Mobile Telecommunications Networks Mini Project 3

108062586 楊子儀

- **How do you deep sign your algorithm?**

  測試下列四種演算法

FDASH (UEs 的 avgRate 大約 280000 ~ 430000 bits)

```
ns3::FdashClient-Node: 0 InterruptionTime: 0 interruptions: 0 avgRate: 374162 minRate: 89000 AvgDt: 45.7483 changes: 8
ns3::FdashClient-Node: 1 InterruptionTime: 0 interruptions: 0 avgRate: 414748 minRate: 89000 AvgDt: 41.2069 changes: 6
ns3::FdashClient-Node: 2 InterruptionTime: 0 interruptions: 0 avgRate: 404845 minRate: 89000 AvgDt: 44.5484 changes: 6
ns3::FdashClient-Node: 3 InterruptionTime: 0 interruptions: 0 avgRate: 423534 minRate: 89000 AvgDt: 43.2304 changes: 6
ns3::FdashClient-Node: 4 InterruptionTime: 0 interruptions: 0 avgRate: 414748 minRate: 89000 AvgDt: 43.7461 changes: 7
ns3::FdashClient-Node: 5 InterruptionTime: 0 interruptions: 0 avgRate: 388388 minRate: 89000 AvgDt: 45.5185 changes: 7
ns3::FdashClient-Node: 6 InterruptionTime: 0 interruptions: 0 avgRate: 414748 minRate: 89000 AvgDt: 44.3451 changes: 7
ns3::FdashClient-Node: 7 InterruptionTime: 0 interruptions: 0 avgRate: 281425 minRate: 89000 AvgDt: 48.2421 changes: 10
ns3::FdashClient-Node: 8 InterruptionTime: 0 interruptions: 0 avgRate: 277102 minRate: 89000 AvgDt: 49.3418 changes: 10
ns3::FdashClient-Node: 9 InterruptionTime: 0 interruptions: 0 avgRate: 430356 minRate: 89000 AvgDt: 42.717 changes: 6
```

RAAHS (UEs 的 avgRate 大約 350000 ~ 430000 bits)

```
ns3::RaahsClient-Node: 0 InterruptionTime: 0 interruptions: 0 avgRate: 432841 minRate: 89000 AvgDt: 31.5683 changes: 10
ns3::RaahsClient-Node: 1 InterruptionTime: 0 interruptions: 0 avgRate: 432841 minRate: 89000 AvgDt: 31.9258 changes: 10
ns3::RaahsClient-Node: 2 InterruptionTime: 0 interruptions: 0 avgRate: 432841 minRate: 89000 AvgDt: 32.0441 changes: 10
ns3::RaahsClient-Node: 3 InterruptionTime: 0 interruptions: 0 avgRate: 432841 minRate: 89000 AvgDt: 32.9503 changes: 10
ns3::RaahsClient-Node: 4 InterruptionTime: 0 interruptions: 0 avgRate: 386816 minRate: 89000 AvgDt: 35.3155 changes: 10
ns3::RaahsClient-Node: 5 InterruptionTime: 0 interruptions: 0 avgRate: 419173 minRate: 89000 AvgDt: 33.1391 changes: 10
ns3::RaahsClient-Node: 6 InterruptionTime: 0 interruptions: 0 avgRate: 432841 minRate: 89000 AvgDt: 32.1532 changes: 10
ns3::RaahsClient-Node: 7 InterruptionTime: 0 interruptions: 0 avgRate: 383887 minRate: 89000 AvgDt: 34.3252 changes: 10
ns3::RaahsClient-Node: 8 InterruptionTime: 0 interruptions: 0 avgRate: 356551 minRate: 89000 AvgDt: 34.0187 changes: 10
ns3::RaahsClient-Node: 9 InterruptionTime: 0 interruptions: 0 avgRate: 386816 minRate: 89000 AvgDt: 35.2622 changes: 10
```

SVAA (UEs 的 avgRate 大約 740000 ~ 820000 bits)

```
ns3::SvaaClient-Node: 0 InterruptionTime: 0 interruptions: 0 avgRate: 814386 minRate: 221000 AvgDt: 11.0736 changes: 7
ns3::SvaaClient-Node: 1 InterruptionTime: 0 interruptions: 0 avgRate: 779792 minRate: 221000 AvgDt: 12.2627 changes: 7
ns3::SvaaClient-Node: 2 InterruptionTime: 0 interruptions: 0 avgRate: 748132 minRate: 221000 AvgDt: 13.3227 changes: 5
ns3::SvaaClient-Node: 3 InterruptionTime: 0 interruptions: 0 avgRate: 812568 minRate: 221000 AvgDt: 11.8593 changes: 7
ns3::SvaaClient-Node: 4 InterruptionTime: 0 interruptions: 0 avgRate: 748132 minRate: 221000 AvgDt: 13.3163 changes: 5
ns3::SvaaClient-Node: 5 InterruptionTime: 0 interruptions: 0 avgRate: 829170 minRate: 221000 AvgDt: 10.3903 changes: 7
ns3::SvaaClient-Node: 6 InterruptionTime: 0 interruptions: 0 avgRate: 748132 minRate: 221000 AvgDt: 13.2176 changes: 5
ns3::SvaaClient-Node: 7 InterruptionTime: 0 interruptions: 0 avgRate: 748132 minRate: 221000 AvgDt: 13.1865 changes: 5
ns3::SvaaClient-Node: 8 InterruptionTime: 0 interruptions: 0 avgRate: 829170 minRate: 221000 AvgDt: 10.9821 changes: 7
ns3::SvaaClient-Node: 9 InterruptionTime: 0 interruptions: 0 avgRate: 762916 minRate: 221000 AvgDt: 12.8802 changes: 7
```

AAASH (UEs 的 avgRate 大約 250000 ~ 300000 bits)

```
ns3::AaashClient-Node: 0 InterruptionTime: 0 interruptions: 0 avgRate: 315756 minRate: 45000 AvgDt: 42.3762 changes: 10
ns3::AaashClient-Node: 1 InterruptionTime: 0 interruptions: 0 avgRate: 324543 minRate: 45000 AvgDt: 41.7475 changes: 10
ns3::AaashClient-Node: 2 InterruptionTime: 0 interruptions: 0 avgRate: 314805 minRate: 45000 AvgDt: 41.8847 changes: 10
ns3::AaashClient-Node: 3 InterruptionTime: 0 interruptions: 0 avgRate: 314805 minRate: 45000 AvgDt: 41.8123 changes: 10
ns3::AaashClient-Node: 4 InterruptionTime: 0 interruptions: 0 avgRate: 298183 minRate: 45000 AvgDt: 41.2624 changes: 10
ns3::AaashClient-Node: 5 InterruptionTime: 0 interruptions: 0 avgRate: 324543 minRate: 45000 AvgDt: 41.5952 changes: 10
ns3::AaashClient-Node: 6 InterruptionTime: 0 interruptions: 0 avgRate: 298183 minRate: 45000 AvgDt: 43.787 changes: 10
ns3::AaashClient-Node: 7 InterruptionTime: 0 interruptions: 0 avgRate: 324543 minRate: 45000 AvgDt: 40.9633 changes: 10
ns3::AaashClient-Node: 8 InterruptionTime: 0 interruptions: 0 avgRate: 314805 minRate: 45000 AvgDt: 41.0308 changes: 10
ns3::AaashClient-Node: 9 InterruptionTime: 0 interruptions: 0 avgRate: 269141 minRate: 45000 AvgDt: 43.5523 changes: 10
```

雖然 SVAA 的 avgRate 可以獲得較高結果，根據 FDASH: A Fuzzy-Based MPEG/DASH Adaptation Algorithm (https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7368091) 的 paper 比較圖，FDASH 的 bit rate 可以得到較好的結果，所以最後希望能用 SVAA 加上 FDASH，看是否能增進效果。

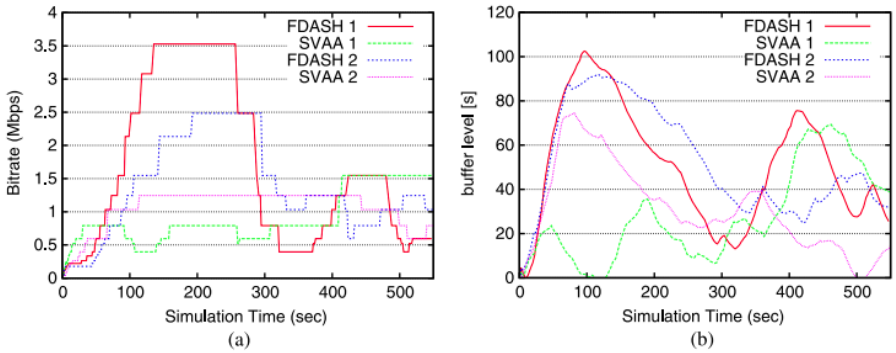

Fig. 16. Two FDASH and two SVAA clients, WiFi, with no background traffic. (a) Bit rate. (b) Buffer level.

1) modify SVAA into FDASH

SVAA 演算法使用 buffer size 估計網路狀態，當 estimate buffer size 低於 target buffer size 的一半時，表示網路狀態很差，會造成影片播放中斷。對下一段要取用的 segment， SVAA 會使用上一次使用的 bit rate。

Algorithm 1 Smooth Video Adaptation Algorithm.

1: $\tilde{v}(k) = F(k)\hat{T}(t_k^{(s)})$;
2: **if** $q(t_k^{(s)}) < \frac{q_{ref}}{2}$ **then**
3: $v(k) = Q(\tilde{T}(k-1))$;
4: return;

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

將這段加入 FDASH 中，可以發現比起原本的演算法很快就加速拿到更快的 bit rate。

origin	modify
2.936 Node: 9 newBitRate: 522000 2.957 Node: 4 newBitRate: 396000 3.037 Node: 5 newBitRate: 396000 3.098 Node: 7 newBitRate: 263000 3.175 Node: 8 newBitRate: 263000 3.31 Node: 2 newBitRate: 522000 c 3.456 Node: 7 newBitRate: 334000 3.468 Node: 0 newBitRate: 396000 3.528 Node: 1 newBitRate: 522000 3.538 Node: 8 newBitRate: 334000 3.541 Node: 6 newBitRate: 522000 3.547 Node: 4 newBitRate: 522000 3.553 Node: 3 newBitRate: 522000 3.727 Node: 5 newBitRate: 396000 3.854 Node: 9 newBitRate: 522000 4.079 Node: 7 newBitRate: 396000 4.107 Node: 0 newBitRate: 396000 4.119 Node: 2 newBitRate: 522000 4.222 Node: 8 newBitRate: 334000 4.311 Node: 5 newBitRate: 396000 4.365 Node: 6 newBitRate: 522000 4.389 Node: 4 newBitRate: 522000 4.412 Node: 1 newBitRate: 522000	2.31 Node: 9 newBitRate: 595000 c 2.323 Node: 8 newBitRate: 595000 3.051 Node: 8 newBitRate: 791000 3.069 Node: 1 newBitRate: 791000 3.103 Node: 0 newBitRate: 791000 3.106 Node: 9 newBitRate: 791000 3.147 Node: 2 newBitRate: 791000 3.147 Node: 7 newBitRate: 791000 3.149 Node: 3 newBitRate: 595000 3.15 Node: 6 newBitRate: 791000 c 3.154 Node: 5 newBitRate: 791000 3.156 Node: 4 newBitRate: 791000 3.92 Node: 3 newBitRate: 791000 c 4.118 Node: 1 newBitRate: 791000 4.333 Node: 0 newBitRate: 791000 4.358 Node: 8 newBitRate: 791000 4.379 Node: 4 newBitRate: 791000 4.382 Node: 9 newBitRate: 791000 4.426 Node: 5 newBitRate: 791000 4.478 Node: 6 newBitRate: 791000 4.531 Node: 2 newBitRate: 791000 4.586 Node: 7 newBitRate: 791000

2) modify FDASH

但修改後，發現結果和 SVAA 的結果很相似，所以修改 buffer size 為 taget size 的 ⅓ 時，在拿上一個 segment 的 bit rate 去估計 60 秒後的 buffer，若結果能大於 target，再用現在的 bit rate 去估計 60 秒後還能得到不錯的結果，則使用現在的 bit rate，反之則使用上一個 segment 的 bit rate。

```
if (currDt < 2 * t / 3 )
  {
    int i = rates_size - 1;
    while (rates[i] > m_bitrateEstimate && i > 0)
      {
        i--;
      }

    nextRate = rates[i]; //origin

    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;
          }
        else
          {
            nextRate = rates[i];
          }
      }
    else
      {
        nextRate = rates[i];
      }

    delay = Seconds (0);
    return;
  }
```

• **What's the difference between yours and the original algorithm?**

1) original algorithm (FDASH):

```
ns3::FdashClient-Node: 0 InterruptionTime: 0 interruptions: 0 avgRate: 374162 minRate: 89000 AvgDt: 45.7483 changes: 8
ns3::FdashClient-Node: 1 InterruptionTime: 0 interruptions: 0 avgRate: 414748 minRate: 89000 AvgDt: 41.2069 changes: 6
ns3::FdashClient-Node: 2 InterruptionTime: 0 interruptions: 0 avgRate: 404845 minRate: 89000 AvgDt: 44.5484 changes: 6
ns3::FdashClient-Node: 3 InterruptionTime: 0 interruptions: 0 avgRate: 423534 minRate: 89000 AvgDt: 43.2304 changes: 6
ns3::FdashClient-Node: 4 InterruptionTime: 0 interruptions: 0 avgRate: 414748 minRate: 89000 AvgDt: 43.7461 changes: 7
ns3::FdashClient-Node: 5 InterruptionTime: 0 interruptions: 0 avgRate: 388388 minRate: 89000 AvgDt: 45.5185 changes: 7
ns3::FdashClient-Node: 6 InterruptionTime: 0 interruptions: 0 avgRate: 414748 minRate: 89000 AvgDt: 44.3451 changes: 7
ns3::FdashClient-Node: 7 InterruptionTime: 0 interruptions: 0 avgRate: 281425 minRate: 89000 AvgDt: 48.2421 changes: 10
ns3::FdashClient-Node: 8 InterruptionTime: 0 interruptions: 0 avgRate: 277102 minRate: 89000 AvgDt: 49.3418 changes: 10
ns3::FdashClient-Node: 9 InterruptionTime: 0 interruptions: 0 avgRate: 430356 minRate: 89000 AvgDt: 42.717 changes: 6
```

2) modify version 1

```
ns3::FdashClient-Node: 0 InterruptionTime: 0 interruptions: 0 avgRate: 814386 minRate: 221000 AvgDt: 11.0736 changes: 7
ns3::FdashClient-Node: 1 InterruptionTime: 0 interruptions: 0 avgRate: 779792 minRate: 221000 AvgDt: 12.2627 changes: 7
ns3::FdashClient-Node: 2 InterruptionTime: 0 interruptions: 0 avgRate: 748132 minRate: 221000 AvgDt: 13.3227 changes: 5
ns3::FdashClient-Node: 3 InterruptionTime: 0 interruptions: 0 avgRate: 812568 minRate: 221000 AvgDt: 11.8593 changes: 7
ns3::FdashClient-Node: 4 InterruptionTime: 0 interruptions: 0 avgRate: 748132 minRate: 221000 AvgDt: 13.3163 changes: 5
ns3::FdashClient-Node: 5 InterruptionTime: 0 interruptions: 0 avgRate: 829170 minRate: 221000 AvgDt: 10.3903 changes: 7
ns3::FdashClient-Node: 6 InterruptionTime: 0 interruptions: 0 avgRate: 748132 minRate: 221000 AvgDt: 13.2176 changes: 5
ns3::FdashClient-Node: 7 InterruptionTime: 0 interruptions: 0 avgRate: 748132 minRate: 221000 AvgDt: 13.1865 changes: 5
ns3::FdashClient-Node: 8 InterruptionTime: 0 interruptions: 0 avgRate: 829170 minRate: 221000 AvgDt: 10.9821 changes: 7
ns3::FdashClient-Node: 9 InterruptionTime: 0 interruptions: 0 avgRate: 762916 minRate: 221000 AvgDt: 12.8802 changes: 7
```

But the result is similar to SVAA algorithm.

3) modify version 2

```
ns3::FdashClient-Node: 0 InterruptionTime: 0 interruptions: 0 avgRate: 814386 minRate: 221000 AvgDt: 11.0736 changes: 7
ns3::FdashClient-Node: 1 InterruptionTime: 0 interruptions: 0 avgRate: 779792 minRate: 221000 AvgDt: 12.2627 changes: 7
ns3::FdashClient-Node: 2 InterruptionTime: 0 interruptions: 0 avgRate: 748132 minRate: 221000 AvgDt: 13.3227 changes: 5
ns3::FdashClient-Node: 3 InterruptionTime: 0 interruptions: 0 avgRate: 812568 minRate: 221000 AvgDt: 11.8593 changes: 7
ns3::FdashClient-Node: 4 InterruptionTime: 0 interruptions: 0 avgRate: 748132 minRate: 221000 AvgDt: 13.3163 changes: 5
ns3::FdashClient-Node: 5 InterruptionTime: 0 interruptions: 0 avgRate: 829170 minRate: 221000 AvgDt: 10.3903 changes: 7
ns3::FdashClient-Node: 6 InterruptionTime: 0 interruptions: 0 avgRate: 748132 minRate: 221000 AvgDt: 13.2176 changes: 5
ns3::FdashClient-Node: 7 InterruptionTime: 0 interruptions: 0 avgRate: 748132 minRate: 221000 AvgDt: 13.1865 changes: 5
ns3::FdashClient-Node: 8 InterruptionTime: 0 interruptions: 0 avgRate: 829170 minRate: 221000 AvgDt: 10.9821 changes: 7
ns3::FdashClient-Node: 9 InterruptionTime: 0 interruptions: 0 avgRate: 762916 minRate: 221000 AvgDt: 12.8802 changes: 7
```

修改後還是沒能改善。

• **What you learn**

這次的 project 主要為調整 MPEG-DASH 的演算法，大致能從修改演算法的過程，首先了解在 server 和 client 如何使用 DASH 傳輸影音資料，以及認識 FDASH 等演算法針對多變的網路環境，使用估計 buffer 狀態，或是 bit rate 去調整每一次傳輸的品質，以達到 client 獲得較好的 QoE。