# Test Cases & Results for ROS2 as middleware

- **Module Definition**
  - Data node
    - D1: Image node
      - PS: Publish the 1920*1080pixels / 60fps image stream **a**
      - Service: Start/Stop
    - D2: Image Process node
      - PS: Subscribe image stream **a**
      - PS: Publish Status String **b** / Result String **c**
      - Service: Start/Stop/Image Process Rate(control the resources of processing)
    - D3: Result Convert node
      - PS: Subscribe Result String **c**
      - OUTPUT: UDP package **d**
  - Log node
    - L1: Passive Log node
      - PS: Subscribe Status String **b**
      - PS: Publish log **e**
    - L2: Active Log node
      - Action: Push log **f** alternatively
    - L3: Log Convert node
      - PS: Subscribe log **e** / **f**
      - OUTPUT: UDP package **g**
  - Script
    - S1: A launch script for system
      - Launch D1->D2->D3
      - Launch L1, L2, L3
    - S2: A stop script for system

## Test cases for developers

### Stability and Robustness

- **Case 1: Single Long Time Run**
  - Image Process Rate to be set to 1.0
  - Launching S1, to run the system for **x** hours
  - Checking stuck/crashing status
- **Standard 1: x** should be larger than 24 without system crashing/stuck
- **Test Steps:**

  Run the following nodes in order.

  ①ros2 run py_pubsub image_source_video (D1py)

  ②ros2 run test_pkg D2cpp

  ③ros2 run test_pkg D3cpp

④ros2 run test_pkg L1
⑥ros2 run test_pkg L2
⑦ros2 run test_pkg L3
⑧ros2 run test_pkg talker / ros2 run test_pkg listener

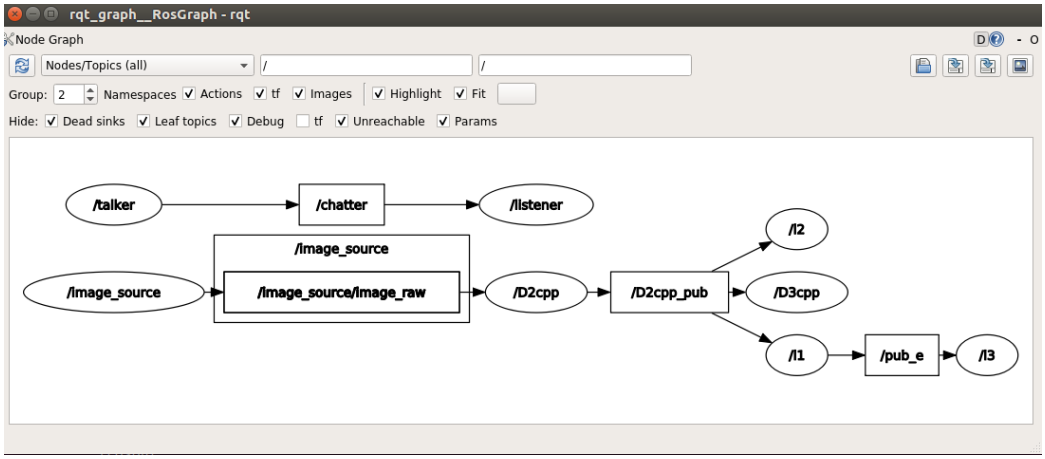


Figure 1. rqt_graph

- **Results:**

  Time of running: **93725 seconds**; Crashing Error: **0;** Stuck Error: **0**

  ①The node network is still communicating normally, and there is no crashed node and the talker and listener still work well;

  ②At **93890s**, through instruction `**ros2 topic echo "/D2cpp_pub" ("/pub_e /chatter …")**`, The data shows correctly and normally. Through the rviz2, we can see that the topic of "/image_source_raw" and the video stream still runs well.

- **Case 2: Loop Long Time Run**
  - Image Process Rate to be set to 1.0
  - Launching S1, to run the system for **x** hours
  - Launching S2, then launching S1, looping for **y** times
  - Checking stuck/crashing status
- **Standard 2: y** should be larger than 1000 without system crashing/stuck
- **Test Steps:**

  ① Use a shell script to automatically launch S1 (to start all nodes) and S2 (to stop all nodes). Command: `./test_s1_s2.sh
- **Results:**

  Cycles: **1200 times**; Crashing Error: **0**; Stuck Error: **0**

  ①The node network is still communicating normally, and there is no crashed node. The talker and listener still works well.

- **Case 3: Broken Run**
  - Image Process Rate to be set to 1.0
  - Launching S1, to run the system for **x1** hours
  - Killing Middleware service for **y** times
    - kill L1/L2, check output of string **c**

- kill D3, check L1/L2 log
- Checking stuck/crashing status and measuring this period as **x2** hours

- **Standard 3: x2** should be larger than 0.25 without system crashing/stuck
- **Test Steps:**

  Run the following nodes in order:

  ①ros2 run py_pubsub image_source_video (D1py)

  ②ros2 run test_pkg D2cpp

  ③ros2 run test_pkg D3cpp **(will be killed later...)**

  ④ros2 run test_pkg L1

  ⑥ros2 run test_pkg L2 **(will be killed later...)**

  ⑦ros2 run test_pkg L3

  ⑧ros2 run test_pkg talker / ros2 run test_pkg listener

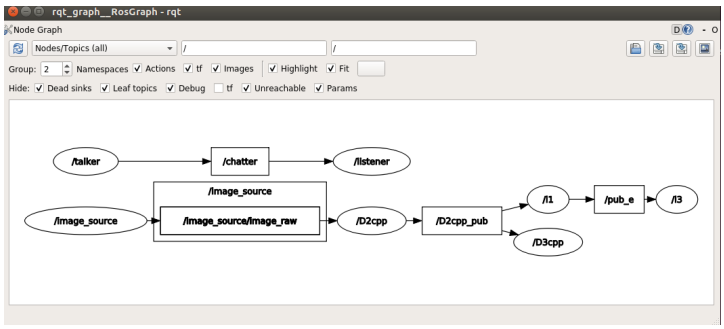


Figure 2. killing L2 node, rqt_graph

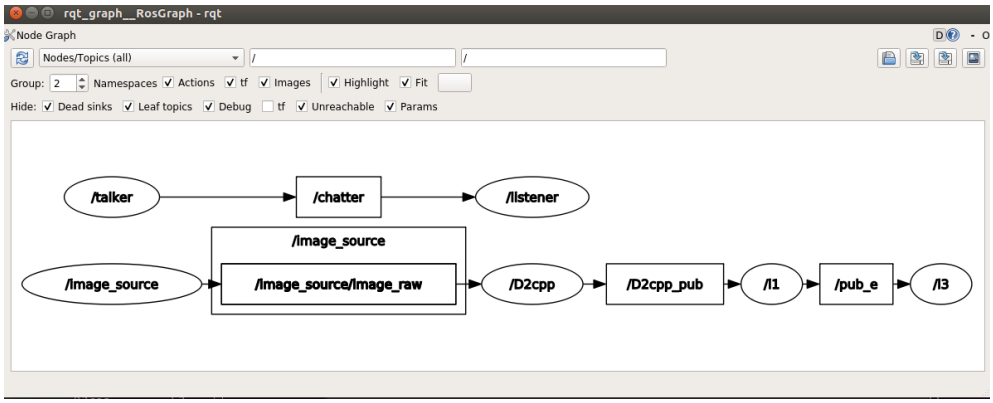


Figure 3. killing L2&D3 node, rqt_graph

- **Results:**

  Time of running: **1** hours; Crashing Error: **0**; Stuck Error: **0**

  ①Kill the node L2, and wait for about 0.3 hours, then restart the L2. Re-establishment of communication between nodes happened;

  ②Kill the node D3, and wait for about 0.3 hours, then restart the D3. Re-establishment of communication between nodes happened.

## System Resource

- **Case 4: Low Resource Run**
  - Image Process Rate to be set to **k** * 1.0, making the CPU occupancy rate **r1** / GPU occupancy rate **r2**
  - Launching S1, to run the system for **x** hours
  - Checking stuck/crashing status
- **Standard 4:** when **r1**/**r2** is larger than 80%, **x** should be larger than 1 without system crashing/stuck
- **Test Steps:**
  run the following nodes in order:
  ①ros2 run py_pubsub image_source_video (D1py)
  **②ros2 run test_pkg client_rateChange**
  **(which makes the image process rate change according to k*1.0)**
  **③ros2 run test_pkg client_time**
  **(which makes the CPU occupancy rate change dynamically from 1-8 CPU utilization processes every hour)**
- **Note:**
  **Use command `cat /proc/cpuinfo | grep 'processor' |wc -l` to check the status of CPUs**

```
root@linsc-GL553VD:~/ros2_ws/ros2_ws# cat /proc/cpuinfo | grep 'processor' |wc -
l
8
```

**Figure 4. the number of cpu to occupy**

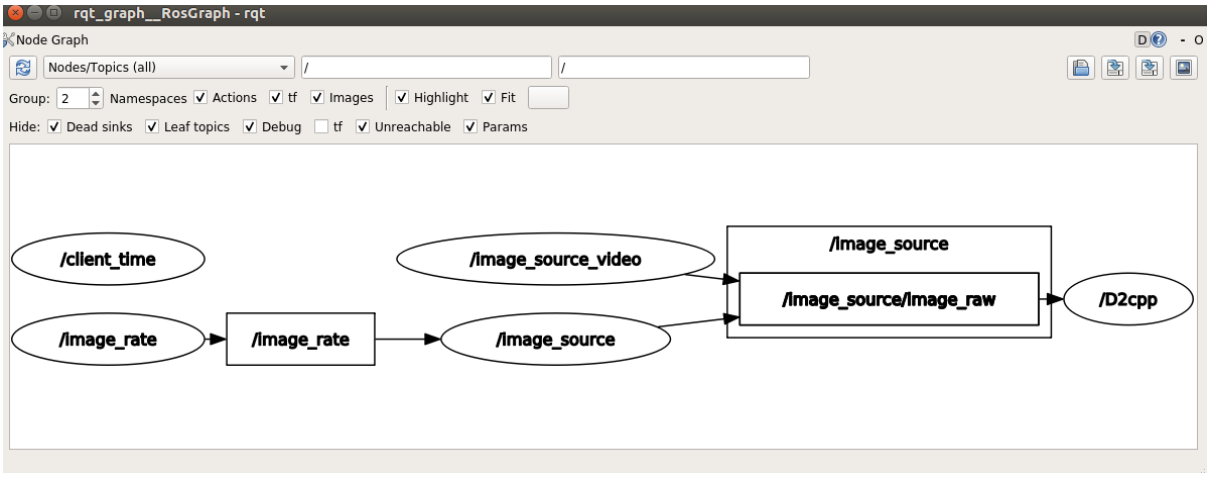


**Figure 5.Low Resource Run rqt_graph**

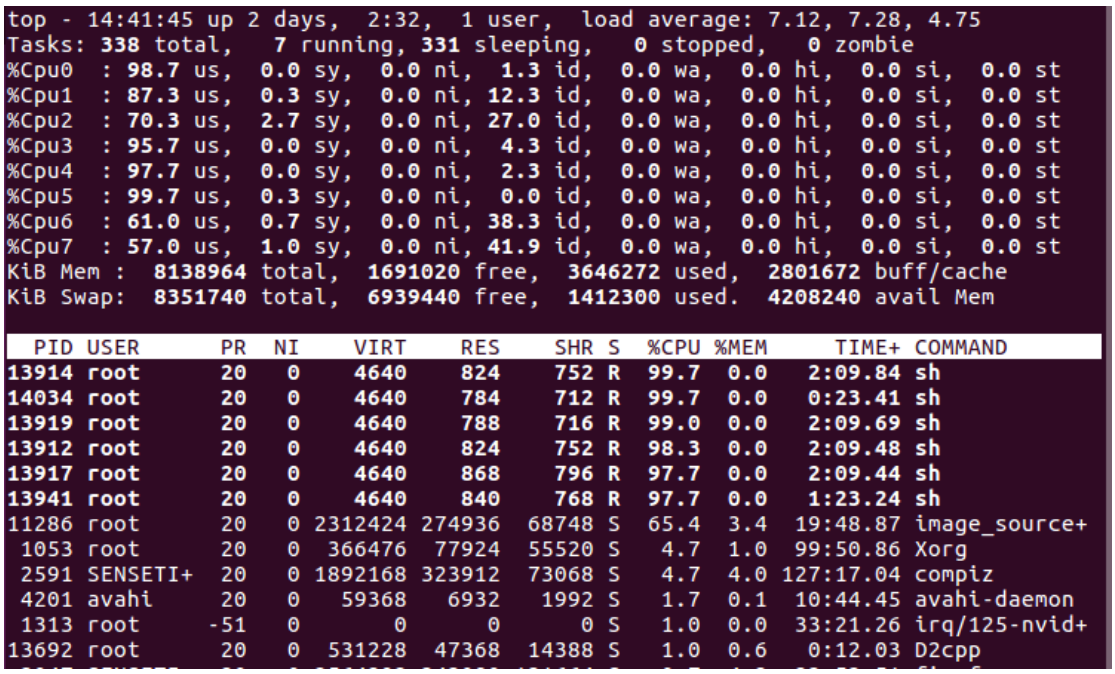


**Figure 6.cpu processes usage (6/8)**

- **Results:**

  **Time of running: 18825 seconds; Error: 0; Warning: 0**

  ①The node network is still communicating normally, and there is no crashed node. The talker and listener still works well;

  ②At **18890s**, Through the rviz2, we saw that the topic of"/image_source_raw"and the video stream was still running well;

  ③Through command `**ros2 run test_pkg client_time**` that published clients to control CPU rate, in case of high CPU usage **(5-8 CPU utilization processes),** all of nodes still work well.

- **Case 5: High Concurrency Run**
  - Duplicating D1->D2->D3 for **y1** times, marking the CPU occupancy rate **r1** / GPU occupancy rate **r2**
  - Launching S1, to run the system
  - Measuring the publisher and subscriber I/O operations per second as **y2**
  - Checking stuck/crashing status
- **Standard 5:** when **r1**/**r2** is larger than 80%, **y1** should be larger than 8, **y2** should be larger than () without system crashing/stuck

- **Test Steps:**

  Run the following nodes in order.

  ①ros2 run py_pubsub image_source_video (D1py)

  ②ros2 run test_pkg D2cpp

  ③ros2 run test_pkg D3cpp

  ④ros2 run test_pkg L1

⑥ros2 run test_pkg L2

⑦ros2 run test_pkg L3

⑧ros2 run test_pkg talker　ros2 run test_pkg listener

**⑨ros2 run test_pkg client_time**

**(which makes the CPU occupancy rate change dynamically from 1-8 CPU utilization processes every half an hour)**

```
root@linsc-GL553VD:~/ros2_ws/ros2_ws# ros2 node list
/talker
/l3
/listener
/D2cpp
/client_time
/l2
/l1
/D3cpp
/image_source
```

**Figure 7.ros2 node list**

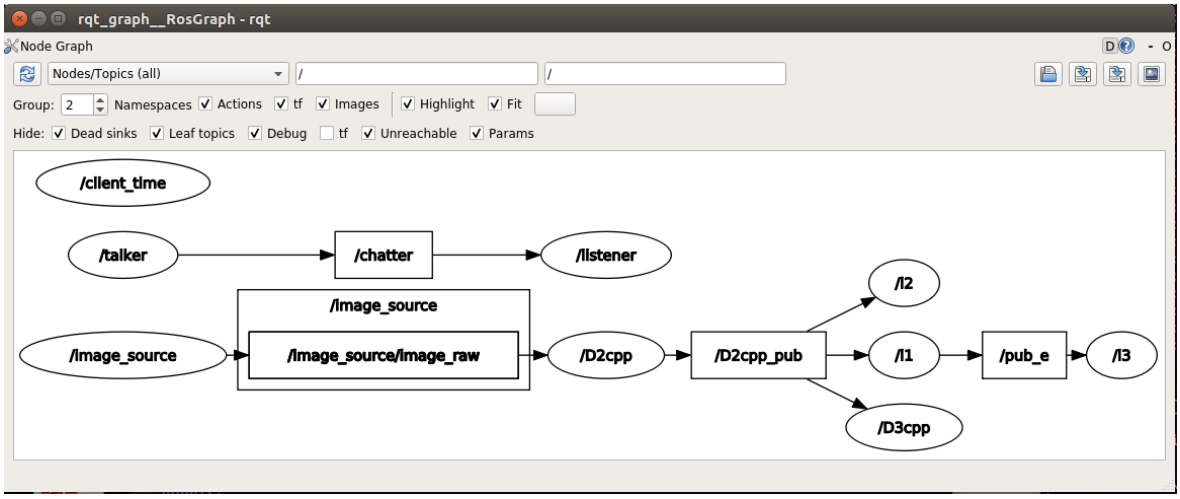


**Figure 8. High Concurrency rqt_graph**

```
top - 14:43:37 up 2 days,  2:34,  1 user,  load average: 7.55, 7.45, 5.10
Tasks: 337 total,   6 running, 331 sleeping,   0 stopped,   0 zombie
%Cpu0  : 94.4 us,  0.3 sy,  0.0 ni,  5.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  : 70.3 us,  0.3 sy,  0.0 ni, 29.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 40.6 us,  6.4 sy,  0.0 ni, 53.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 55.4 us,  0.3 sy,  0.0 ni, 44.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 33.2 us,  2.7 sy,  0.0 ni, 64.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  : 92.7 us,  1.0 sy,  0.0 ni,  6.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  8138964 total,  1720548 free,  3616024 used,  2802392 buff/cache
KiB Swap:  8351740 total,  6939504 free,  1412236 used.  4238288 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
14229 root      20   0    4640    860    792 R 100.0  0.0   0:19.26 sh
14236 root      20   0    4640    924    852 R 100.0  0.0   0:19.19 sh
14231 root      20   0    4640    784    712 R  99.7  0.0   0:19.20 sh
14234 root      20   0    4640    780    708 R  99.3  0.0   0:19.23 sh
14261 root      20   0    4640    840    768 R  95.3  0.0   0:12.05 sh
11286 root      20   0 2312424 274936  68748 S  75.7  3.4  20:56.45 image_source+
 1053 root      20   0  366476  77984  55580 S  11.3  1.0  99:55.23 Xorg
 2591 SENSETI+  20   0 1892168 324048  73140 S   6.6  4.0 127:21.32 compiz
13692 root      20   0  531228  47416  14388 S   2.7  0.6   0:14.24 D2cpp
```

**Figure 9.cpu processes usage ( 5 /8)**

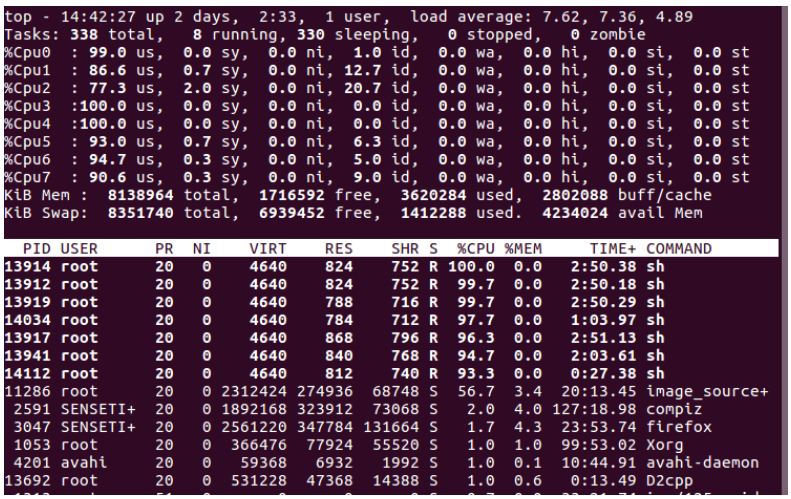


**Figure 10.cpu processes usage ( 7 /8)**

- **Results:**

**Time of running:** 93725 seconds; Error: 0; Warnings: **1**

①Look at Figure 9 and 10, the CPU processes usage changes from 5/8 to 7/8 which made the CPU utilization allocated to the node of image_source was reduced from 75.7% to 56.7%, and to the node of D2cpp was reduced from 2.7% to 1.0%.

②The node network is still communicating normally, and there is no crashed node and the talker and listener still worked well;

③At 93890s, through command `**ros2 topic echo "/D2cpp_pub" ("/pub_e /chatter …")**`, the data showed correctly and normally.

Through the rviz2, we saw that the topic of "/image_source_raw"and the video stream was still running well;

④ At 93932s, running command `**ros2 run test_pkg client 0 1(2,3,4….)** ` to publish clients that changed the number of CPU rate, it can still deliver messages normally and work well.

⑤At about > 93725 seconds, there was a warning showing. **The warning was from "ros2 run test_pkg client_time", but it did not affect the normal communication of the network**. Restarting this node can solve this.

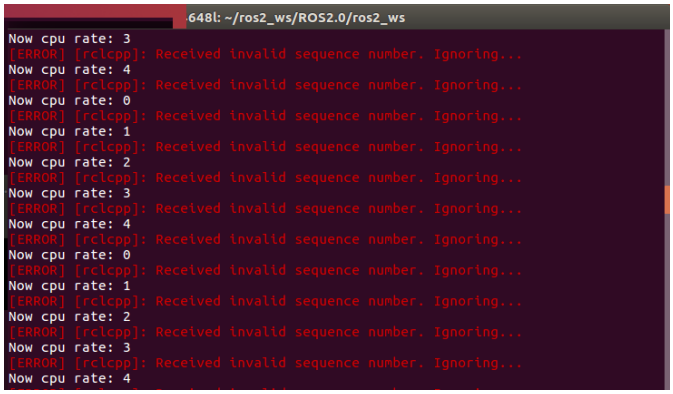


**Figure 11.The node " client_time" warning**

# Test cases for user experience

## Eligibility

- **Case 6: Low Latency Run**
  - Image Process Rate to be set to 1.0
  - Launching S1, to run the system for **x** hours
  - Measuring the system latency from D1 to D3 as **y** ms
- **Standard 6: y** should be less than 30
- **Test Steps:**

  Run the following nodes in order.

  ①ros2 run py_pubsub image_source_video (D1py)

  ②ros2 run test_pkg D2cpp

  ③ros2 run test_pkg D3cpp
- **Result:**

  Time of running: 10 min; Error: 0; Warnings:**0**

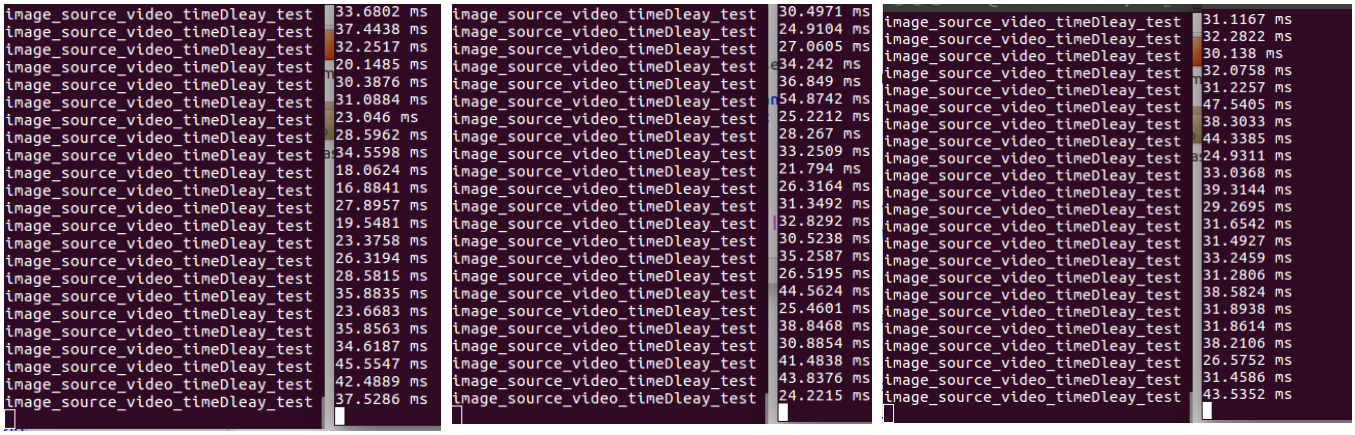


**Figure 12.The node “Image_source_video” Latency test**

①From Figure 12, the average latency of the node “image_source_video” was about 30 ms.

## Usability

- **Case 7: Interruption Run**
  - Image Process Rate to be set to 1.0
  - Launching S1, to run the system for **x** hours
  - Call Service of S1/S2 for **y** times within one reaction period
  - Checking stuck/crashing status
- **Standard 7:** when **y** is larger than 2, system should not be crashing/stuck
- **Test Steps:**

  Run the following nodes in order.

  ①ros2 run py_pubsub image_source_video (D1py)

  ②ros2 run test_pkg D2cpp

  ③ros2 run test_pkg D3cpp

  ④ros2 run test_pkg L1

⑥ros2 run test_pkg L2  (Push log **f** alternatively)

⑦ros2 run test_pkg L3  (Accept)

⑧ros2 run test_pkg talker / ros2 run test_pkg listener

**⑨ros2 run test_pkg client_time**

**(which makes the CPU occupancy rate change dynamically from 1-8 CPU utilization processes every second)**

- **Result:**

**Time of running: 12528 seconds; Error:0; Warning:0**

① The node network was still communicating normally, and there was no crashed node and the talker and listener still worked well.

- **Case 8: Data frame loss test**

Msg : **“Hello LinSC666! Test wheather LosingData...msg->data”+int[num]**

The number of Msg: **1,000,000**

CPU processes usage: **6/8  &&  1/8**

| | Sending Frequency | | | |
|---|---|---|---|---|
| | 100Hz | 500Hz | 1KHz | 10KHz |
| Number of dropped frames | 999982/1000000 (miss 18) | 999936/1000000 (miss 64) | 999896/1000000 (miss 104) | 571209/1000000 (miss 428791)<br>998973/1000000 (Low CPU processes usage: **1/8**,miss **1027**) |

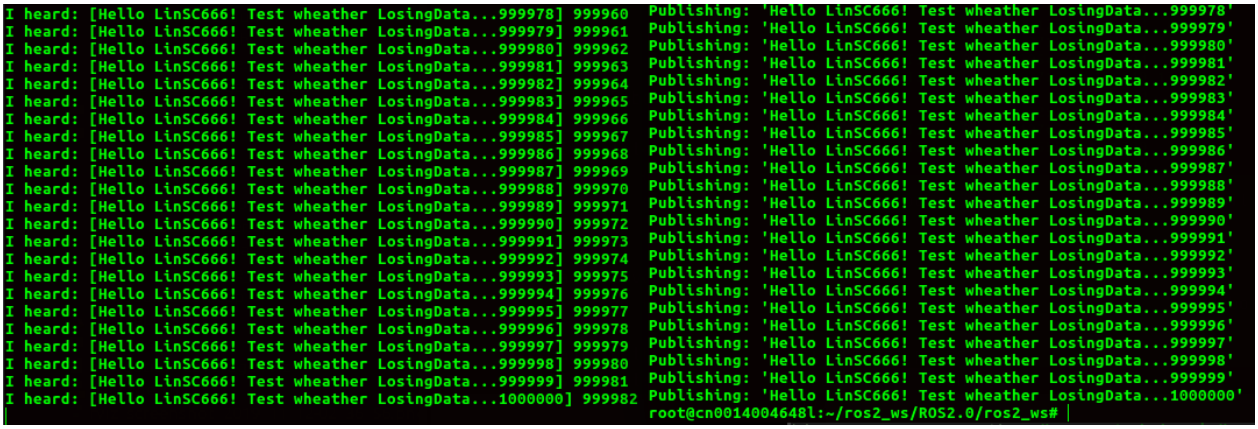


**Figure 13.Number of dropped frames under 100Hz,6/8 cpu usage**

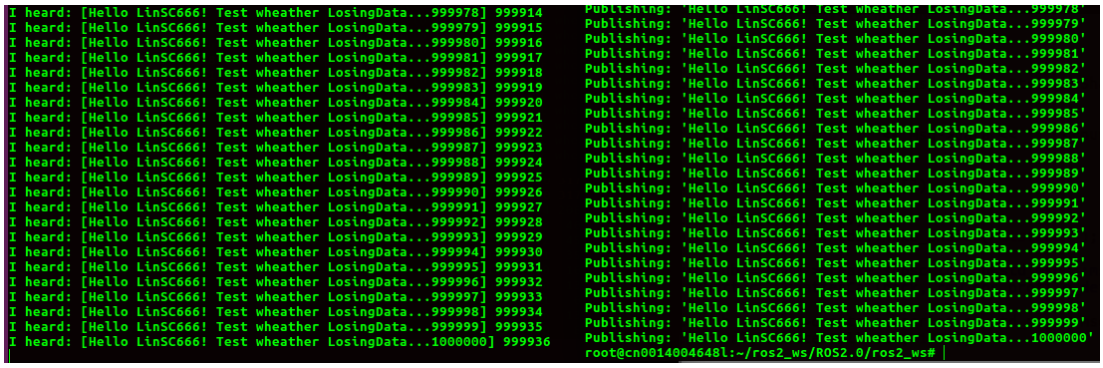


**Figure 14.Number of dropped frames under 500Hz,6/8 cpu usage**

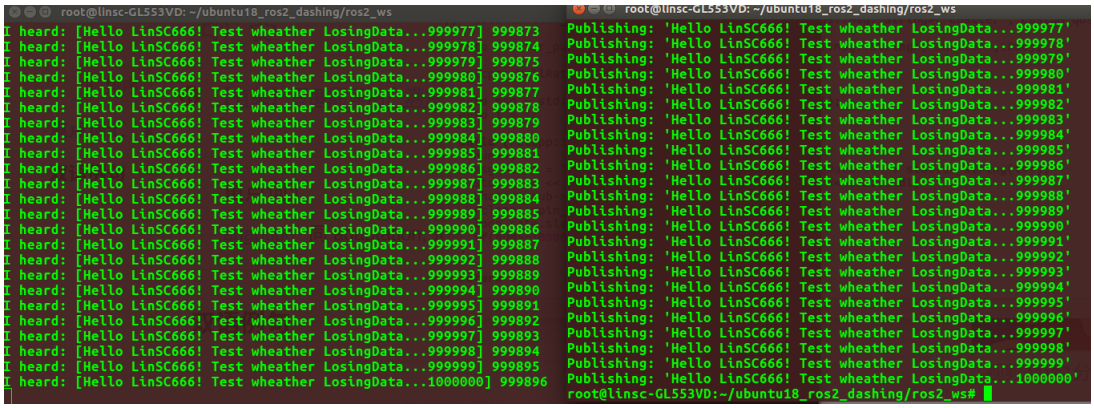


**Figure 15.Number of dropped frames under 1KHz,6/8 cpu usage**

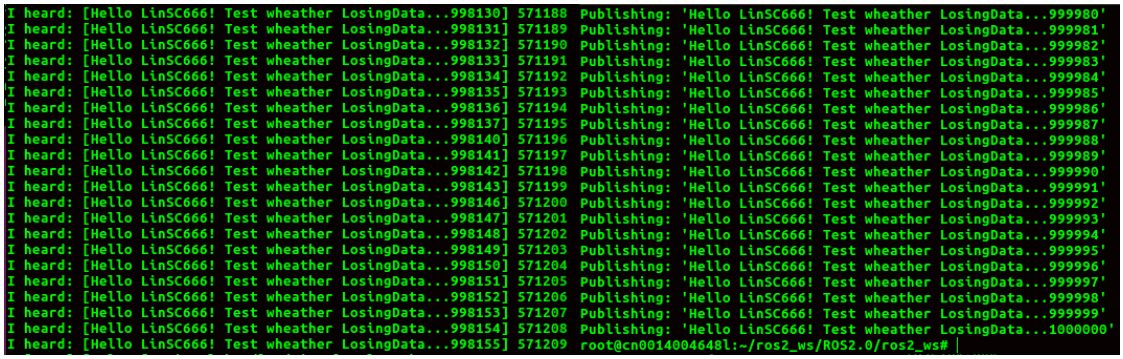


**Figure 16.Number of dropped frames under 10KHz,6/8 cpu usage（ serious data lost)**

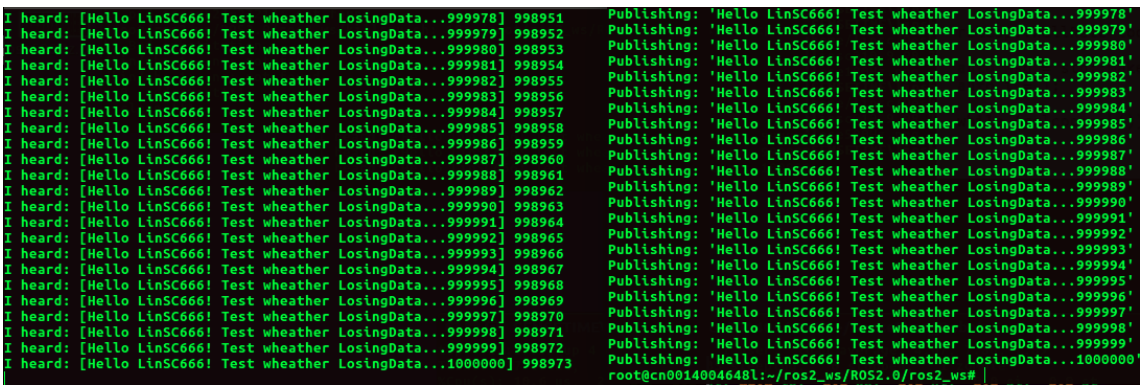


**Figure 17.Number of dropped frames under 10KHz,1/8 cpu usage**

**- Result:**

**①under CPU processes usage= 6/8 , when Sending Frequency <=1KHz, it will lose little some data frame.**

**②under CPU processes usage= 6/8,when Sending Frequency >=10KHz, it lost a lot of data frames,and it occurred serious data lost.But it showed good under CPU processes usage= 1/8.**

<table>
<tr><td colspan="4">Topic : “image_source/image_raw”<br><br>The number of Frame_id: 1,000,000<br><br>Picture resolution : 980 * 720<br><br>CPU processes usage: 6/8 && 1/8</td></tr>
<tr><td rowspan="2"></td><td colspan="3">Sending Frequency</td></tr>
<tr><td>100Hz</td><td>500Hz</td><td>1KHz</td></tr>
<tr><td rowspan="2">Number of dropped frames</td><td rowspan="2">999976/1000000<br>(high CPU processes usage: 6/8,miss 24)</td><td rowspan="2">997248/1000000<br>(high CPU processes usage: 6/8,miss 2752)</td><td>992400/1000000<br>(high CPU processes usage: 6/8,miss 7600)</td></tr>
<tr><td>997968/1000000<br>(Low CPU processes usage: 1/8,miss 2032)</td></tr>
</table>

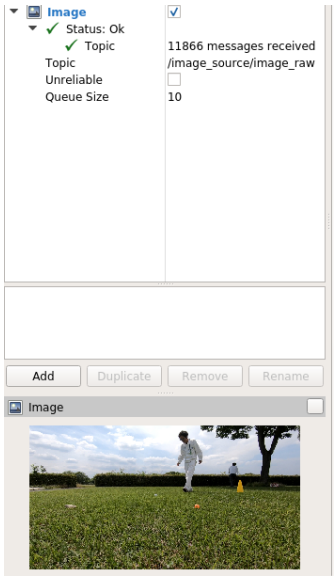


**Figure 18.The Received Image show**

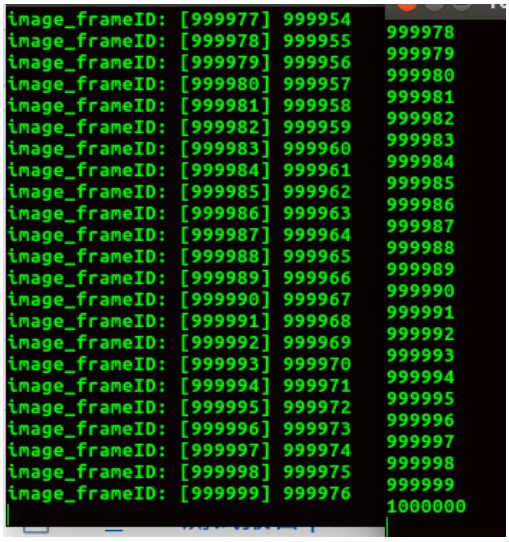


**Figure 19.Number of dropped image_frames under 100Hz,6/8 cpu usage**

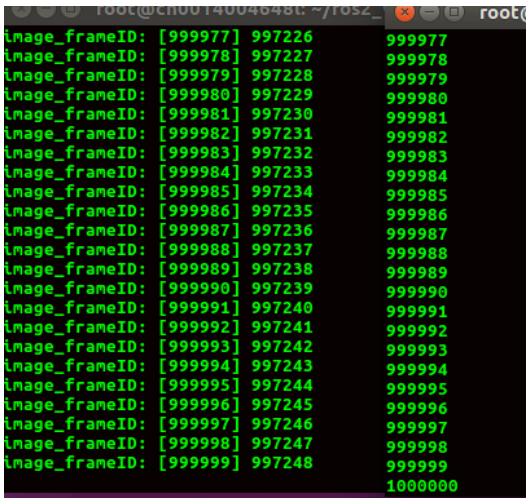


**Figure 20.Number of dropped image_frames under 500Hz,6/8 cpu usage**

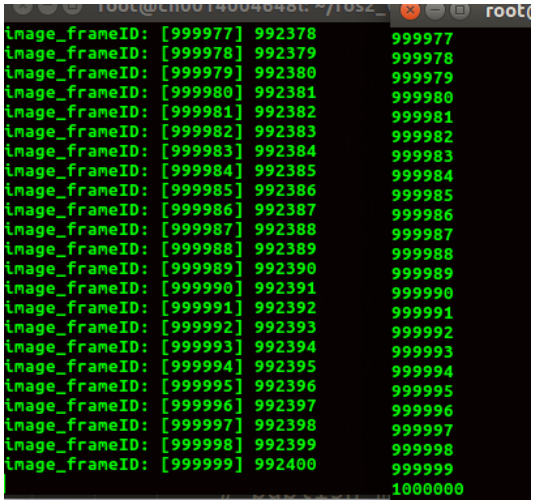


**Figure 21.Number of dropped image_frames under 1KHz,6/8 cpu usage**

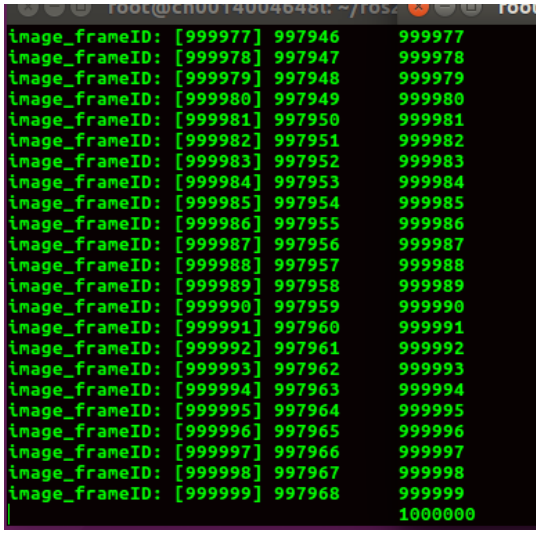


**Figure 22.Number of dropped image_frames under 1KHz,1/8 cpu usage**

- **Result:**

**①Each image resolution was 980*720, under CPU processes usage= 6/8 , when Sending Frequency <=100Hz, it will lose little some image frame.**

**②under CPU processes usage= 6/8,when Sending Frequency >=500Hz, it lost a lot of image frames,and it occurred serious data lost.But it showed good under CPU processes usage= 1/8.**