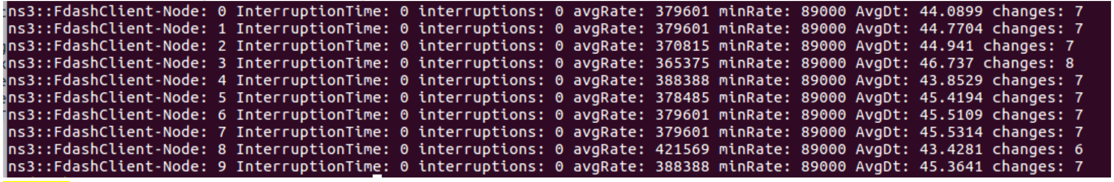
Mini Project 3 姓名:郭紘安 學號:109062578

1.How do you design your algorithm?

Step 1 : Chose Example Algorithm

FDASH : FdashClient

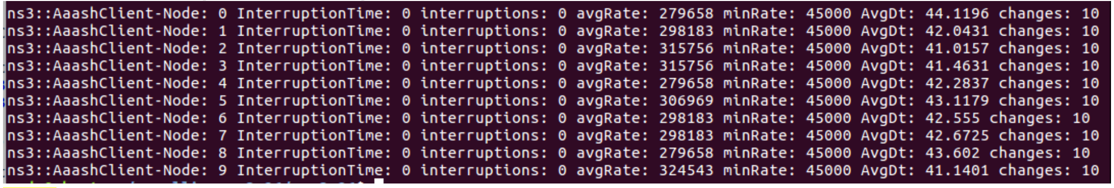
```
./waf --run 'src/dash/examples/dash-lte --users=10 --algorithms="ns3::FdashClient"
--linkRate=10Mbps --bufferSpace=100000000'
```



```
ns3::FdashClient-Node: 0 InterruptionTime: 0 interruptions: 0 avgRate: 379601 minRate: 89000 AvgDt: 44.0899 changes: 7
ns3::FdashClient-Node: 1 InterruptionTime: 0 interruptions: 0 avgRate: 379601 minRate: 89000 AvgDt: 44.7704 changes: 7
ns3::FdashClient-Node: 2 InterruptionTime: 0 interruptions: 0 avgRate: 370815 minRate: 89000 AvgDt: 44.941 changes: 7
ns3::FdashClient-Node: 3 InterruptionTime: 0 interruptions: 0 avgRate: 365375 minRate: 89000 AvgDt: 46.737 changes: 8
ns3::FdashClient-Node: 4 InterruptionTime: 0 interruptions: 0 avgRate: 388388 minRate: 89000 AvgDt: 43.8529 changes: 7
ns3::FdashClient-Node: 5 InterruptionTime: 0 interruptions: 0 avgRate: 378485 minRate: 89000 AvgDt: 45.4194 changes: 7
ns3::FdashClient-Node: 6 InterruptionTime: 0 interruptions: 0 avgRate: 379601 minRate: 89000 AvgDt: 45.5109 changes: 7
ns3::FdashClient-Node: 7 InterruptionTime: 0 interruptions: 0 avgRate: 379601 minRate: 89000 AvgDt: 45.5314 changes: 7
ns3::FdashClient-Node: 8 InterruptionTime: 0 interruptions: 0 avgRate: 421569 minRate: 89000 AvgDt: 43.4281 changes: 6
ns3::FdashClient-Node: 9 InterruptionTime: 0 interruptions: 0 avgRate: 388388 minRate: 89000 AvgDt: 45.3641 changes: 7
```

AAASH : AaashClient

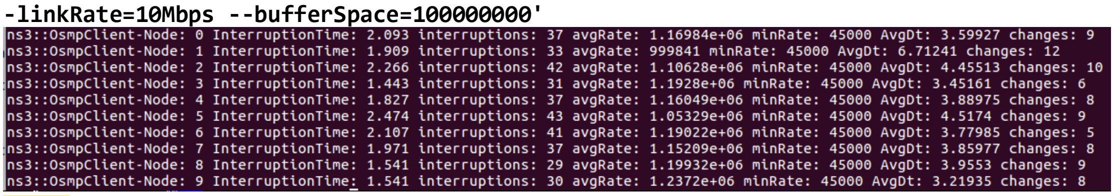
```
./waf --run 'src/dash/examples/dash-lte --users=10 --algorithms="ns3::AaashClient"
--linkRate=10Mbps --bufferSpace=100000000'
```



```
ns3::AaashClient-Node: 0 InterruptionTime: 0 interruptions: 0 avgRate: 279658 minRate: 45000 AvgDt: 44.1196 changes: 10
ns3::AaashClient-Node: 1 InterruptionTime: 0 interruptions: 0 avgRate: 298183 minRate: 45000 AvgDt: 42.0431 changes: 10
ns3::AaashClient-Node: 2 InterruptionTime: 0 interruptions: 0 avgRate: 315756 minRate: 45000 AvgDt: 41.0157 changes: 10
ns3::AaashClient-Node: 3 InterruptionTime: 0 interruptions: 0 avgRate: 315756 minRate: 45000 AvgDt: 41.4631 changes: 10
ns3::AaashClient-Node: 4 InterruptionTime: 0 interruptions: 0 avgRate: 279658 minRate: 45000 AvgDt: 42.2837 changes: 10
ns3::AaashClient-Node: 5 InterruptionTime: 0 interruptions: 0 avgRate: 306969 minRate: 45000 AvgDt: 43.1179 changes: 10
ns3::AaashClient-Node: 6 InterruptionTime: 0 interruptions: 0 avgRate: 298183 minRate: 45000 AvgDt: 42.555 changes: 10
ns3::AaashClient-Node: 7 InterruptionTime: 0 interruptions: 0 avgRate: 298183 minRate: 45000 AvgDt: 42.6725 changes: 10
ns3::AaashClient-Node: 8 InterruptionTime: 0 interruptions: 0 avgRate: 279658 minRate: 45000 AvgDt: 43.602 changes: 10
ns3::AaashClient-Node: 9 InterruptionTime: 0 interruptions: 0 avgRate: 324543 minRate: 45000 AvgDt: 41.1401 changes: 10
```

OSMP : OsmpClient

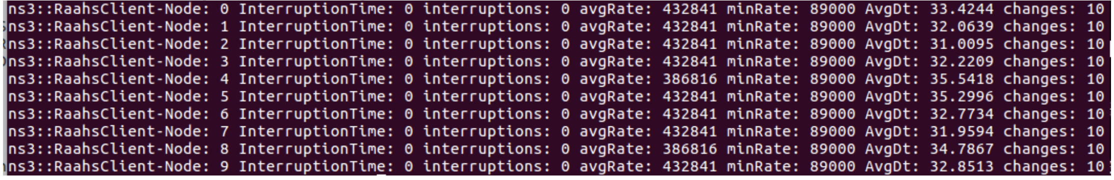
```
./waf --run 'src/dash/examples/dash-lte --users=10 --algorithms="ns3::OsmpClient" -
-linkRate=10Mbps --bufferSpace=100000000'
```



```
ns3::OsmpClient-Node: 0 InterruptionTime: 2.093 interruptions: 37 avgRate: 1.16984e+06 minRate: 45000 AvgDt: 3.59927 changes: 9
ns3::OsmpClient-Node: 1 InterruptionTime: 1.909 interruptions: 33 avgRate: 999841 minRate: 45000 AvgDt: 6.71241 changes: 12
ns3::OsmpClient-Node: 2 InterruptionTime: 2.266 interruptions: 42 avgRate: 1.10628e+06 minRate: 45000 AvgDt: 4.45513 changes: 10
ns3::OsmpClient-Node: 3 InterruptionTime: 1.443 interruptions: 31 avgRate: 1.1928e+06 minRate: 45000 AvgDt: 3.45161 changes: 6
ns3::OsmpClient-Node: 4 InterruptionTime: 1.827 interruptions: 37 avgRate: 1.16049e+06 minRate: 45000 AvgDt: 3.88975 changes: 8
ns3::OsmpClient-Node: 5 InterruptionTime: 2.474 interruptions: 43 avgRate: 1.05329e+06 minRate: 45000 AvgDt: 4.5174 changes: 9
ns3::OsmpClient-Node: 6 InterruptionTime: 2.107 interruptions: 41 avgRate: 1.19022e+06 minRate: 45000 AvgDt: 3.77985 changes: 5
ns3::OsmpClient-Node: 7 InterruptionTime: 1.971 interruptions: 37 avgRate: 1.15209e+06 minRate: 45000 AvgDt: 3.85977 changes: 8
ns3::OsmpClient-Node: 8 InterruptionTime: 1.541 interruptions: 29 avgRate: 1.19932e+06 minRate: 45000 AvgDt: 3.9553 changes: 9
ns3::OsmpClient-Node: 9 InterruptionTime: 1.541 interruptions: 30 avgRate: 1.2372e+06 minRate: 45000 AvgDt: 3.21935 changes: 8
```

RAAHS : RaahsClient

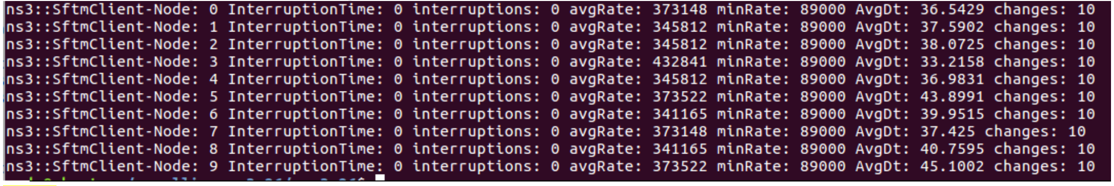
```
./waf --run 'src/dash/examples/dash-lte --users=10 --algorithms="ns3::RaahsClient"
--linkRate=10Mbps --bufferSpace=100000000'
```



```
ns3::RaahsClient-Node: 0 InterruptionTime: 0 interruptions: 0 avgRate: 432841 minRate: 89000 AvgDt: 33.4244 changes: 10
ns3::RaahsClient-Node: 1 InterruptionTime: 0 interruptions: 0 avgRate: 432841 minRate: 89000 AvgDt: 32.0639 changes: 10
ns3::RaahsClient-Node: 2 InterruptionTime: 0 interruptions: 0 avgRate: 432841 minRate: 89000 AvgDt: 31.0095 changes: 10
ns3::RaahsClient-Node: 3 InterruptionTime: 0 interruptions: 0 avgRate: 432841 minRate: 89000 AvgDt: 32.2209 changes: 10
ns3::RaahsClient-Node: 4 InterruptionTime: 0 interruptions: 0 avgRate: 386816 minRate: 89000 AvgDt: 35.5418 changes: 10
ns3::RaahsClient-Node: 5 InterruptionTime: 0 interruptions: 0 avgRate: 432841 minRate: 89000 AvgDt: 35.2996 changes: 10
ns3::RaahsClient-Node: 6 InterruptionTime: 0 interruptions: 0 avgRate: 432841 minRate: 89000 AvgDt: 32.7734 changes: 10
ns3::RaahsClient-Node: 7 InterruptionTime: 0 interruptions: 0 avgRate: 432841 minRate: 89000 AvgDt: 31.9594 changes: 10
ns3::RaahsClient-Node: 8 InterruptionTime: 0 interruptions: 0 avgRate: 386816 minRate: 89000 AvgDt: 34.7867 changes: 10
ns3::RaahsClient-Node: 9 InterruptionTime: 0 interruptions: 0 avgRate: 432841 minRate: 89000 AvgDt: 32.8513 changes: 10
```

SFTM : SftmClient

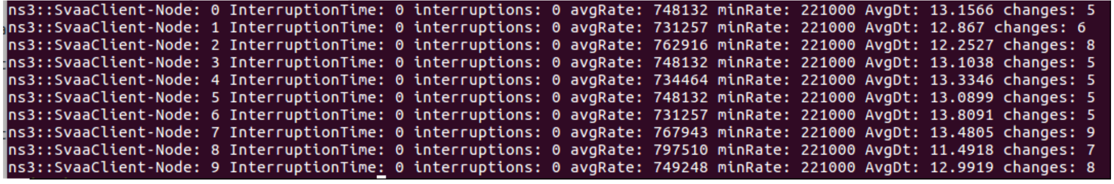
```
./waf --run 'src/dash/examples/dash-lte --users=10 --algorithms="ns3::SftmClient" -
-linkRate=10Mbps --bufferSpace=100000000'
```



```
ns3::SftmClient-Node: 0 InterruptionTime: 0 interruptions: 0 avgRate: 373148 minRate: 89000 AvgDt: 36.5429 changes: 10
ns3::SftmClient-Node: 1 InterruptionTime: 0 interruptions: 0 avgRate: 345812 minRate: 89000 AvgDt: 37.5902 changes: 10
ns3::SftmClient-Node: 2 InterruptionTime: 0 interruptions: 0 avgRate: 345812 minRate: 89000 AvgDt: 38.0725 changes: 10
ns3::SftmClient-Node: 3 InterruptionTime: 0 interruptions: 0 avgRate: 432841 minRate: 89000 AvgDt: 33.2158 changes: 10
ns3::SftmClient-Node: 4 InterruptionTime: 0 interruptions: 0 avgRate: 345812 minRate: 89000 AvgDt: 36.9831 changes: 10
ns3::SftmClient-Node: 5 InterruptionTime: 0 interruptions: 0 avgRate: 373522 minRate: 89000 AvgDt: 43.8991 changes: 10
ns3::SftmClient-Node: 6 InterruptionTime: 0 interruptions: 0 avgRate: 341165 minRate: 89000 AvgDt: 39.9515 changes: 10
ns3::SftmClient-Node: 7 InterruptionTime: 0 interruptions: 0 avgRate: 373148 minRate: 89000 AvgDt: 37.425 changes: 10
ns3::SftmClient-Node: 8 InterruptionTime: 0 interruptions: 0 avgRate: 341165 minRate: 89000 AvgDt: 40.7595 changes: 10
ns3::SftmClient-Node: 9 InterruptionTime: 0 interruptions: 0 avgRate: 373522 minRate: 89000 AvgDt: 45.1002 changes: 10
```

SVAA : SvaaClient

```
 ./waf --run 'src/dash/examples/dash-lte --users=10 --algorithms="ns3::SvaaClient" -
-linkRate=10Mbps --bufferSpace=100000000'
```



```
ns3::SvaaClient-Node: 0 InterruptionTime: 0 interruptions: 0 avgRate: 748132 minRate: 221000 AvgDt: 13.1566 changes: 5
ns3::SvaaClient-Node: 1 InterruptionTime: 0 interruptions: 0 avgRate: 731257 minRate: 221000 AvgDt: 12.867 changes: 6
ns3::SvaaClient-Node: 2 InterruptionTime: 0 interruptions: 0 avgRate: 762916 minRate: 221000 AvgDt: 12.2527 changes: 8
ns3::SvaaClient-Node: 3 InterruptionTime: 0 interruptions: 0 avgRate: 748132 minRate: 221000 AvgDt: 13.1038 changes: 5
ns3::SvaaClient-Node: 4 InterruptionTime: 0 interruptions: 0 avgRate: 734464 minRate: 221000 AvgDt: 13.3346 changes: 5
ns3::SvaaClient-Node: 5 InterruptionTime: 0 interruptions: 0 avgRate: 748132 minRate: 221000 AvgDt: 13.0899 changes: 5
ns3::SvaaClient-Node: 6 InterruptionTime: 0 interruptions: 0 avgRate: 731257 minRate: 221000 AvgDt: 13.8091 changes: 5
ns3::SvaaClient-Node: 7 InterruptionTime: 0 interruptions: 0 avgRate: 767943 minRate: 221000 AvgDt: 13.4805 changes: 9
ns3::SvaaClient-Node: 8 InterruptionTime: 0 interruptions: 0 avgRate: 797510 minRate: 221000 AvgDt: 11.4918 changes: 7
ns3::SvaaClient-Node: 9 InterruptionTime: 0 interruptions: 0 avgRate: 749248 minRate: 221000 AvgDt: 12.9919 changes: 8
```

Step 2 : Design Algorithm

使用 Example Algorithm 的 fdash-client.cc 來設計演算法

Fuzzy Logic Controllers (FLCs)是模糊邏輯的最重要應用之一。

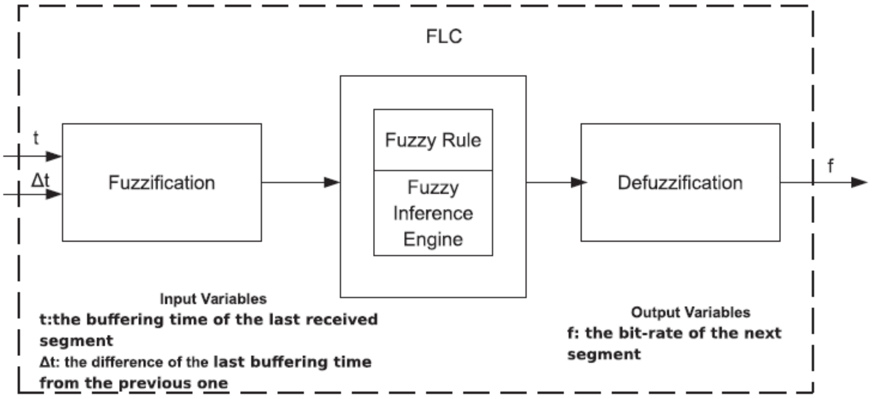
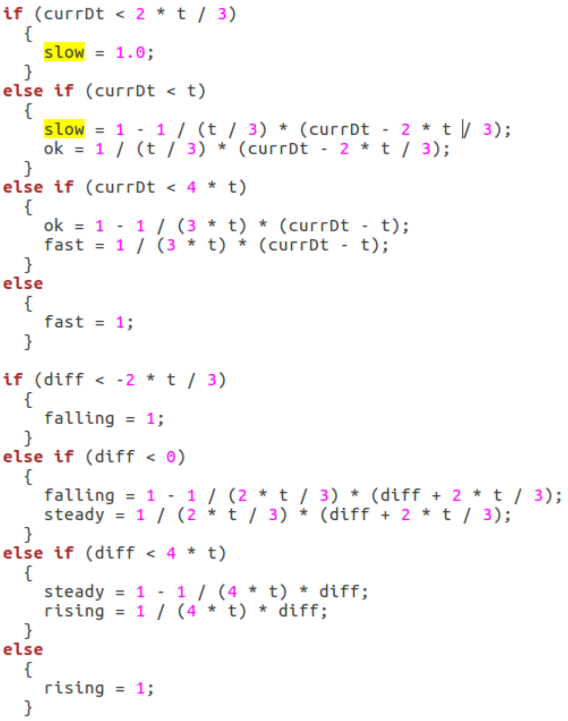

FLC
t
Δt
Fuzzification
Fuzzy Rule
Fuzzy Inference Engine
Defuzzification
f
Input Variables
t:the buffering time of the last received segment
Δt: the difference of the last buffering time from the previous one
Output Variables
f: the bit-rate of the next segment

Fig. 1. Structure of the FLC.

1. Fuzzification: 在此過程中，通過在一個或幾個隸屬度函數中查找，將輸入數據的每個元素轉換為隸屬度。



```
if (currDt < 2 * t / 3)
  {
    slow = 1.0;
  }
else if (currDt < t)
  {
    slow = 1 - 1 / (t / 3) * (currDt - 2 * t / 3);
    ok = 1 / (t / 3) * (currDt - 2 * t / 3);
  }
else if (currDt < 4 * t)
  {
    ok = 1 - 1 / (3 * t) * (currDt - t);
    fast = 1 / (3 * t) * (currDt - t);
  }
else
  {
    fast = 1;
  }

if (diff < -2 * t / 3)
  {
    falling = 1;
  }
else if (diff < 0)
  {
    falling = 1 - 1 / (2 * t / 3) * (diff + 2 * t / 3);
    steady = 1 / (2 * t / 3) * (diff + 2 * t / 3);
  }
else if (diff < 4 * t)
  {
    steady = 1 - 1 / (4 * t) * diff;
    rising = 1 / (4 * t) * diff;
  }
else
  {
    rising = 1;
  }
```

2. Fuzzy rule (or knowledge) base: 模糊規則是具有條件和結論的簡單 if-then 規則。

fuzzy if-then rules:

Rule 1 (r1): if (short) and (falling) then R

Rule 2 (r2): if (close) and (falling) then SR

Rule 3 (r3): if (long) and (falling) then NC

Rule 4 (r4): if (short) and (steady) then SR

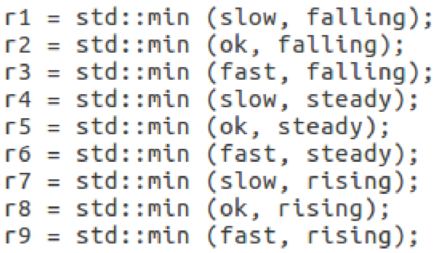
Rule 5 (r5): if (close) and (steady) then NC

Rule 6 (r6): if (long) and (steady) then SI

Rule 7 (r7): if (short) and (rising) then NC

Rule 8 (r8): if (close) and (rising) then SI

Rule 9 (r9): if (long) and (rising) then I

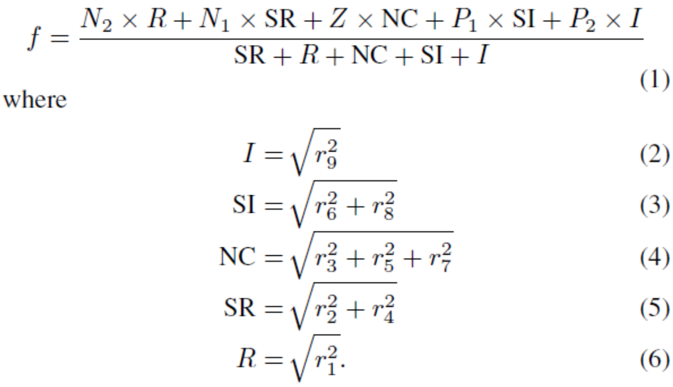


```
r1 = std::min (slow, falling);
r2 = std::min (ok, falling);
r3 = std::min (fast, falling);
r4 = std::min (slow, steady);
r5 = std::min (ok, steady);
r6 = std::min (fast, steady);
r7 = std::min (slow, rising);
r8 = std::min (ok, rising);
r9 = std::min (fast, rising);
```

3. Fuzzy rule Fuzzy inference engine: 通過計算激活程度和每個規則的輸出來執行模糊推理過程。
   Defuzzification: 在此步驟過程中，模糊集將轉換為清晰集。

f 表示下一段分辨率的增加/減少因子。

輸出的 linguistic variables 分為減少（R）,小減少（SR）,不變（NC）,小增加（SI）和增加（I）。



$$f = \frac{N_2 \times R + N_1 \times \mathrm{SR} + Z \times \mathrm{NC} + P_1 \times \mathrm{SI} + P_2 \times I}{\mathrm{SR} + R + \mathrm{NC} + \mathrm{SI} + I} \qquad (1)$$

where

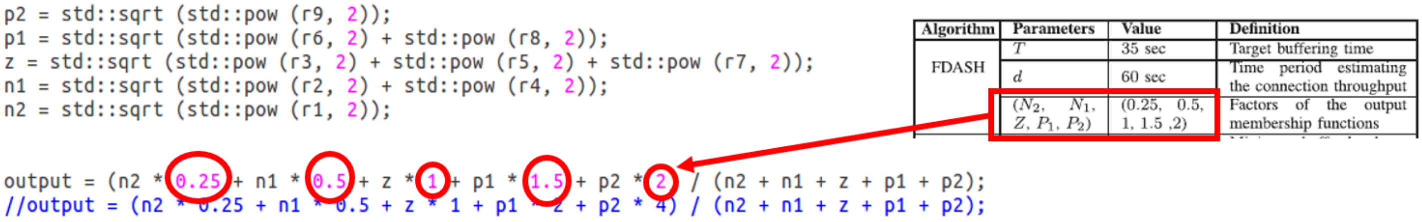
$$I = \sqrt{r_9^2} \qquad (2)$$

$$\mathrm{SI} = \sqrt{r_6^2 + r_8^2} \qquad (3)$$

$$\mathrm{NC} = \sqrt{r_3^2 + r_5^2 + r_7^2} \qquad (4)$$

$$\mathrm{SR} = \sqrt{r_2^2 + r_4^2} \qquad (5)$$

$$R = \sqrt{r_1^2}. \qquad (6)$$

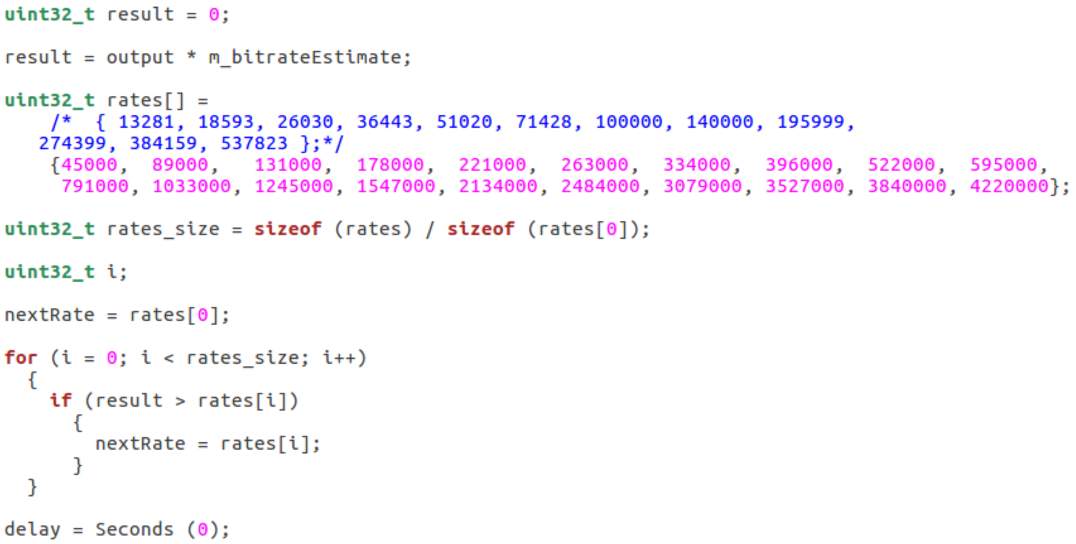


```
p2 = std::sqrt (std::pow (r9, 2));
p1 = std::sqrt (std::pow (r6, 2) + std::pow (r8, 2));
z = std::sqrt (std::pow (r3, 2) + std::pow (r5, 2) + std::pow (r7, 2));
n1 = std::sqrt (std::pow (r2, 2) + std::pow (r4, 2));
n2 = std::sqrt (std::pow (r1, 2));


output = (n2 * 0.25 + n1 * 0.5 + z * 1 + p1 * 1.5 + p2 * 2) / (n2 + n1 + z + p1 + p2);
//output = (n2 * 0.25 + n1 * 0.5 + z * 1 + p1 * 2 + p2 * 4) / (n2 + n1 + z + p1 + p2);
```

Algorithm	Parameters	Value	Definition
FDASH	T	35 sec	Target buffering time
	d	60 sec	Time period estimating the connection throughput
	(N_2, N_1, Z, P_1, P_2)	(0.25, 0.5, 1, 1.5 ,2)	Factors of the output membership functions

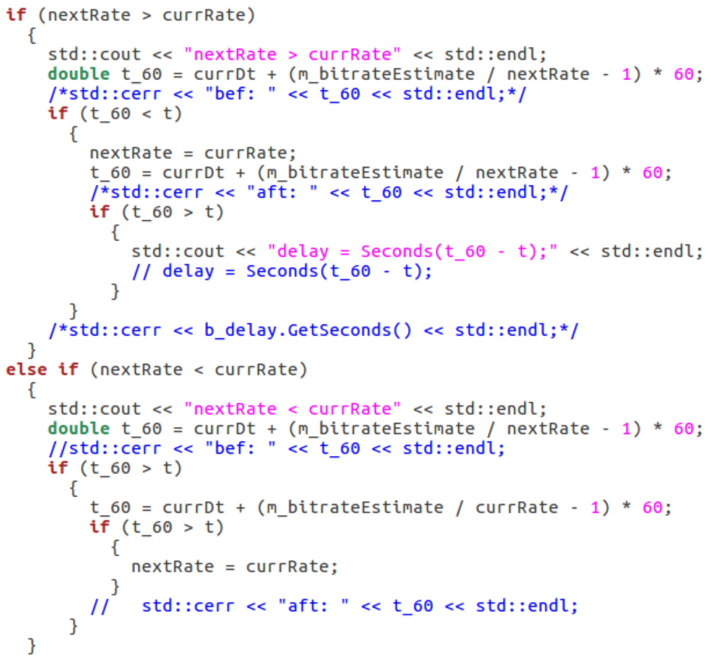
PS:這邊有透過 Paper 中提供的參數來修改 fdash-client.cc!!

4. 之後將 rate 往上加一直加到最佳的傳輸速度後更新到 nextRate



```
uint32_t result = 0;

result = output * m_bitrateEstimate;

uint32_t rates[] =
    /*  { 13281, 18593, 26030, 36443, 51020, 71428, 100000, 140000, 195999,
   274399, 384159, 537823 };*/
    {45000,  89000,   131000,  178000,  221000,  263000,  334000,  396000,  522000,  595000,
     791000, 1033000, 1245000, 1547000, 2134000, 2484000, 3079000, 3527000, 3840000, 4220000};

uint32_t rates_size = sizeof (rates) / sizeof (rates[0]);

uint32_t i;

nextRate = rates[0];

for (i = 0; i < rates_size; i++)
  {
    if (result > rates[i])
      {
        nextRate = rates[i];
      }
  }

delay = Seconds (0);
```

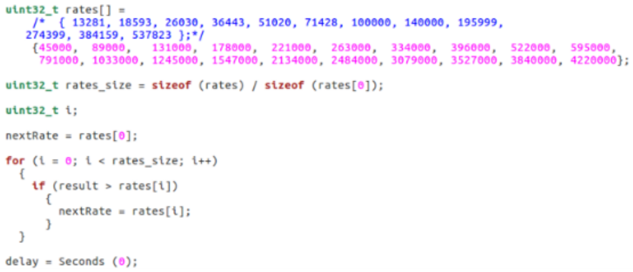
5. 然後先算 60 秒後會不會爆掉，藉此調整 nextRate



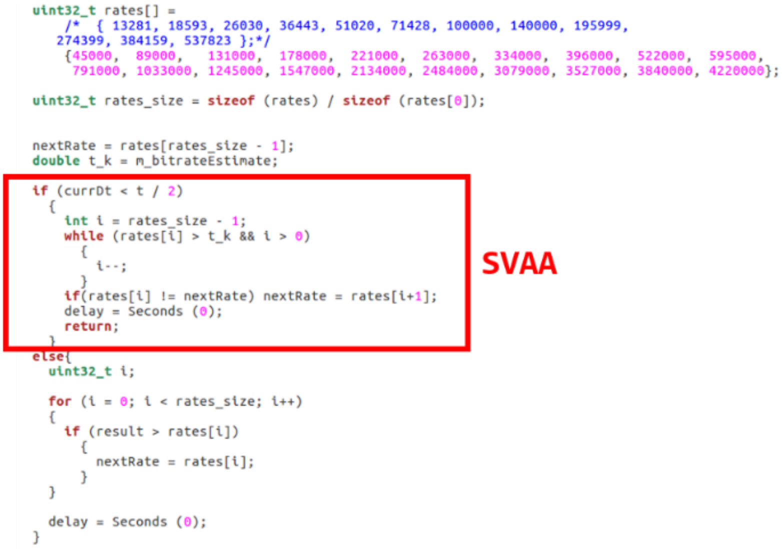
```
if (nextRate > currRate)
  {
    std::cout << "nextRate > currRate" << std::endl;
    double t_60 = currDt + (m_bitrateEstimate / nextRate - 1) * 60;
    /*std::cerr << "bef: " << t_60 << std::endl;*/
    if (t_60 < t)
      {
        nextRate = currRate;
        t_60 = currDt + (m_bitrateEstimate / nextRate - 1) * 60;
        /*std::cerr << "aft: " << t_60 << std::endl;*/
        if (t_60 > t)
          {
            std::cout << "delay = Seconds(t_60 - t);" << std::endl;
            // delay = Seconds(t_60 - t);
          }
      }
    /*std::cerr << b_delay.GetSeconds() << std::endl;*/
  }
else if (nextRate < currRate)
  {
    std::cout << "nextRate < currRate" << std::endl;
    double t_60 = currDt + (m_bitrateEstimate / nextRate - 1) * 60;
    //std::cerr << "bef: " << t_60 << std::endl;
    if (t_60 > t)
      {
        t_60 = currDt + (m_bitrateEstimate / currRate - 1) * 60;
        if (t_60 > t)
          {
            nextRate = currRate;
          }
        //   std::cerr << "aft: " << t_60 << std::endl;
      }
  }
```

2.What’s the difference between yours and original algorithm?

可以從 log 中發現一開始 FDASH 的 Rate 上升很慢因此將原本 FDASH 之前加入一個 SVAA 的判斷式藉此來提升一開始的傳輸速度進而提升整體的 avgRate

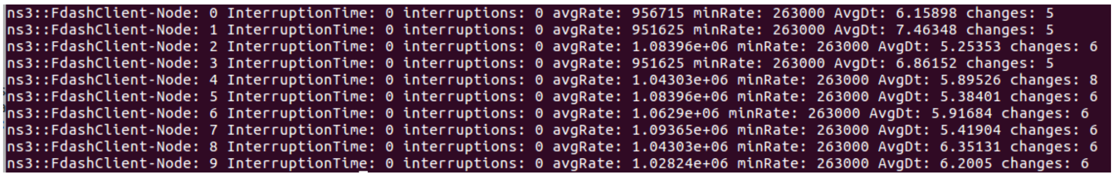


```
uint32_t rates[] =
    /*  { 13281, 18593, 26030, 36443, 51020, 71428, 100000, 140000, 195999,
   274399, 384159, 537823 };*/
    {45000,  89000,   131000,  178000,  221000,  263000,  334000,  396000,  522000,  595000,
     791000, 1033000, 1245000, 1547000, 2134000, 2484000, 3079000, 3527000, 3840000, 4220000};

uint32_t rates_size = sizeof (rates) / sizeof (rates[0]);

uint32_t i;

nextRate = rates[0];

for (i = 0; i < rates_size; i++)
  {
    if (result > rates[i])
      {
        nextRate = rates[i];
      }
  }

delay = Seconds (0);
```



```
uint32_t rates[] =
    /*  { 13281, 18593, 26030, 36443, 51020, 71428, 100000, 140000, 195999,
   274399, 384159, 537823 };*/
    {45000,  89000,   131000,  178000,  221000,  263000,  334000,  396000,  522000,  595000,
     791000, 1033000, 1245000, 1547000, 2134000, 2484000, 3079000, 3527000, 3840000, 4220000};

uint32_t rates_size = sizeof (rates) / sizeof (rates[0]);


nextRate = rates[rates_size - 1];
double t_k = m_bitrateEstimate;
if (currDt < t / 2)
  {
    int i = rates_size - 1;
    while (rates[i] > t_k && i > 0)
      {
        i--;
      }
    if(rates[i] != nextRate) nextRate = rates[i+1];
    delay = Seconds (0);
    return;
  }
else{
  uint32_t i;

  for (i = 0; i < rates_size; i++)
  {
    if (result > rates[i])
      {
        nextRate = rates[i];
      }
  }

  delay = Seconds (0);
}
```

執行結果



```
ns3::FdashClient-Node: 0 InterruptionTime: 0 interruptions: 0 avgRate: 956715 minRate: 263000 AvgDt: 6.15898 changes: 5
ns3::FdashClient-Node: 1 InterruptionTime: 0 interruptions: 0 avgRate: 951625 minRate: 263000 AvgDt: 7.46348 changes: 5
ns3::FdashClient-Node: 2 InterruptionTime: 0 interruptions: 0 avgRate: 1.08396e+06 minRate: 263000 AvgDt: 5.25353 changes: 6
ns3::FdashClient-Node: 3 InterruptionTime: 0 interruptions: 0 avgRate: 951625 minRate: 263000 AvgDt: 6.86152 changes: 5
ns3::FdashClient-Node: 4 InterruptionTime: 0 interruptions: 0 avgRate: 1.04303e+06 minRate: 263000 AvgDt: 5.89526 changes: 8
ns3::FdashClient-Node: 5 InterruptionTime: 0 interruptions: 0 avgRate: 1.08396e+06 minRate: 263000 AvgDt: 5.38401 changes: 6
ns3::FdashClient-Node: 6 InterruptionTime: 0 interruptions: 0 avgRate: 1.0629e+06 minRate: 263000 AvgDt: 5.91684 changes: 6
ns3::FdashClient-Node: 7 InterruptionTime: 0 interruptions: 0 avgRate: 1.09365e+06 minRate: 263000 AvgDt: 5.41904 changes: 6
ns3::FdashClient-Node: 8 InterruptionTime: 0 interruptions: 0 avgRate: 1.04303e+06 minRate: 263000 AvgDt: 6.35131 changes: 6
ns3::FdashClient-Node: 9 InterruptionTime: 0 interruptions: 0 avgRate: 1.02824e+06 minRate: 263000 AvgDt: 6.2005 changes: 6
```

3.What you learn?

學到如何在 LTE 架構下使用自己改過的 FDASH 演算法來優化 MPEG DASH 的影片傳輸，好讓影片傳輸的 avgRate 更高 QoE，能夠更好。