**네트워크 프로토콜 설계 및 실습 과제 3**

전기정보공학부 2016-10769 이권형

**1. 코드에 대한 설명**

sender와 receiver에서 source code는 다음과 같다.

#include "contiki.h"

#include "net/rime.h"

#include "random.h"

#include "dev/button-sensor.h"

#include "dev/leds.h"

#include "dev/light-sensor.h"

#include "dev/cc2420.h"

#include <stdio.h>

#include <string.h>

*struct* message {

*uint8\_t* seqno;

};

/\*---------------------------------------------------------------------------\*/

PROCESS(example\_broadcast\_process, "Broadcast example");

AUTOSTART\_PROCESSES(&example\_broadcast\_process);

/\*---------------------------------------------------------------------------\*/

static *void*

broadcast\_recv(*struct* broadcast\_conn \*c, const *rimeaddr\_t* \*from)

{

*struct* message msg;

memcpy(&msg, packetbuf\_dataptr(), sizeof(*struct* message));

printf("broadcast message received from %d.%d: seqno ( %u )\n", from->u8[0], from->u8[1], msg.seqno);

}

static const *struct* broadcast\_callbacks broadcast\_call = {broadcast\_recv};

static *struct* broadcast\_conn broadcast;

/\*---------------------------------------------------------------------------\*/

PROCESS\_THREAD(example\_broadcast\_process, ev, data)

{

static *struct* etimer et;

static *uint8\_t* seqno;

*struct* message msg;

PROCESS\_EXITHANDLER(broadcast\_close(&broadcast);)

PROCESS\_BEGIN();

broadcast\_open(&broadcast, 129, &broadcast\_call);

while(1) {

etimer\_set(&et, CLOCK\_SECOND \* 4);

PROCESS\_WAIT\_EVENT\_UNTIL(etimer\_expired(&et));

msg.seqno = seqno;

//packetbuf\_copyfrom("Hello", 6);

packetbuf\_copyfrom(&msg, sizeof(*struct* message));

broadcast\_send(&broadcast);

printf("broadcast message sent\n");

seqno++;

}

PROCESS\_END();

}

/\*---------------------------------------------------------------------------\*/

**코드1. sender의 source code**

sender의 경우 2주차에 사용하였던 broadcast sender를 사용하였다. broadcast\_open을 통해 channel 129에 대해 broadcast message를 보내도록 한다. 반복문 안에서, 4초동안 delay 이후 seqno에 해당하는 message를 send한다.

#include "contiki.h"

#include "net/rime.h"

#include "random.h"

#include "dev/button-sensor.h"

#include "dev/leds.h"

#include "dev/light-sensor.h"

#include "dev/cc2420.h"

#include <stdio.h>

#include <string.h>

*struct* message {

*uint8\_t* seqno;

};

/\*---------------------------------------------------------------------------\*/

PROCESS(example\_broadcast\_process, "Broadcast example");

AUTOSTART\_PROCESSES(&example\_broadcast\_process);

/\*---------------------------------------------------------------------------\*/

static *void*

broadcast\_recv(*struct* broadcast\_conn \*c, const *rimeaddr\_t* \*from)

{

*struct* message msg;

memcpy(&msg, packetbuf\_dataptr(), sizeof(*struct* message));

printf("broadcast message received from %d.%d: seqno ( %u )\n", from->u8[0], from->u8[1], msg.seqno);

}

static const *struct* broadcast\_callbacks broadcast\_call = {broadcast\_recv};

static *struct* broadcast\_conn broadcast;

/\*---------------------------------------------------------------------------\*/

PROCESS\_THREAD(example\_broadcast\_process, ev, data)

{

static *struct* etimer et;

static *uint8\_t* seqno;

*struct* message msg;

PROCESS\_EXITHANDLER(broadcast\_close(&broadcast);)

PROCESS\_BEGIN();

broadcast\_open(&broadcast, 129, &broadcast\_call);

while(1) {

/\* Delay 2-4 seconds \*/

etimer\_set(&et, CLOCK\_SECOND \* 4 + random\_rand() % (CLOCK\_SECOND \* 4));

PROCESS\_WAIT\_EVENT\_UNTIL(etimer\_expired(&et));

msg.seqno = seqno;

//packetbuf\_copyfrom("Hello", 6);

seqno++;

}

PROCESS\_END();

}

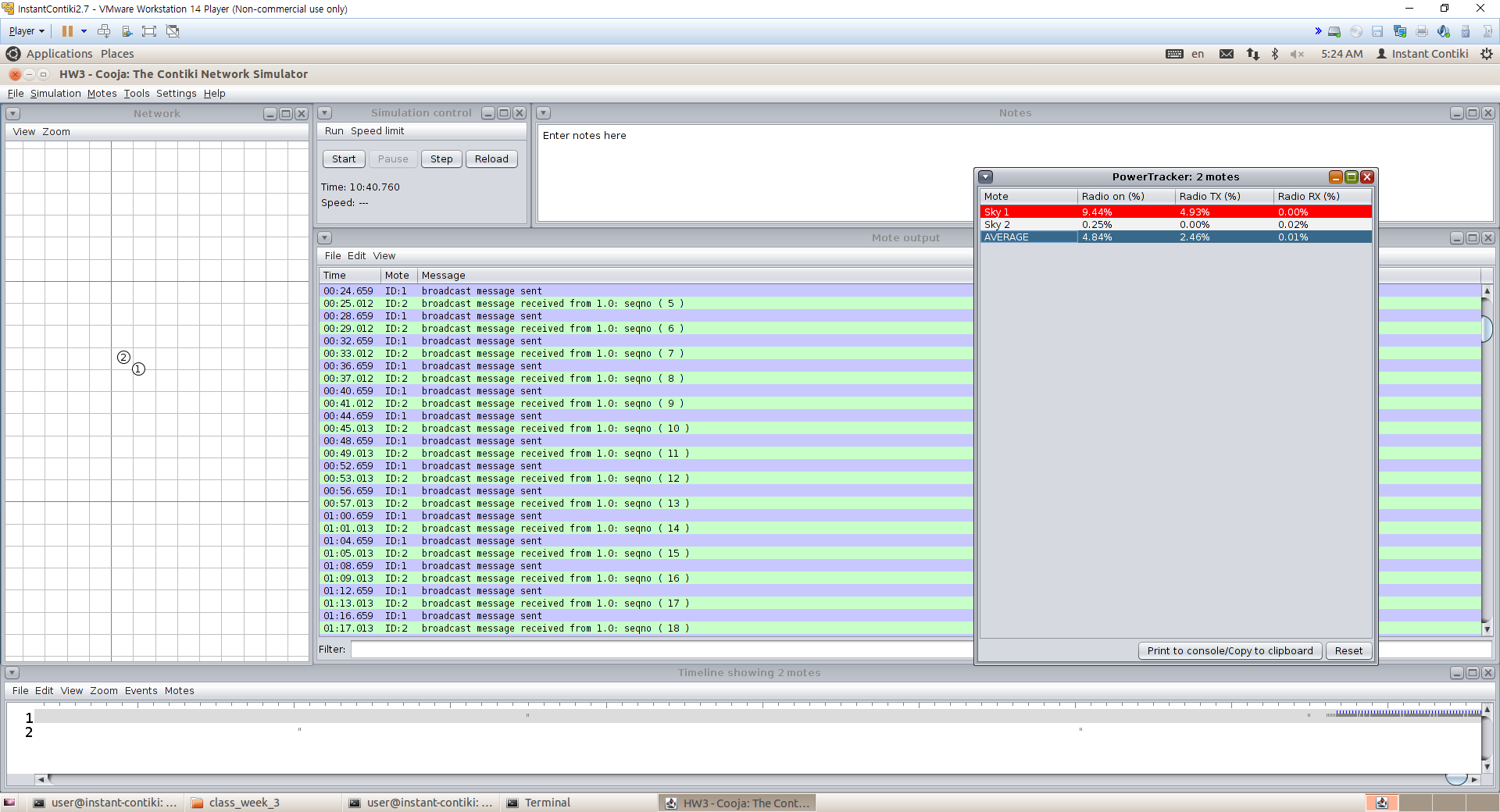
/\*---------------------------------------------------------------------------\*/

**코드2. receiver의 source code**

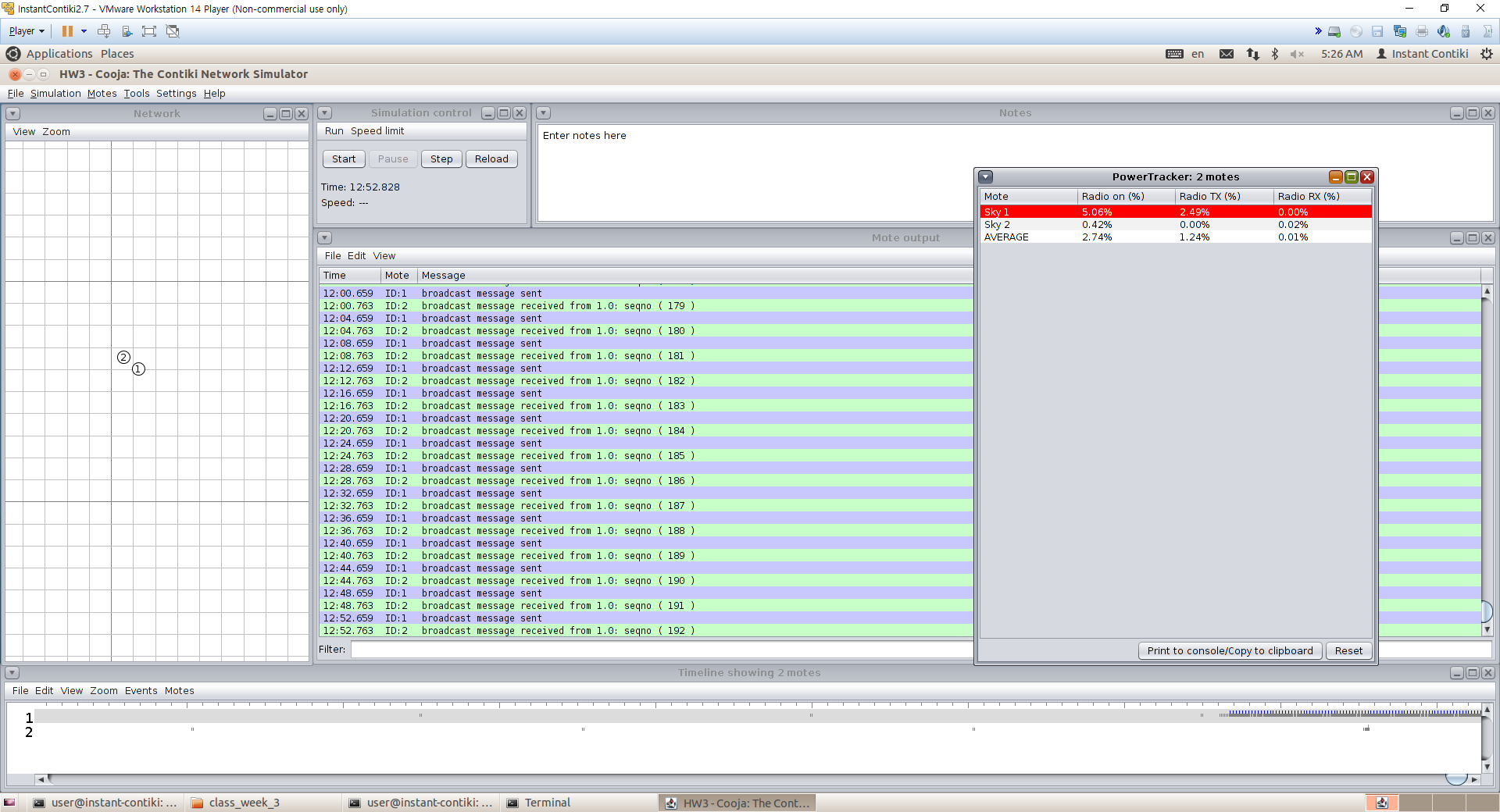
receiver의 경우 sender와는 다르게 message를 보내지 않고 받기만 한다. 따라서 while문 내에서 아무런 행동을 하지 않고, broadcast\_recv를 callback함수로 등록하여 message에 대해 seqno와 node의 number를 표시한다.

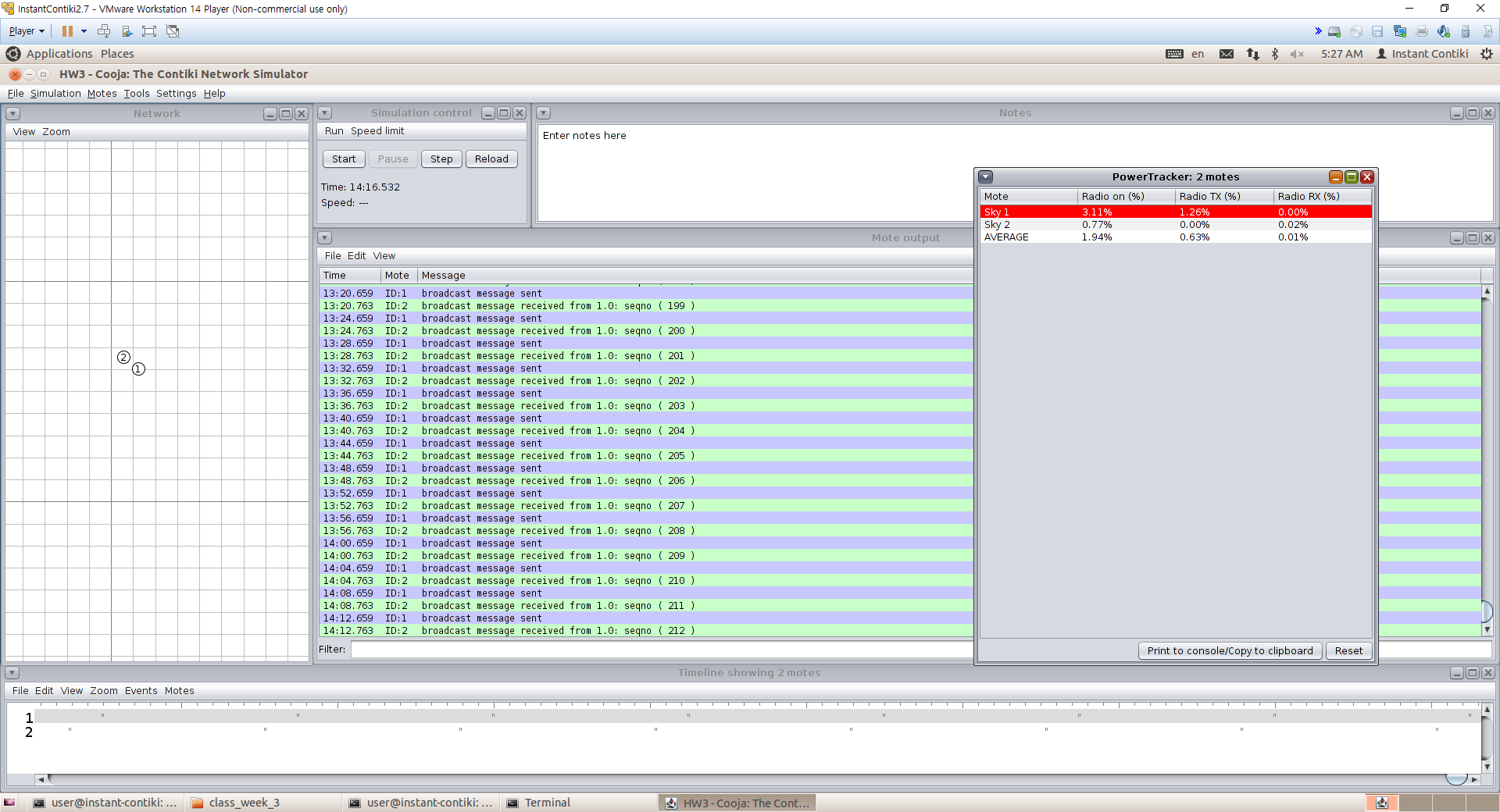
**2. 출력화면 캡처 및 그래프**

다음은 CHANNEL\_CHECK\_RATE에 따른 결과 출력을 나타낸 것이다. 2부터 시작하여 2의 지수로 늘려가며 측정하였다. 2, 4, 8, 16, 32, 64, 128, 256 총 8가지의 결과이다. RSSI나 LQI등 duty cycle과 power 이외의 요인은, node의 위치가 같기 때문에 항상 같을 것으로 생각하여 제외하였다. packet drop ratio의 경우 128일 때 100, 나머지의 경우 0으로 일정하였다.

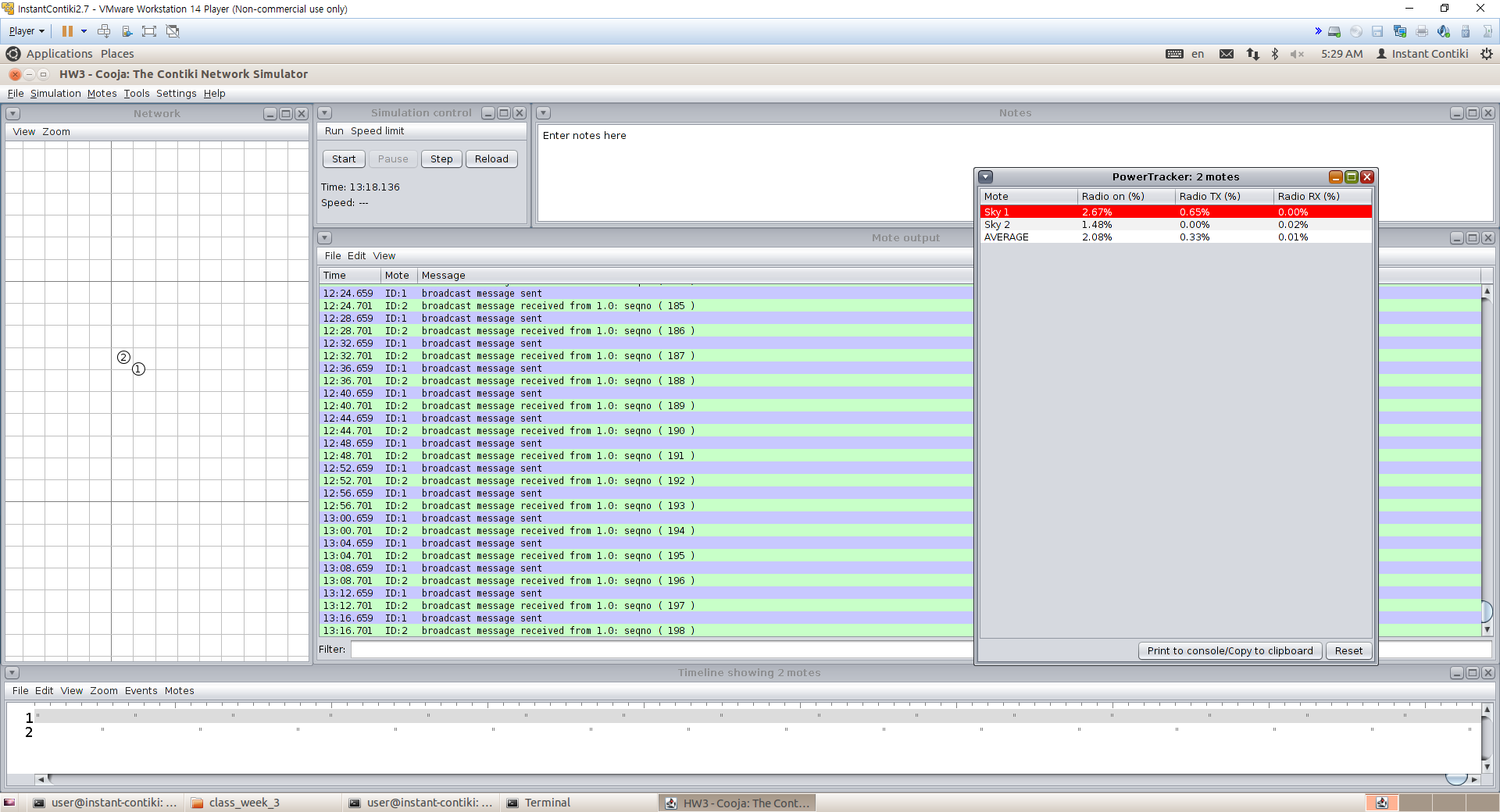


**그림1.** CHANNEL\_CHECK\_RATE 2Hz에 대한 결과

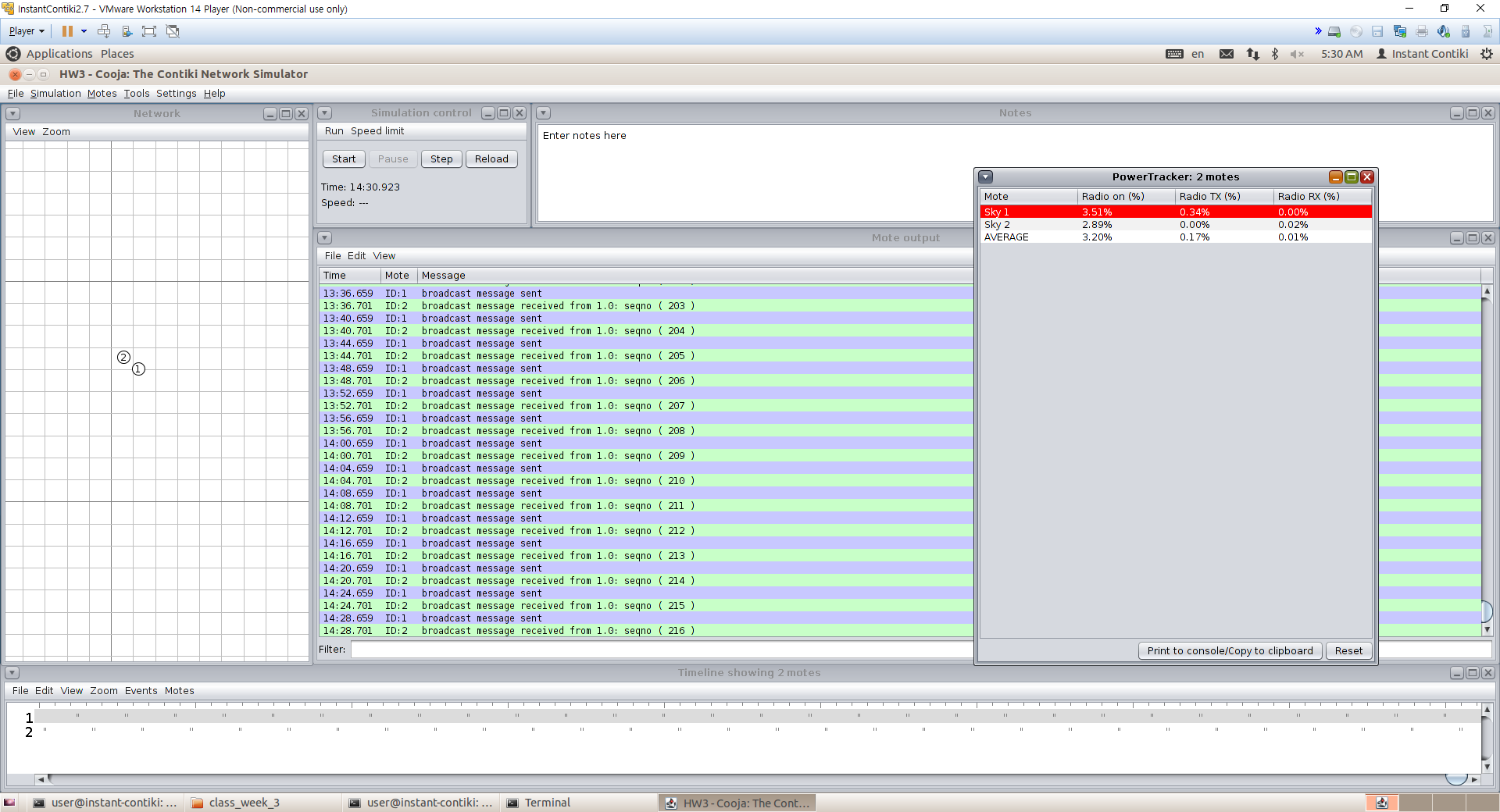


**그림2.** CHANNEL\_CHECK\_RATE 4Hz에 대한 결과 

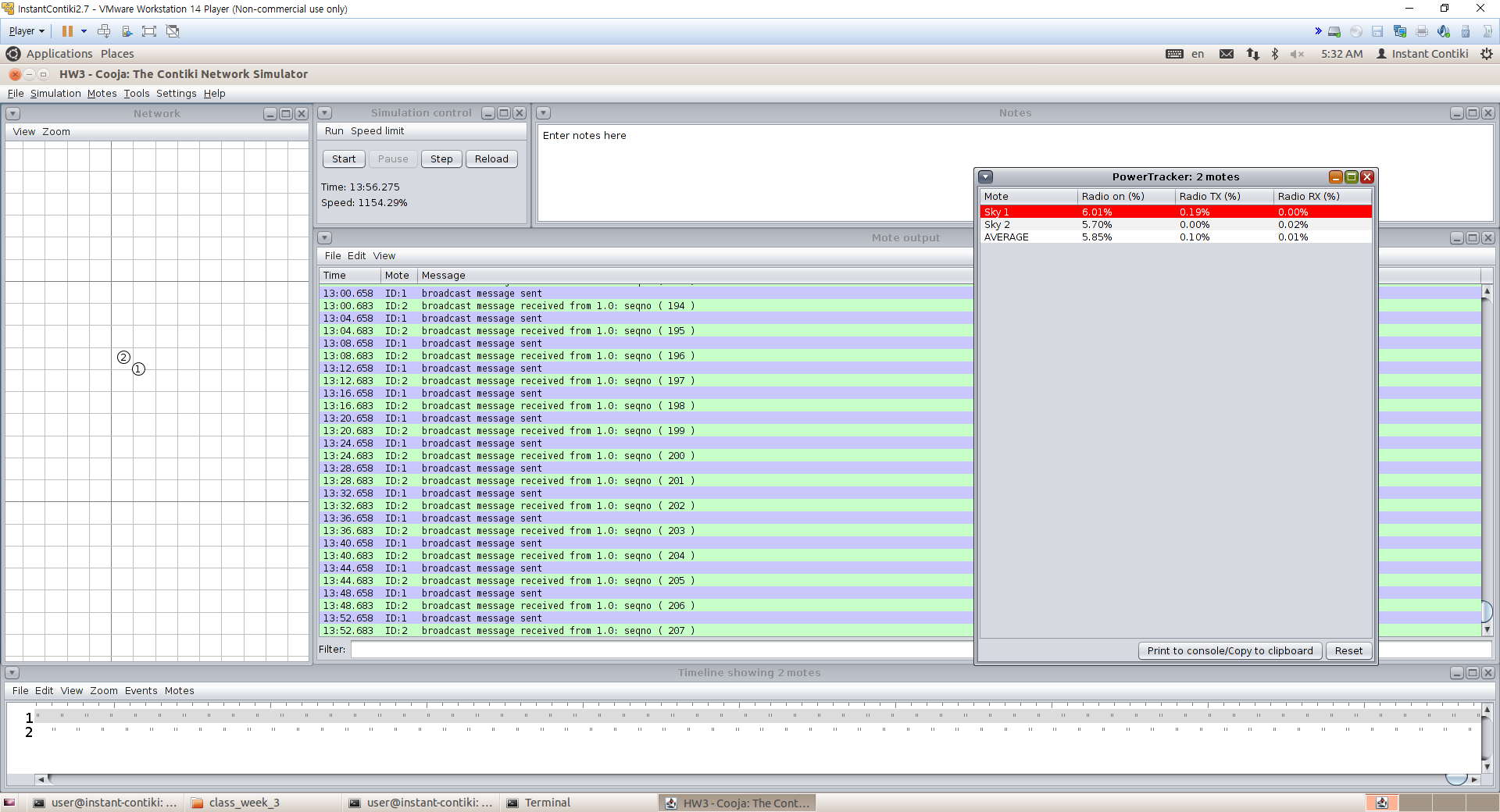
**그림3.** CHANNEL\_CHECK\_RATE 8Hz에 대한 결과

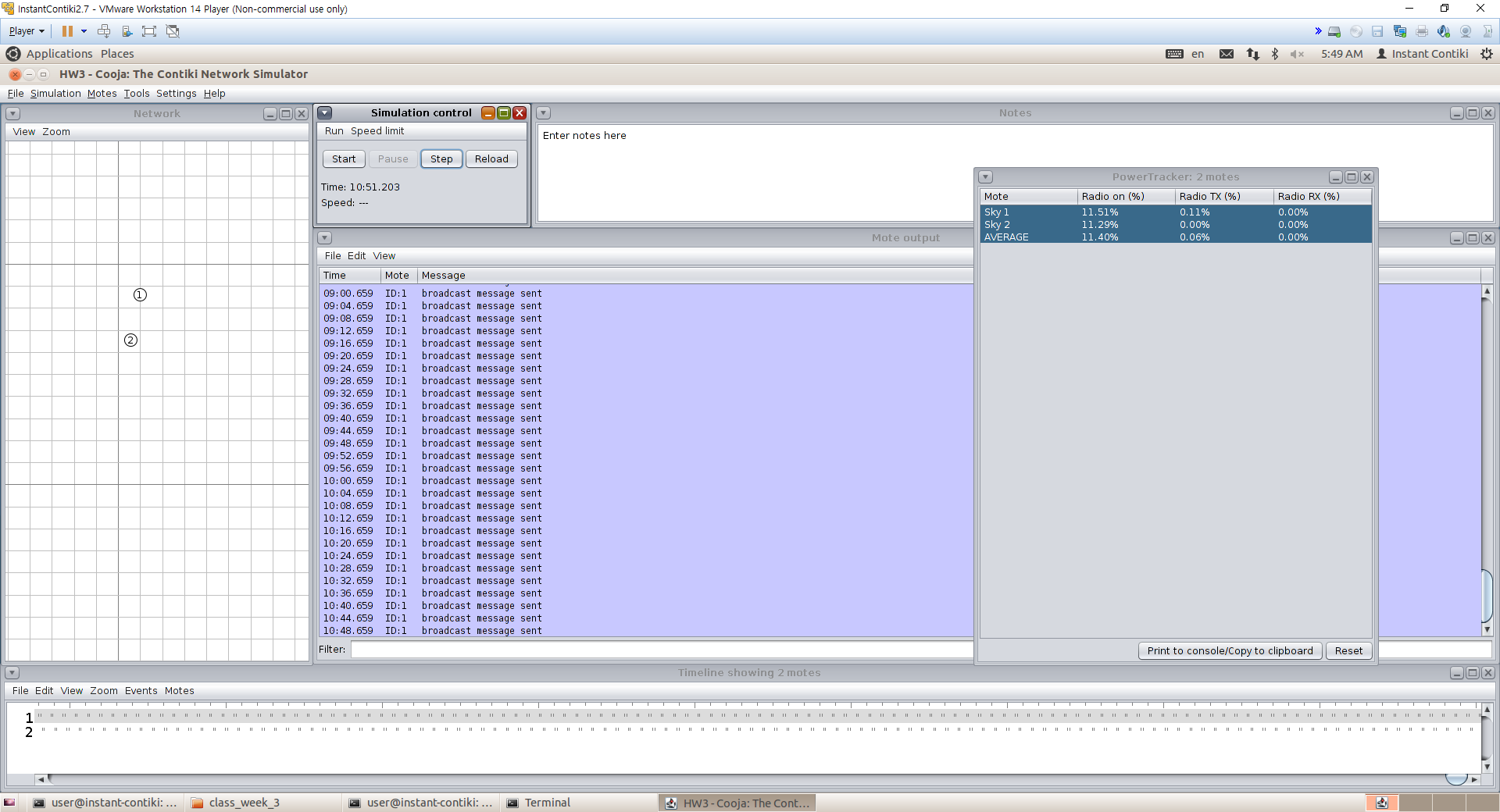


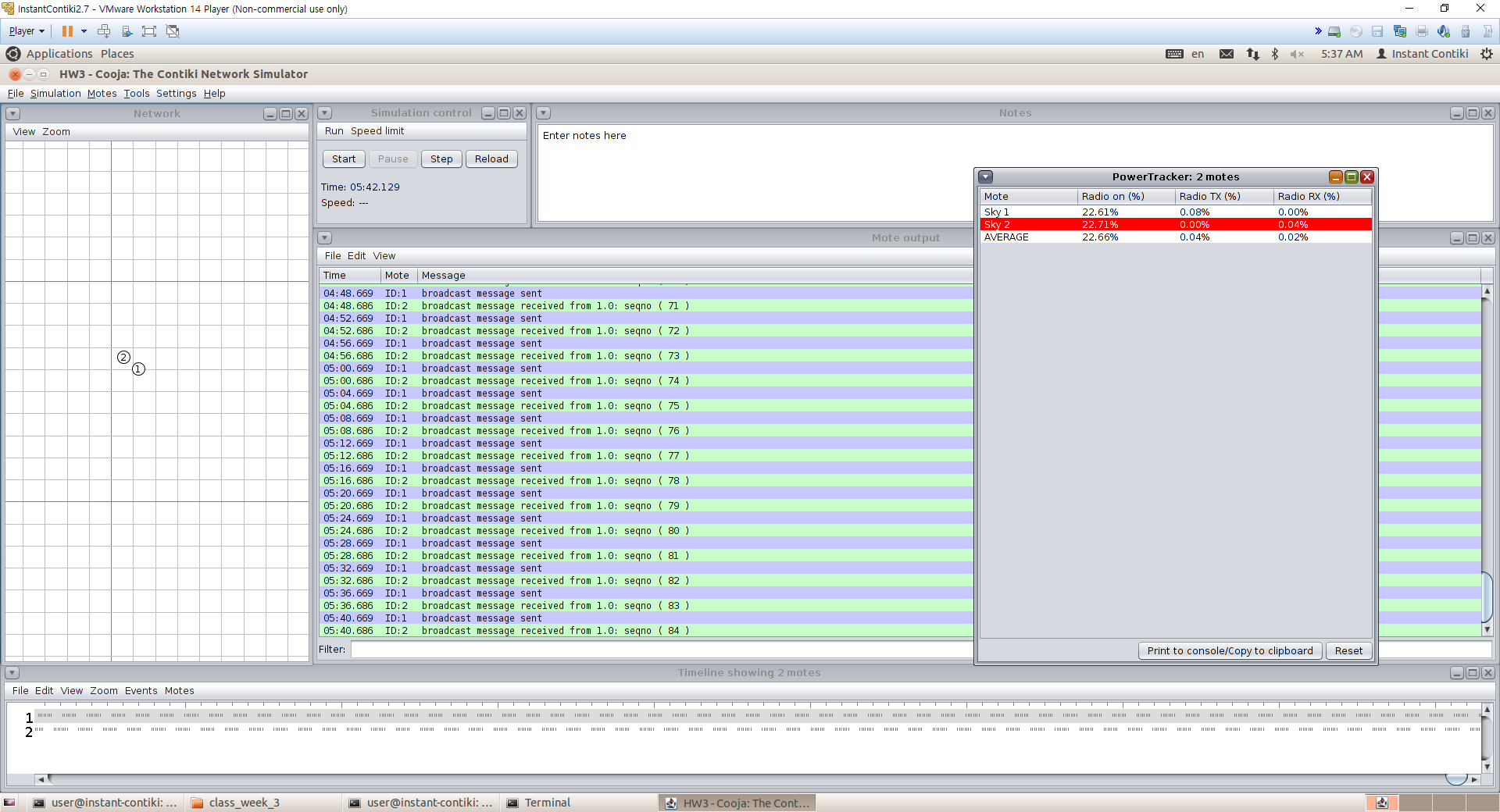
**그림4.** CHANNEL\_CHECK\_RATE 16Hz에 대한 결과



**그림5.** CHANNEL\_CHECK\_RATE 32Hz에 대한 결과

 **그림6.** CHANNEL\_CHECK\_RATE 64Hz에 대한 결과

 **그림7.** CHANNEL\_CHECK\_RATE 128Hz에 대한 결과

 **그림8.** CHANNEL\_CHECK\_RATE 256Hz에 대한 결과

또한 이에 나타난 Rx node와 Tx node에서의 duty cycle에 대한 graph는 다음과 같다. Rx node에서는 Tx를 측정하지 않고, Tx node에서는 Rx를 측정하지 않는다.

**그래프1.** TX node에서의 duty cycle. x축은 channel check rate(Hz)이고, y축은 duty cycle(%)이다.

**그래프2.** RX node에서의 duty cycle. x축은 channel check rate(Hz)이고, y축은 duty cycle(%)이다.

**3. 결과 해석**

그래프 1에서 볼 수 있듯, TX node에서 TX duty cycle은 1/x 개형을 그리는 것을 확인할 수 있다. 또한 radio on의 경우 초반에는 1/x 개형을 그리다가 이후 선형적으로 증가하는 것을 확인할 수 있다.

radio on은 크게 두 가지 성분으로 나눌 수 있는데, 이는 packet transmission에 사용되는 cycle과 CCA에 사용되는 cycle의 합으로 생각할 수 있다. 이 때, packet transmission에 사용되는 cycle이 TX duty cycle인데, 이는 strobe time에 비례한다. 그런데, strobe time은 cycle time + 2 \* check time이므로, check time은 channel check rate에 대한 역수이다. 따라서 TX duty cycle의 경우도 channel check rate에 대해 분수함수 개형을 나타내는 것이다. 두 번째로, CCA에 사용되는 cycle의 경우 channel check rate에 비례한다. 따라서 이에 대한 duty cycle은 channel check rate에 선형적으로 나타난다.

위의 결과를 토대로, TX node에서의 radio on duty cycle은 channel check rate x에 대해 a / x + b의 개형을 가진다(a와 b는 상수). 따라서 16Hz에서 최소가 되는 것을 확인할 수 있다.

그래프 2에서 확인할 수 있듯 RX node에서의 RX duty cycle은 일정하게 나타나고, radio on duty cycle은 선형으로 나타나는 것을 확인할 수 있다. RX duty cycle의 경우 packet을 받는 횟수는 일정하며, strobe time중 한번만 data를 받으면 성공하기 때문에 channel check rate에 관계 없이 일정하게 나타났다.

radio on duty cycle의 경우, TX node에서 transmission에 사용되는 duty cycle을 제외한 것과 같다. 그런데, 전송 부분은 1/x에 해당하므로 RX node에서는 선형적으로 나타나는 것이다.

결론적으로, duty cycle은 channel의 사용 비율을 나타내는 것이므로 이것에 비례해 power를 소모할 것이다. 따라서 모든 node에 대해 duty cycle을 합하여 가장 최소가 되는 점을 구할 수 있을 것이다. 다음은 그 그래프이다.

**그래프3.** 모든 node의 radio on duty cycle을 합한 그래프. x축은 channel check rate(Hz), y축은 duty cycle(%)를 나타낸다.

위의 그래프에서 확인할 수 있듯, 기본설정으로 되어 있는 channel check rate, 8Hz에 대해 duty cycle이 최소가 됨을 알 수 있다. 따라서 본 실험을 통해 가장 효율적으로 power를 사용할 수 있는 check rate를 구할 수 있었다.

**참고자료**

[2] Kurose, J. F. and K. W. Ross (2012). Computer Networking: A Top-Down Approach (6th Edition), Pearson.