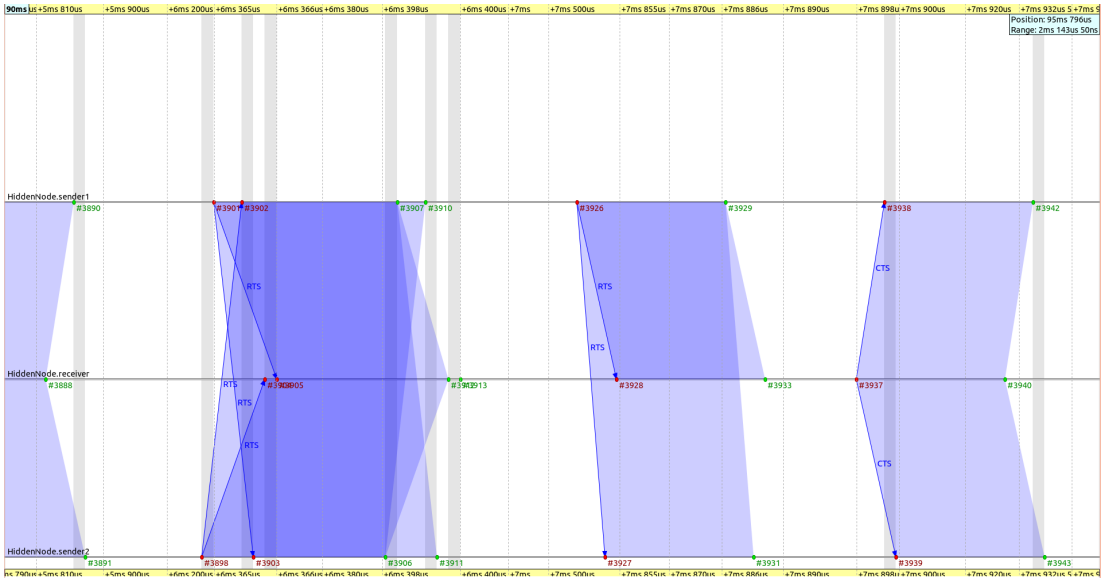

Figura 2.31: Diagrama de secuencia de la colisión producida sobre t=0.1s (aproximadamente)

Ahora las colisiones ocurren cuando los dos nodos emisores envían un paquete RTS lo suficientemente rápido como para que no de tiempo a que uno de los emisores, reciba el paquete RTS del otro antes de enviar el suyo. En este caso, ***sender2***, envía un paquete RTS para reservar el medio (Evento #3898), después, ***sender1***, antes de recibir el paquete RTS de ***sender2*** (#Evento 3902), envía su paquete RTS porque piensa que el medio está libre (Evento #3901), por tanto, el nodo ***receiver*** recibe dos paquetes RTS a la vez y se produce una colisión.

Respecto al escenario (1), analizaremos de nuevo el **tiempo potencial de colisión**. En el escenario (1), el **tiempo potencial de colisión** es el tiempo transcurrido desde que un nodo emisor transmite un paquete UDP hasta que el otro lo recibe. Por otro lado, en el escenario (4), el **tiempo potencial de colisión** es el tiempo transcurrido desde que un nodo emisor envía un paquete RTS hasta que el otro lo recibe.

El tiempo que tarda en transmitirse un paquete UDP de un nodo a otro (500B) es superior al tiempo que tarda en transmitirse un paquete RTS de un nodo a otro (20B), por tanto, el escenario (4) debería tener **menos colisiones de media** que el escenario (1) y, efectivamente, se cumple, en el escenario (4) se producen 16 colisiones, frente a 20 colisiones en el escenario (1), la diferencia es mínima, pero existe.

2.10 Pregunta 10

Repita los escenarios 1, 2 y 3 con una velocidad de transmisión de 6 Mbps y obtenga el número de paquetes recibidos por receiver, el número de colisiones y el retardo extremo a extremo medio en cada uno. ¿Cómo afecta a cada escenario la disminución de la velocidad? ¿Por qué?

ACLARACIÓN: Se muestran las gráficas indicadas en el enunciado y las que sea necesario mostrar para una velocidad de tranmisión de 9 Mbps, ya que hay escenarios para los que no se midió alguno de estos valores con una velocidad de transmisión de 9 Mbps.

Antes de comenzar hay que diferenciar dos conceptos, la velocidad de transmisión y la velocidad de la capa de transporte. La velocidad de la capa de trasnporte (UDP) dependerá del *sendInterval* que le hayamos establecido a la *UdpBasicApp* de cada nodo emisor, lo que significa que en la capa de transporte los paquetes se seguirán pasando a las capas inferiores a la misma velocidad independientemente de la velocidad de transmisión, eso sí, si aumentamos la velocidad de transmisión los paquetes saldrán más rápido de la cola MAC y es menos probable que esta se llene y tenga que descartar paquetes.

Escenario 1

HiddenNode.receiver.udp	packetReceived:vector(packetBytes)	3370

Figura 2.32: Número de paquetes UDP recibidos por *receiver* (6 Mbps)

HiddenNode.receiver.wlan[0].mac	packetDropIncorrectlyReceived:vector(packetBytes)	22

Figura 2.33: Número de colisiones (6 Mbps)

HiddenNode.receiver.wlan[0].mac	packetDropIncorrectlyReceived:vector(packetBytes)	20

Figura 2.34: Número de colisiones (9 Mbps)

HiddenNode.receiver.app[0]	endToEndDelay:vector	0.0019601665849685755

Figura 2.35: Retardo extremo a extremo medio (6 Mbps)

Paquetes UDP recibidos por receiver (6 Mbps) = 3370 > 3355 = Paquetes UDP recibidos por receiver (9 Mbps - Figura 2.18)

Número de colisiones (6 Mbps) = 22 > 20 = Número de colisiones (9 Mbps - Figura 2.34)

Retardo extremo a extremo medio (6 Mbps) = 0.00196 > 0.00098s = Retardo extremo a extremo medio (9 Mbps - Figura 2.29)

Vemos que el número de paquetes recibidos aumenta, esto no cuadra con lo explicado previamente (Sección 2.10 página 30). ¿Qué está pasando? En este caso, para el escenario (1), 6 Mbps es una velocidad de transmisión suficiente para que la cola MAC no se llene, por tanto, el número de paquetes recibidos (y enviados) dependerá del *sendInterval*, que sigue siendo el mismo.

Los paquetes recibidos dependen de la velocidad de la capa de transporte, si a las capas inferiores de los nodos emisores se envían paquetes a la misma velocidad, mientras la cola MAC no descarte paquetes, la cantidad de paquetes enviados y, consecuentemente, recibidos, será similar.

Las colisiones y, consecuentemente, el retardo, aumentan, esto se debe a que, al transmitir a menor velocidad, el *tiempo potencial de colisión*, que en este caso es el tiempo que tarda en transmitirse un paquete UDP, aumentará.

Escenario 2

HiddenNode.receiver.udp	packetReceived:vector(packetBytes)	2955

Figura 2.36: Número de paquetes UDP recibidos por *receiver* (6 Mbps)

HiddenNode.receiver.wlan[0].mac	packetDropIncorrectlyReceived:vector(packetBytes)	754

Figura 2.37: Número de colisiones (6 Mbps)

HiddenNode.receiver.app[0]	endToEndDelay:vector	0.047198041567743695

Figura 2.40: Retardo extremo a extremo medio (9 Mbps)

HiddenNode.receiver.wlan[0].mac	packetDropIncorrectlyReceived:vector(packetBytes)	893

Figura 2.38: Número de colisiones (9 Mbps)

HiddenNode.receiver.app[0]	endToEndDelay:vector	0.16280747141015228

Figura 2.39: Retardo extremo a extremo medio (6 Mbps)

Paquetes UDP recibidos por receiver (6 Mbps) = 2955 < 3293 = Paquetes UDP recibidos por receiver (9 Mbps - Figura 2.19)

Número de colisiones (6 Mbps) = 754 < 893 = Número de colisiones (9 Mbps - Figura 2.38)

Retardo extremo a extremo medio (6 Mbps) = 0.16280 > 0.04719s = Retardo extremo a extremo medio (9 Mbps - Figura 2.40)

En el escenario (2) el número de paquetes recibidos disminuye, este es el comportamiento ”esperado”, por lo explicado al principio (Sección 2.10 página 30), al reducir la velocidad de transmisión, las colas MAC de los nodos emisores se llenan y tienen que comenzar a descartar paquetes.

El número de colisiones disminuye, lo normal sería que aumente, pero en este caso, simplemente como se transmiten menos paquetes, se producen menos colisiones.

Escenario 3

HiddenNode.receiver.udp	packetReceived:vector(packetBytes)	3284

Figura 2.41: Número de paquetes UDP recibidos por *receiver* (6 Mbps)

HiddenNode.receiver.udp	packetReceived:vector(packetBytes)	3298

Figura 2.42: Número de paquetes UDP recibidos por *receiver* (9 Mbps)

Paquetes UDP recibidos por receiver (6 Mbps) = 3284 < 3298 = Paquetes UDP recibidos por receiver (9 Mbps - Figura 2.42)

HiddenNode.receiver.wlan[0].mac	packetDropIncorrectlyReceived:vector(packetBytes)	137

Figura 2.43: Número de colisiones (6 Mbps)

HiddenNode.receiver.wlan[0].mac	packetDropIncorrectlyReceived:vector(packetBytes)	77

Figura 2.44: Número de colisiones (9 Mbps)

HiddenNode.receiver.app[0]	endToEndDelay:vector	0.003286848435986602

Figura 2.45: Retardo extremo a extremo medio (6 Mbps)

HiddenNode.receiver.app[0]	endToEndDelay:vector	0.0013845223015054883

Figura 2.46: Retardo extremo a extremo medio (9 Mbps)

Número de colisiones (6 Mbps) = 137 > 77 = Número de colisiones (9 Mbps - Figura 2.44)

Retardo extremo a extremo medio (6 Mbps) = 0.00328 > 0.00138 = Retardo extremo a extremo medio (9 Mbps - Figura 2.46)

En el escenario (3), el número de paquetes recibidos es similar, mismo caso que en el escenario (1), si 6 Mbps es una velocidad de tranmisión suficiente para que las colas MAC no se saturen en este escenario, el número de paquetes recibidos será similar a los recibidos con una velocidad de 9 Mbps.

El número de colisiones aumenta, este es el comportamiento lógico, como se mencionó previamente, si reducimos la velocidad de transmisión, el **tiempo potencial de colisión** aumentará, que en este caso es el tiempo que pasa desde que un nodo emisor transmite un paquete RTS, hasta que el otro recibe el paquete CTS que envía el nodo receptor indicando que está ocupado. Al aumentar el **tiempo potencial de colisión**, aumentará el número de colisiones y, consecuentemente, el retardo medio.

2.11 Pregunta 11

Haga lo mismo con una velocidad de 18 Mbps. ¿Afecta a algún escenario el aumento de velocidad? ¿Por qué?

HiddenNode.receiver.udp	packetReceived:vector(packetBytes)	3312
HiddenNode.receiver.app[0]	packetReceived:vector(packetBytes)	3312

Figura 2.47: Número de paquetes UDP recibidos por *receiver* escenario (1)

HiddenNode.receiver.wlan[0].mac	packetDropIncorrectlyReceived:vector(packetBytes)	3

Figura 2.48: Número de colisiones escenario (1)

HiddenNode.receiver.app[0]	endToEndDelay:vector	4.5780344971316423E-4

Figura 2.49: Retardo extremo a extremo medio escenario (1)

HiddenNode.receiver.app[0]	packetReceived:vector(packetBytes)	3334
HiddenNode.receiver.udp	packetReceived:vector(packetBytes)	3334

Figura 2.50: Número de paquetes UDP recibidos por *receiver* escenario (2)

HiddenNode.receiver.wlan[0].mac	packetDropIncorrectlyReceived:vector(packetBytes)	732

Figura 2.51: Número de colisiones escenario (2)

HiddenNode.receiver.app[0]	endToEndDelay:vector	0.002538299075029634

Figura 2.52: Retardo extremo a extremo medio escenario (2)

HiddenNode.receiver.app[0]	packetReceived:vector(packetBytes)	3291
HiddenNode.receiver.udp	packetReceived:vector(packetBytes)	3291

Figura 2.53: Número de paquetes UDP recibidos por *receiver* escenario (3)

HiddenNode.receiver.wlan[0].mac	packetDropIncorrectlyReceived:vector(packetBytes)	56

Figura 2.54: Número de colisiones escenario (3)

HiddenNode.receiver.app[0]	endToEndDelay:vector	6.606229056937101E-4

Figura 2.55: Retardo extremo a extremo medio escenario (3)

El aumento de velocidad provoca un comportamiento similar en los 3 escenarios. Todos reciben un número de paquetes similar a los que recibían con 9 Mbps, es decir, el aumento de velocidad **NO afectó** a la cantidad de paquetes recibidos, esto se debe a lo mencionado en el ejercicio anterior para los escenarios (1) y (3) (Sección 2.10 página 31), lo que pasa es que ningún escenario necesita una velocidad de tranmisión mayor que 9 Mbps (las colas

MAC no se llenan ya con esa velocidad), entonces, aunque aumentemos la velocidad de transmisión, como la velocidad de la capa de transporte (UDP) es la misma, la cantidad de paquetes recibidos será similar.

Por otra parte, el número de colisiones y el retardo disminuyen en todos los escenarios, esto es lógico, la cantidad de paquetes enviados es similar y la velocidad de transmisión es el doble que antes, lo que quiere decir que se enviará una cantidad de paquetes similar, pero con un **tiempo potencial de colisión** reducido a la mitad (aproximadamente, dependiendo del escenario esto se cumple en mayor o menor medida), lo que hace mucho menos probable que se produzcan colisiones.