Describe the behaviour of the Watts used as the Raspberry Pi boots.
The voltage stays constant even when the Raspberry Pi/switch is off, and then the current slowly increases from zero to a certain value as it turns on. As a result, the Watts also start from zero and slowly increases to about a constant 4.6 W when we are prompted with the login for the Raspberry Pi.

Does it reach a steady state?
Yes, as long as we continue to idle. At the time of writing, at the start of the exercise, we are not doing anything with the Raspberry Pi so it will reach the idle steady state value.
(It is interesting to note when we run the ps and ps aux commands on the Pi, the power consumption increases to 4.8-4.9 W momentarily before going back down to 4.6 W. Moreover, it takes about 2-3 seconds for this power consumption spike to be displayed on the tester, so whatever is on the tester's display is very tangibly delayed).

Network Activity

Task 1
captured.pcap
The frequency of messaging is about 2 events every 5 seconds -> 24 events/min. I did this by capturing 50 messages and writing the capture results to a pcap file. I saw that 2 DNS messages are sent every 5 seconds. There is also a set of 2 ARP requests sent every 40 seconds which I have not included in the above figure.

Task 2
Raspberry Pi to lab machine: Power consumption fluctuates between 4.91 to 5.14W. Observed over 20 seconds.

Lab machine to Raspberry Pi: Power consumption fluctuates between 4.93 to 5.22W. Observed over 20 seconds.

It is ever so slightly more expensive for the Raspberry Pi to receive than to send, but both are largely equal. This could be because the lab machine is larger and so the signals it sends are more powerful than the smaller Raspberry Pi.

Task 3
Before changing rx-usecs: Power consumption fluctuates between 5.57 to 6.26W. Estimated average to be 5.7W.
Rx-usecs (for ethtool -c eth0): 57

```
ubuntu@ubuntu:~/Desktop/CWM-ProgNets/assignment3$ iperf -c 192.168.10.2 -i 5 -t 20 -b 1G -u
------------------------------------------------------------
Client connecting to 192.168.10.2, UDP port 5001
Sending 1470 byte datagrams, IPG target: 10.95 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 192.168.10.1 port 55969 connected with 192.168.10.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-5.0000 sec   571 MBytes   957 Mbits/sec
[  1] 5.0000-10.0000 sec   570 MBytes   957 Mbits/sec
[  1] 10.0000-15.0000 sec   570 MBytes   957 Mbits/sec
[  1] 15.0000-20.0000 sec   570 MBytes   957 Mbits/sec
[  1] 0.0000-20.0005 sec  2.23 GBytes   957 Mbits/sec
[  1] Sent 1627653 datagrams
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  1] 0.0000-21.0113 sec  1.13 GBytes   462 Mbits/sec  63.124 ms 802741/1627653 (49%)
```

According to https://knowledge.informatica.com/s/article/80204?language=en_US and https://www.ibm.com/docs/en/linux-on-systems?topic=practices-performance-tuning, the rx-usecs parameter quantifies the interrupt coalesce. It is the number of microseconds to delay such interrupt after a packet arrives, and the reason an interrupt is performed is to make the CPU aware that the packet has arrived into the input queue (https://www.ibm.com/docs/en/aix/7.2.0?topic=options-interrupt-coalescing). In this example, a value of 57 means that interrupts will be done every 57 microseconds so one single interrupt can be generated for 57 microseconds' worth of packet arrivals (instead of a single interrupt for every single arriving packet). Interrupt coalesce is especially useful at higher data rates because it means the interrupt does not need to be done for every single packet that is arriving, so less resources have to be committed, but you do get more latency because of the delay.

Since this is an explicit delay, a higher value of rx-usecs will mean a higher jitter value is expected.

After changing rx-usecs: Power consumption fluctuates between 5.58 to 5.92W. Estimated average to be 5.6W.

```
ubuntu@ubuntu:~/Desktop/CWM-ProgNets/assignment3$ iperf -c 192.168.10.2 -i 5 -t 20 -b 1G -u
------------------------------------------------------------
Client connecting to 192.168.10.2, UDP port 5001
Sending 1470 byte datagrams, IPG target: 10.95 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 192.168.10.1 port 34990 connected with 192.168.10.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-5.0000 sec   570 MBytes   957 Mbits/sec
[  1] 5.0000-10.0000 sec   570 MBytes   957 Mbits/sec
[  1] 10.0000-15.0000 sec   570 MBytes   957 Mbits/sec
[  1] 15.0000-20.0000 sec   570 MBytes   957 Mbits/sec
[  1] 0.0000-20.0002 sec  2.23 GBytes   957 Mbits/sec
[  1] Sent 1627555 datagrams
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  1] 0.0000-20.0009 sec  1.09 GBytes   469 Mbits/sec   0.062 ms 829121/1627554 (51%)
```

Jitter is much lower. It decreased from 63.124 to 0.062 ms. However, I was not expecting such a large difference. This could be because as the latency is (theoretically) reduced by a blanket, 57 microseconds, the latency becomes very small in absolute terms. If the jitter is proportional to the latency, then it would make sense that the jitter also becomes very small in absolute terms.

CPU Activity

Power consumption fluctuates between 5.92 to 6.59W. Average estimated to be around 6.3W. The power consumption is not actually that much more than the idle 4.6W. This is only about a 30-40% increase from idle.

Using htop, we can see that interestingly, cores 2 and 3 are always at 100%, but cores 0 and 1 are usually only at 0-1%, and only go up to 100% during the stress test. The 4th lecture revealed that cores 0 and 1 have different purposes to cores 2 and 3.

The fact that two of the cores seem to always be at full load tells me that a 30-40% increase from idle is not unreasonable as even at idle, two of the cores are already engaged and the increase in power is just coming from the other two cores.

Theoretical Experiments

Country: Indonesia: CI = 640g $CO_{2\,eq}$/kWh = 0.64kg $CO_{2\,eq}$/kWh

Task 1

EC of the Raspberry Pi for this exercise is 17 Wh (correct to whole numbers) = 0.017 kWh (includes idling)

CF = CI x EC = 0.01088 kg $CO_{2\,eq}$ = 10.9 g $CO_{2\,eq}$ if done in Indonesia.

Task 2

If 30 billion Raspberry Pi's were connected, doing the same thing done in this exercise (including idling), and this was done in Indonesia:

a. Collective carbon footprint = 0.01088 x 10^9 kg $CO_{2\,eq}$= 326 400 000 kg $CO_{2\,eq}$ = 326.4 Gg $CO_{2\,eq}$.
b. Collective carbon footprint = 0.01088 x 10^9 kg $CO_{2\,eq}$= 326 400 000 kg $CO_{2\,eq}$ = 326.4 Gg $CO_{2\,eq}$. This is a very big number. I have personally never needed to use Gigagrams before.
c. The article https://ourworldindata.org/co2-emissions shows that in 2023, the worldwide total carbon emissions is 37.79 billion tonnes = 37.79 Pg $CO_{2\,eq}$ (petagrams!). If ICT emissions are 1.2% of the global greenhouse gas emissions, that it should be estimated that the ICT emissions is 453 Gg $CO_{2\,eq}$. Thus the estimate using Raspberry Pi is a slight (Raspberry Pi estimate is only 72.05% of the number derived from global total carbon emissions). This is probably because in the real world, there are machines much more complicated and much more power consuming than Raspberry Pis, especially consumer electronics. While consumer electronics are not that complex in the grand scheme of things, it is incredibly common in households today and thus form a large proportion of the Things connected to the internet.