# 基于Fov的全景视频缓存 ---- 论文学习汇报

CUC - 19 - 数媒技 - 杨雪婷

2021.01.22

# 汇报内容介绍

1. 整体介绍

2. 算法和模型分析

3. 评价指标&实验结果

4. 实验环境&实验数据

# Part One： 整体介绍

## (1) 目前困难：

- 减少宽带消耗
- 解决网络延迟

## (2) 现有解决方法：

现有**减少宽带消耗**的方法（FoV-adaptive 360°视频流）：

- 用户视角FOV内发送高分辨率的贴图
- 其他贴图发送低分辨率或不发送缓存请求

但是没有解决**网络延迟**

## （3）研究思路：

缓存**靠近**终端用户的**流行内容**(Caching popular content close to the end users)
-----> 同时解决网络延迟和宽带需求。

- 靠近：启用**基站**作为缓存模块来实现(BSs)位于RAN(无线接入网)边缘。
- 流行内容：预测下一次的Fov，**提高命中率**

## （4）研究结果

FoV-aware caching policy：

a probabilistic model based on previous users' watching histories

**作用：当缓存容量超过时，缓存模块利用这个策略来确定清除块的优先级。**

# Part Two： 算法和模式

# 整体系统

## 1. 网络模式

## 2. 360°视频模式

## 网络：

- 如果缓存有请求的内容，直接调用；
- 如果没有缓存，先从CDN上调用下载，然后再给客户端。（增加延迟 & 带宽消耗）

## 请求：

## 目标：

满足当前网络吞吐量的情况下，为用户实际的FoV内提供尽可能高的质量。

1. **网络通畅：Fov内的高质量缓存，其他低质量**

2. **网络阻塞：全部视频块都低质量缓存**

# 缓存策略的算法

## （1）得到缓存策略需要的参数

1. Users’ FoV Pattern

2. Impact of Video’s Bitrate on FoV’s Quality

# 实验前期准备

# 对于Users’ FoV Pattern -- 确定common-Fov

已知：用户的Fov和内容关系极大，赛车-->正前方，观光游览-->360°

实验：收集真实轨迹的头部跟踪数据集，并显示用户在虚拟环境有某些类似的观看模式

数据集：10短视频，每段50人观看，计算**FOV的热力图。**

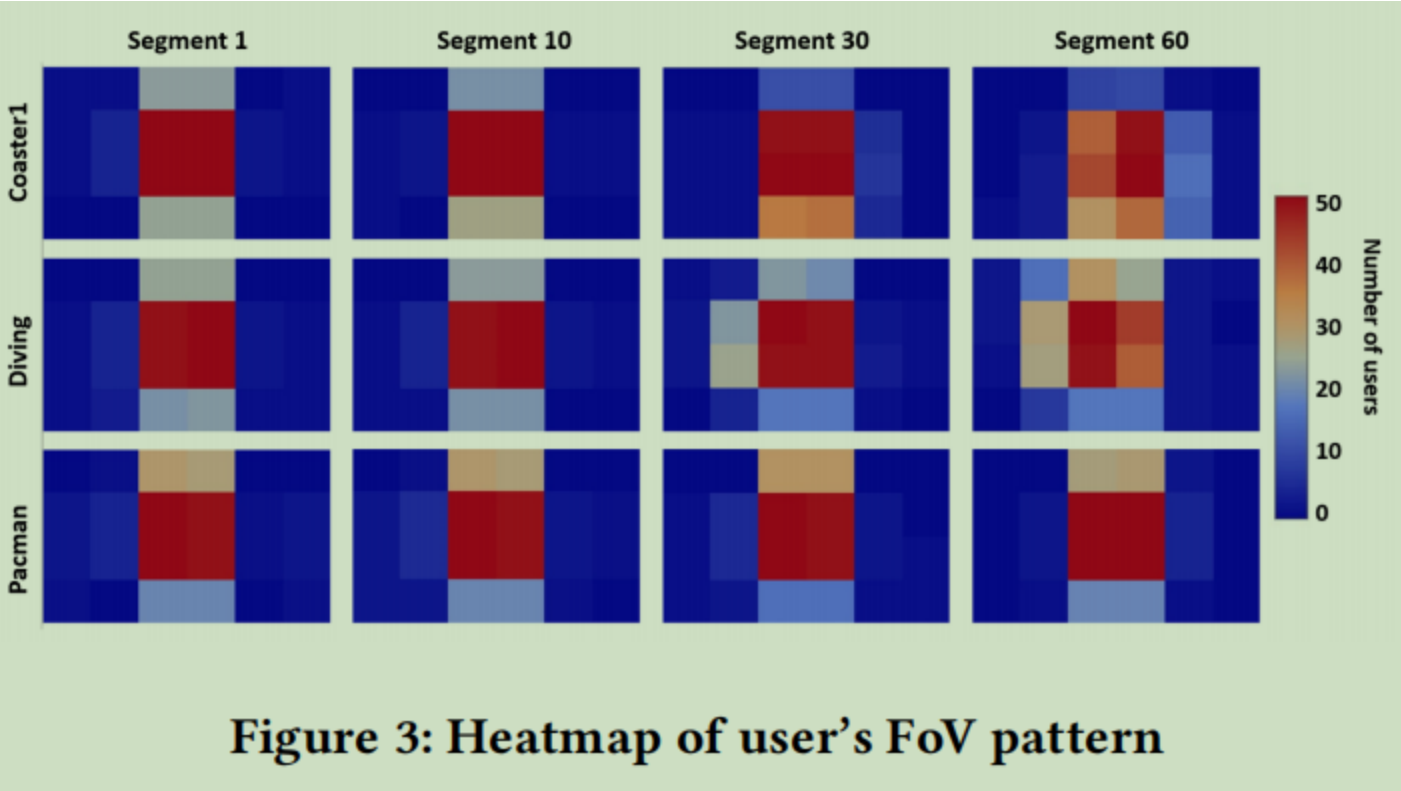
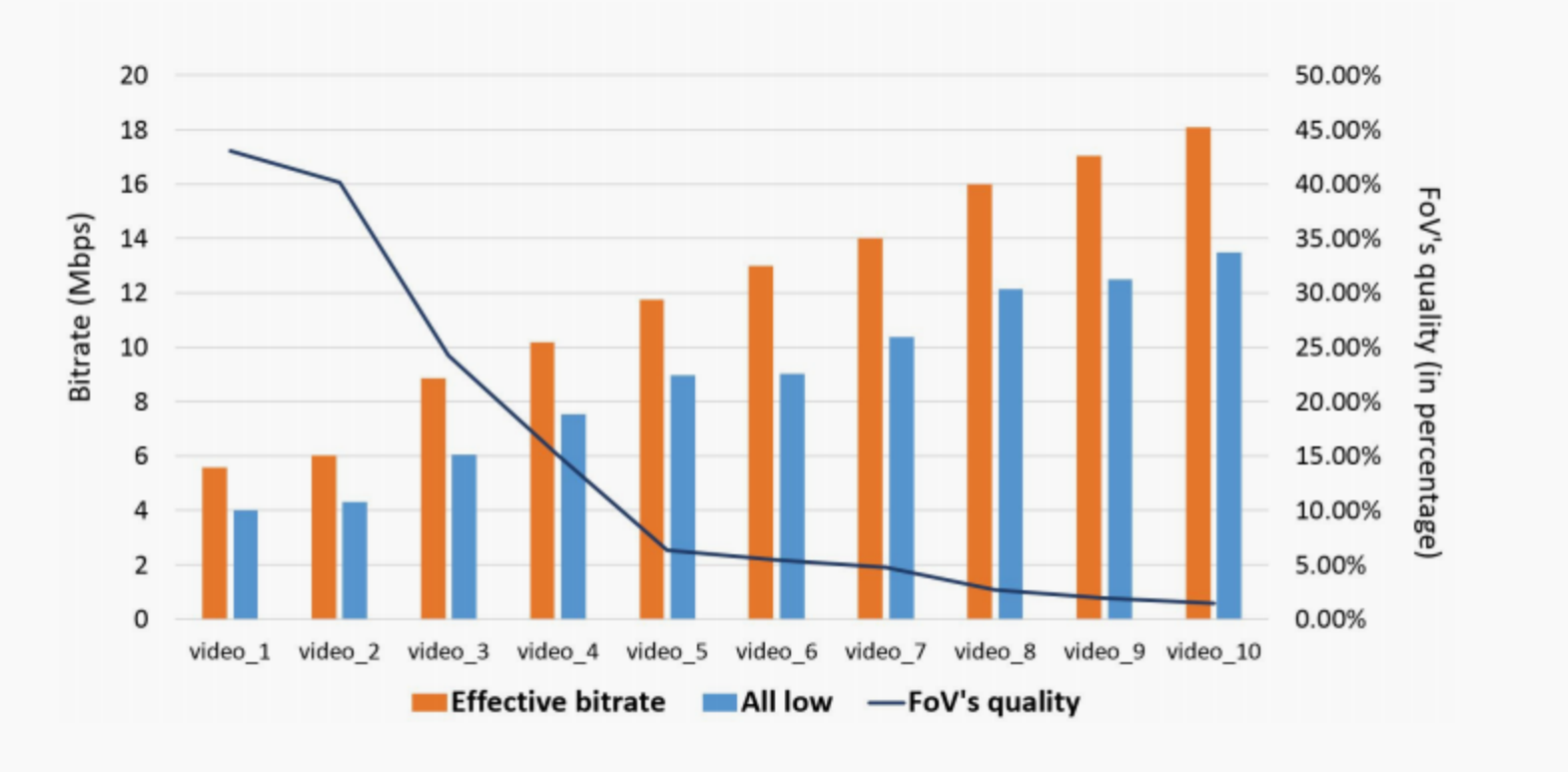



Figure 3: Heatmap of user’s FoV pattern

# 对于Bitrate to FoV’s Quality

已知：视频由比特率编码<--视频内容的纹理和运动,也就是说场景越复杂，需要更高的比特率来维持相同水平。



## (2) FoV-aware Caching Policy

***Learning Parameters***

- 参数Q_vsi : 视频块i在common-Fov中出现的概率

  训练数据：对于第i块的请求（信息为：第i块是否在请求的Fov中）

- 参数F_i：用户请求视频块i为高质量的概率

  训练数据：对于Fov中第i块的请求（信息为：是否需要高质量）

描述参数的模型的选择：

- 因为都分别**只有一个变量影响**，选择**朴素贝叶斯算法**来构造每个参数的概率模型。
- 最大似然估计（MLE），使得Q_vsi和F_i最接近数据集中两者的公式。

建立：

$$
\begin{aligned}
\theta_{v,s,i} &= \arg\max_{\theta} P(\mathcal{D}_{\theta_{v,s,i}} \mid \theta) \\
&= \arg\max_{\theta} ln(\theta_{v,s,i}^{\alpha_{inFoV_{v,s,i}}} * (1-\theta_{v,s,i})^{\alpha_{outOfFoV_{v,s,i}}}) \\
\psi_v &= \arg\max_{\theta} P(\mathcal{D}_{\psi_v} \mid \psi) \\
&= \arg\max_{\psi} ln(\psi_v^{\beta_{high_v}} * (1-\psi_v)^{\beta_{low_v}})
\end{aligned}
$$

为了使得上述方程最大化（Max），我们对方程求导：以第一个方程为例

$$\frac{d}{d\theta}\left[ln(\theta_{v,s,i}^{\alpha_{inFoV_{v,s,i}}} * (1-\theta_{v,s,i})^{\alpha_{outOfFoV_{v,s,i}}})\right] =$$

$$\frac{d}{d\theta}\left[\alpha_{inFoV_{v,s,i}} ln(\theta_{v,s,i}) + \alpha_{outOfFoV_{v,s,i}} ln(1-\theta_{v,s,i})\right] =$$

$$\alpha_{inFoV_{v,s,i}} \frac{d}{d\theta} ln(\theta_{v,s,i}) + \alpha_{outOfFoV_{v,s,i}} \frac{d}{d\theta} ln(1-\theta_{v,s,i}) =$$

$$\frac{\alpha_{inFoV_{v,s,i}}}{\theta_{v,s,i}} - \frac{\alpha_{outOfFoV_{v,s,i}}}{1-\theta_{v,s,i}} = 0$$

最终得到了两个参数的模型：

$$\theta_{v,s,i} = \frac{\alpha_{inFoV_{v,s,i}}}{\alpha_{inFoV_{v,s,i}} + \alpha_{outOfFoV_{v,s,i}}}$$

$$\psi_v = \frac{\beta_{high_v}}{\beta_{high_v} + \beta_{low_v}}$$

## （3）建立更新参数模型

学习更新Q_vsi和F_i得到R（第i视频块在将来被以q的质量请求的概率）

- 对于低质量版本的视频块有两种可能被请求：
  - i. 不在Fov内
  - ii. 在Fov内但是用户请求为低质量
- 对于高质量的视频块请求只有一种可能：在Fov内且高质量请求。

$$\gamma_{v,s,i,q} = \begin{cases} \theta_{v,s,i} * \psi_v, & \text{if } q = high \\ (1 - \theta_{v,s,i}) + \theta_{v,s,i} * (1 - \psi_v), & \text{if } q = low \end{cases}$$

因此我们可以通过以下的update_parameters函数得到R：

**Function** update_parameters($t_{v,s,i,q}$):
  **if** *is_in_FoV*($t_{v,s,i,q}$) **then**
    $\alpha_{\text{inFoV}_{v,s,i}}$ += 1
    **if** $q$ == *high* **then**
      $\beta_{\text{high}_v}$ += 1
    **else**
      $\beta_{\text{low}_v}$ += 1
    **end**
  **else**
    $\alpha_{\text{outOfFoV}_{v,s,i}}$ += 1
  **end**
  $\theta_{v,s,i} = \alpha_{\text{inFoV}_{v,s,i}} / (\alpha_{\text{inFoV}_{v,s,i}} + \alpha_{\text{outOfFoV}_{v,s,i}})$
  $\psi_v = \beta_{\text{high}_v} / (\beta_{\text{high}_v} + \beta_{\text{low}_v})$
  **if** $q$ == *high* **then**
    $\gamma_{v,s,i,q} = \theta_{v,s,i} * \psi_v$
  **else**
    $\gamma_{v,s,i,q} = (1 - \theta_{v,s,i}) + \theta_{v,s,i} * (1 - \psi_v)$
  **end**
  return $\gamma_{v,s,i,q}$

# （4）建立 *Caching Policy*

目标：描述缓存模块在接收到质量为q的视频块i的请求时所采取的动作。

```
Function upon_request(t_{v,s,i,q}):
    γ_{v,s,i,q} ← update_parameters(t_{v,s,i,q})
    if cache.contains(t_{v,s,i,q}) then
        tile ← cache.get(t_{v,s,i,q})
        send(tile)
    else
        // downloading tile from remote content server
        tile ← download(t_{v,s,i,q})
        // adding tile to cache
        cache.add(tile, key=γ_{v,s,i,q})
        // sending tile to client
        send(tile)
        while cache.size() > cache_capacity do
            cache.remove_min()
        end
    end
```

# Part Three 评价指标&实验结果

**我们将我们的研究成果和现有的3种做对比：**

- *end-to-end*: 客户端和服务器间无缓存（炮灰）
- *LFU*: 当缓存已满时，访问频率最低的贴图将被从缓存中删除
- *LRU*: 我们删除从最后一次访问以来持续时间最长的贴图

## 验证策略优良的指标：

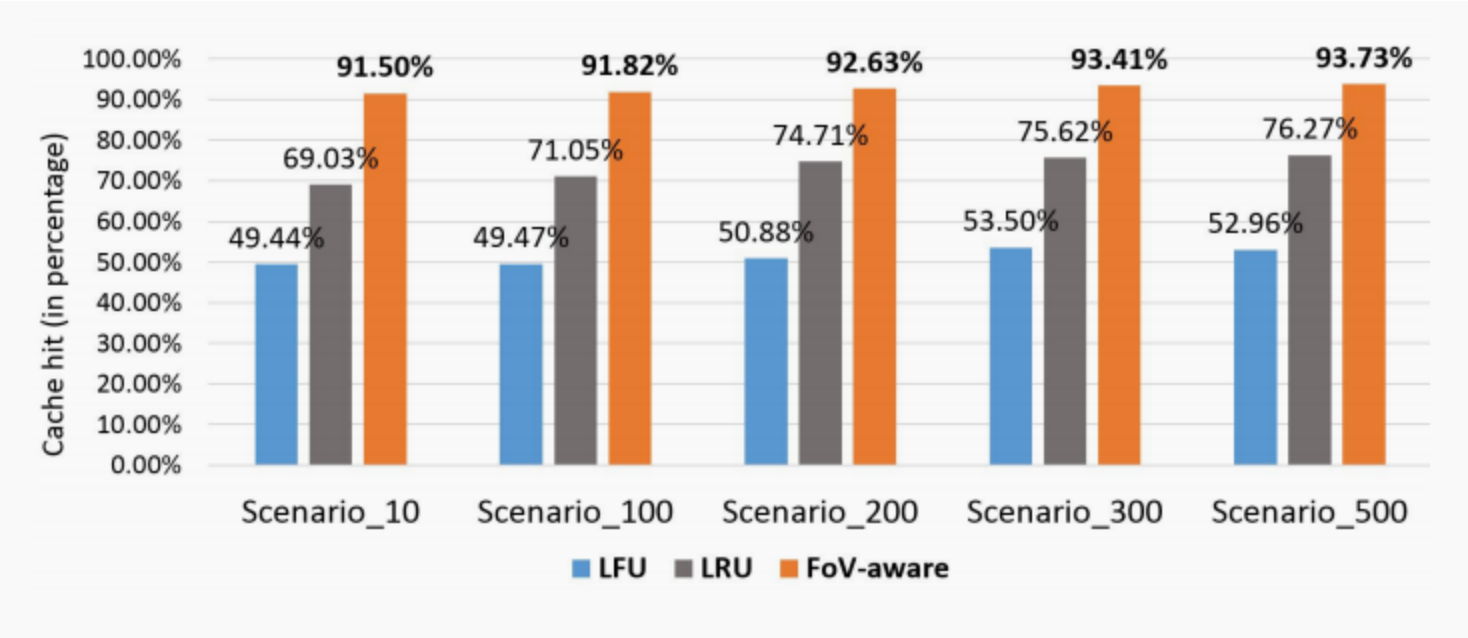
- *Tile hit* (缓存命中率):请求的贴图在缓存中找到的比例。
- *Bandwidth saving*[带宽节省(以百分比表示)]:当客户端请求命中缓存时，所节省的带宽超过所有请求的总带宽消耗。
- *Reb*(缓冲频率)：没有任何视频可以播放了
- *DoR* [延迟时间(以秒为单位)]:该指标显示客户端经历延迟的总持续时间。
- Qua[高质量的FoV(以百分比表示)]:客户端接收瓷砖内FoV高质量超过360视频的总段数的片段数。

## (1) Caching Policy Performance

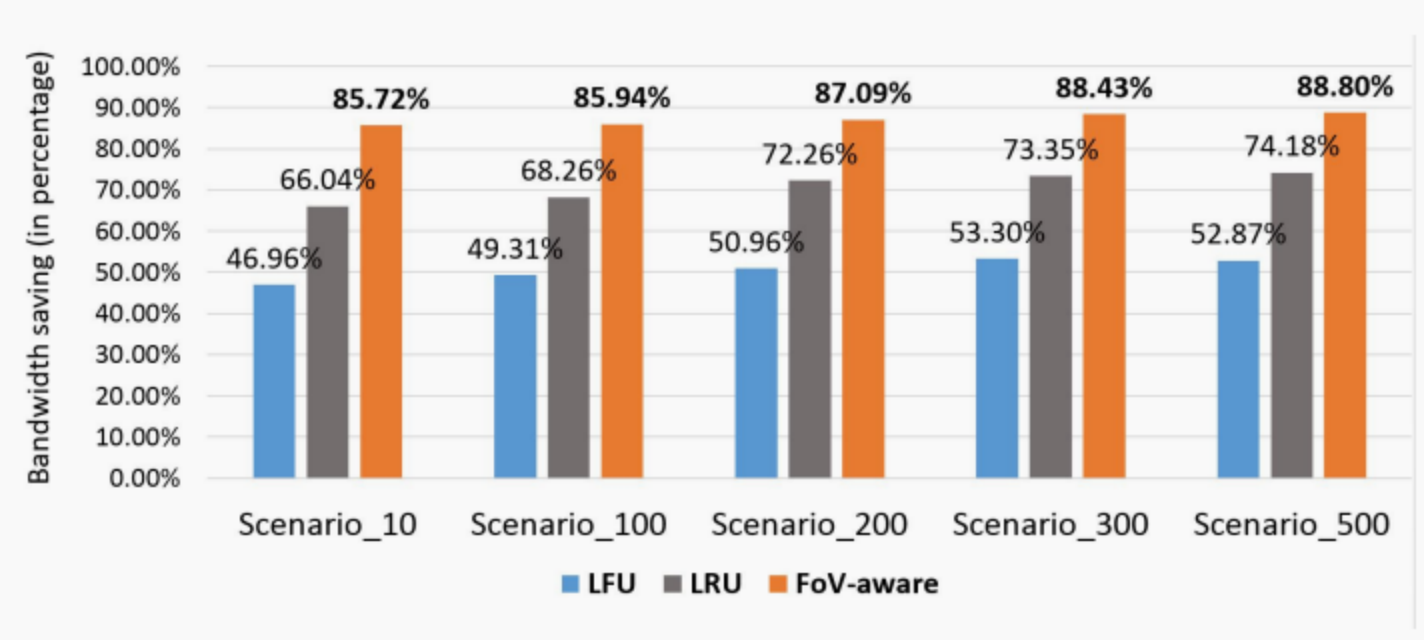
性能指标：*Tile hit + Bandwidth saving*

- 较高的缓存命中率减少了对内容服务器的请求数量，从而节省了核心网络中的带宽。

Tile hit:


Cache hit (in percentage)
100.00%
90.00%
80.00%
70.00%
60.00%
50.00%
40.00%
30.00%
20.00%
10.00%
0.00%
49.44%
69.03%
91.50%
49.47%
71.05%
91.82%
50.88%
74.71%
92.63%
53.50%
75.62%
93.41%
52.96%
76.27%
93.73%
Scenario_10
Scenario_100
Scenario_200
Scenario_300
Scenario_500
LFU
LRU
FoV-aware

# Bandwidth saving:



## 结论：

**最好的缓存策略：删除将来被请求的可能性最小的块。**

FoV-aware缓存策略利用常见的fov现象(即，观看同一视频的大多数用户的fov有显著的重叠)来驱动对未来请求贴图的更好预测。

# (2) Streaming Performance

性能指标：Reb，Dor，Qua

**Table 2: Number, duration of rebuffering events and high quality for FoV in End to End**

| Scenarios | 10 | | | 100 | | | 200 | | | 300 | | | 500 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Reb | DoR | Qua | Reb | DoR | Qua | Reb | DoR | Qua | Reb | DoR | Qua | Reb | DoR | Qua |
| End to End | 20.84% | 5.75 Sec | 20.65% | 22.85% | 6.17 Sec | 11.08% | 25.84% | 7.02 Sec | 3.12% | 26.69 % | 8.16 Sec | 0.31% | 40.19% | 10.92 Sec | 0.0% |

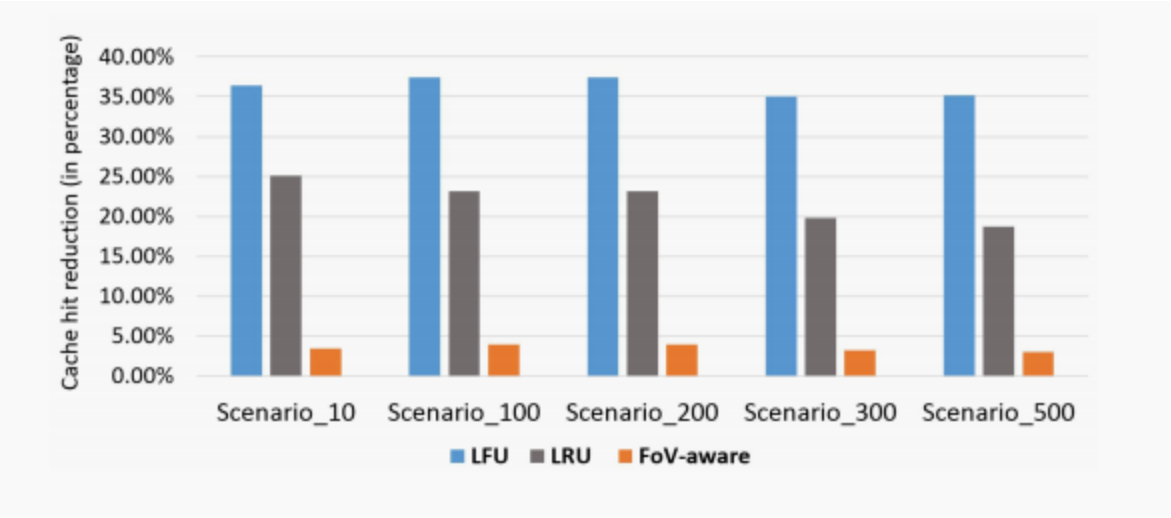
**Table 3: Number, duration of rebuffering events and high quality for FoV in FoV-aware, LFU and LRU - All scenarios - Cache capacity: case_1**

| Scenarios | 10 | | | 100 | | | 200 | | | 300 | | | 500 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Reb | DoR | Qua | Reb | DoR | Qua | Reb | DoR | Qua | Reb | DoR | Qua | Reb | DoR | Qua |
| FoV-aware | **20.20%** | **5.71** Sec | **22.08%** | **20.36%** | **5.73** Sec | **21.45%** | **20.78%** | **5.78** Sec | **19.17%** | **20.92%** | **5.80** Sec | **16.32%** | **22.50%** | **6.10** Sec | **15.18%** |
| LFU | 20.71% | 5.75 Sec | 19.24% | 22.34% | 6.03 Sec | 14.83% | 23.86% | 6.47 Sec | 10.88% | 25.53% | 6.94 Sec | 8.84% | 31.75% | 8.46 Sec | 7.50% |
| LRU | 20.69% | 5.72 Sec | 21.49% | 21.16% | 5.79 Sec | 18.77% | 21.61% | 5.87 Sec | 14.52% | 21.76% | 6.07 Sec | 11.61% | 24.81% | 6.69 Sec | 10.23% |

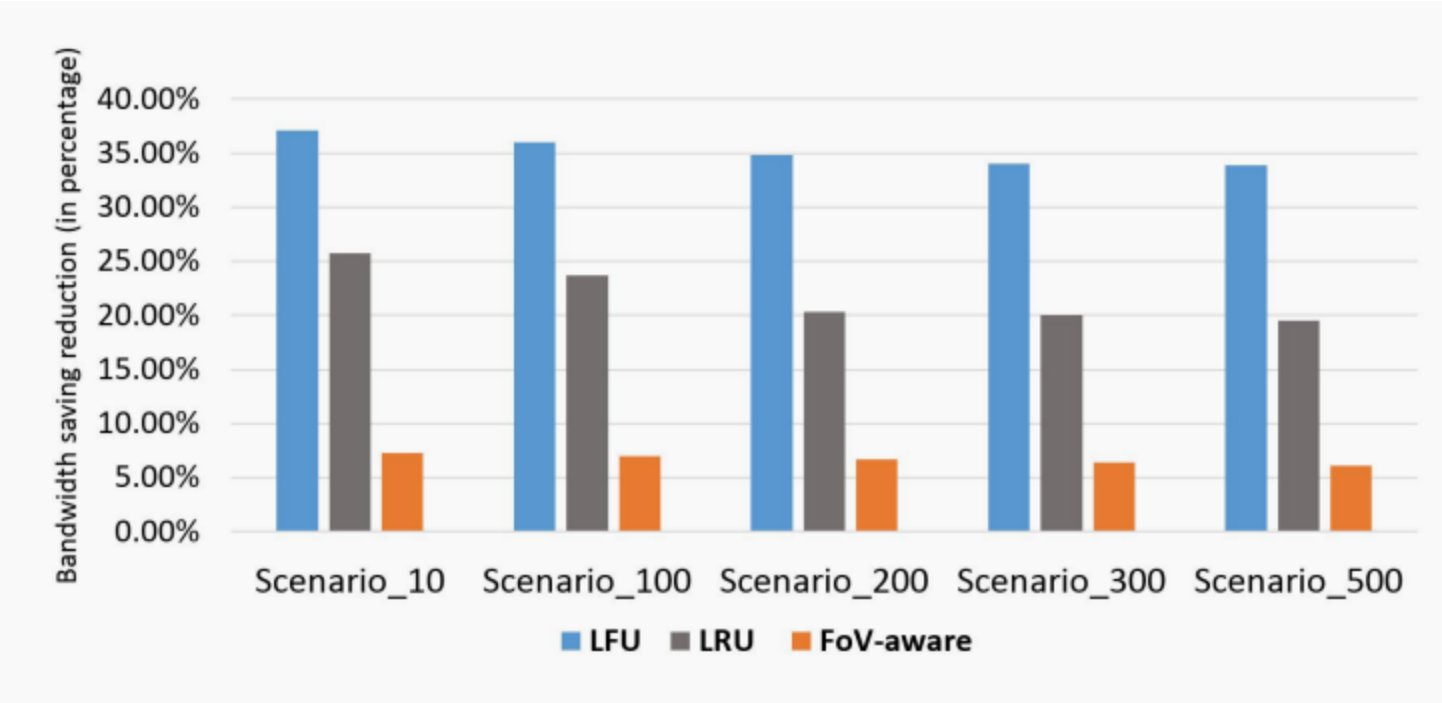
# （3） Impact of Cache Capacity

研究缓存容量对所有三种缓存策略的影响，我们让缓存容量都变小。

对于命中率，Fov - aware的策略比另外两种降低的都小



同时对于Bandwidth saving来说，也是Fov策略性能下降的更小。



# Part Four 实验环境和数据集

# 实验环境：`Mahimahi`

实验搭建环境：Ubuntu（版本14.04或更高版本）

开源可下载！下载网站

功能介绍：

### 记录并重播HTTP流量

使用Mahimahi的RecordShell，您可以完全记录真实的HTTP内容（例如网页）并将其存储到磁盘。然后可以使用ReplayShell重复播放记录的站点。ReplayShell通过本地镜像页面的服务器分布来准确地建模Web应用程序的多服务器结构。

### 模拟众多网络条件

Mahimahi的网络仿真工具可用于仿真许多不同的链接条件，以重播记录的HTTP流量。Mahimahi支持仿真固定传播延迟（DelayShell），固定和可变链接速率（LinkShell），随机数据包丢失（LossShell）。LinkShell还支持各种排队规则，例如DropTail，DropHead和活动队列管理方案（例如CoDel）。Mahimahi的每种网络仿真工具都可以任意地嵌套在一起，从而提供了更多的实验灵活性。

### 使用未修改的客户端应用程序

可以使用商业浏览器（例如Google Chrome，Mozilla Firefox和Safari）来记录和重播网站。此外，其他客户端应用程序（包括虚拟机和移动设备仿真器）可以在Mahimahi的每个工具中未经修改地运行。

# 实验中模拟的场景：

**Table 1: latency and packet loss**

| Scenario | latency | Packet Loss |
|---|---|---|
| Scenario_500 | 500 ms | 0.05% |
| Scenario_300 | 300 ms | 0.05% |
| Scenario_200 | 200 ms | 0.04% |
| Scenario_100 | 100 ms | 0.03% |
| Scenario_10 | 10 ms | 0.01% |

## 环境：

- Mininet
- Mahimahi
- NS-2
- NS-3
- Emulab

我们鼓励学生在他们的实验平台上使用MiniNet或Mahimahi仿真系统。MiniNet最适合多节点拓扑，而Mahimahi在修改和测试单个链路上运行的拥塞控制协议时很好。虽然我们通常更喜欢学生使用模拟器，因为模拟器具有更真实的网络特性，例如给定节点拓扑的实时流量处理，例如NS-3。我们为所有学生提供亚马逊网络服务（AWS）计算云（EC2）上的计算资源。

## 使用评价：

可以使用mahimahi仿真器作为传输层的仿真模拟，

mahimahi很多时候也被用在TCP的拥塞控制场景里，它能够模拟中间瓶颈链路的Buffer变化情况，从而反映带宽和RTT的变化。

## 数据集：

源自论文：《360° Video Viewing Dataset in Head-Mounted Virtual Reality》

此数据集涵盖各种各样的360◦视频与不同的内容(所有视频都可以在YouTube上获得)。这个数据集包括50个客户端观看每一个的FoV轨迹。我们总共有500条观看痕迹。

## 视频分段：

使用`GPAC MP4Box`工具将每个视频分成1秒的片段。

这样一个片段的每个平铺都可以单独使用。

最终所有贴图：高质量26.3Mbps,低质量8.7Mbps