DTMCash: explorando redundância de *viewports* de usuários para otimização de *streaming* de vídeo 360º

Gustavo Dias[1], Luciano de S. Fraga[1], Kleber V. Cardoso[1], Sand L. Correa[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)

{gustavodias, kleber, sandluz}@ufg.br

lucianofraga@inf.ufg.br

Abstract. *The advancement of immersive technologies, such as Augmented Reality (AR) and Virtual Reality (VR), has brought significant challenges to the transmission of 360° videos, with increasing bandwidth requirements and low latency due to the size of frames. Our proposal, DTMCash, stands out in addressing these challenges by adopting dynamic tiles and combining user viewports, effectively tackling the issue of transmission in multi-user scenarios. Compared to a state-of-the-art solution, our approach reduces the aggregate bandwidth consumption of the Internet link by at least 48.2%, while maintaining the same consumption on the wireless link, also providing greater efficiency in cache usage.*

Resumo. *O avanço das tecnologias imersivas, como Realidade Aumentada (AR) e Realidade Virtual (VR), trouxe desafios significativos na transmissão de vídeos em 360°, com requisitos crescentes de largura de banda e baixa latência devido ao tamanho dos quadros. Nossa proposta, o DTMCash, destaca-se ao enfrentar esses desafios por meio da adoção de tiles dinâmicos e da combinação dos viewports dos usuários, abordando eficazmente a questão da transmissão em cenários multi-usuário. Comparado a uma solução estado-da-arte, nossa solução reduz o consumo de largura de banda agregada do enlace de Internet em pelo menos 48.2%, enquanto mantém o mesmo consumo no enlace sem fio, proporcionando também maior economia no uso de cache.*

1. Introdução

Nos últimos anos, avanços em tecnologias de realidade aumentada (*augmented reality* - AR) e realidade virtual (*virtual reality* - VR) introduziram uma nova era de experiências imersivas, redefinindo a maneira como o usuário interage com o conteúdo digital. Atualmente, tecnologias de AR e VR estão sendo adotadas em diversos domínios de aplicação, incluindo educação, saúde e entretenimento [Wu et al. 2023, Cao et al. 2023]. Esse movimento é motivado, em grande parte, pelo surgimento de diversos dispositivos de AR/VR no mercado, denominados *head-mounted displays* (HMDs), dotados com grande capacidade de sensoriamento e poder de processamento.

Um dos formatos de vídeo mais utilizados atualmente em cenários imersivos é o vídeo 360º, também conhecido como panorâmico ou omnidirecional [van der Hooft et al. 2023]. Esses vídeos são normalmente capturados por câmeras omnidirecionais ou gerados por múltiplas câmeras, cujas visualizações individuais são unidas e combinadas em uma única esfera. Cada cena esférica de um quadro do vídeo é então mapeada

para outras formas geométricas para aproveitar as vantagens de compressão oferecidas por *codecs* avançados de vídeo tradicionais (2D). Um dos métodos de mapeamento mais adotados para este propósito é a projeção equirretangular (*equirectangular projection* - ERP) [Yaqoob et al. 2020], onde as linhas longitudinais e latitudinais são mapeadas para linhas retas. Uma consequência do uso do ERP, no entanto, é que as áreas próximas dos polos se tornam altamente distorcidas, reduzindo a probabilidade de visualização dessas áreas. Após a fase de mapeamento, o vídeo é comprimido para reduzir a largura de banda durante a transmissão, a qual pode ocorrer, por exemplo, através de um serviço de vídeo sob demanda.

De fato, desde 2015, Youtube e Facebook oferecem vídeos sob demanda em formato 360º. No entanto, a entrega massiva e com qualidade desse tipo de conteúdo sobre uma infraestrutura de rede sem fio ainda é um desafio. Particularmente, vídeos 360º exigem alta largura de banda de rede, uma vez que os quadros neste tipo de vídeo devem ser 4 a 6 vezes maiores que os de vídeos tradicionais para atingirem a mesma qualidade percebida pelo usuário [Qian et al. 2018]. Por serem maiores, o processamento de tais quadros requer também maior poder computacional nos HMDs e, consequentemente, maior consumo de bateria nesses dispositivos. Por fim, a transmissão de vídeos 360º exige requisitos estritos de latência (até 25 ms) para evitar enjoo cibernético [Liu et al. 2018].

Para mitigar os problemas envolvidos na transmissão sem fio de vídeos 360º, os principais trabalhos da literatura combinam o conceito de transmissão ciente de *viewport* (*viewport-aware streaming*) com o cache na borda da rede [Lo et al. 2018, Papaioannou and Koutsopoulos 2019, Hsu 2020]. Na transmissão ciente de *viewport*, cada quadro do vídeo é dividido em *tiles* de tamanho fixo (i.e., *tiles* com tamanho e localização fixos no quadro) e apenas os *tiles* dentro da janela de visualização do usuário (*viewport* - VP) são transmitidos em alta resolução. Essa abordagem reduz a largura de banda necessária para a transmissão dos quadros do vídeo, ao mesmo tempo que aumenta a qualidade percebida pelo usuário. Por outro lado, caches localizadas na borda da rede permitem que vídeos sejam armazenados perto dos usuários que irão consumí-los, reduzindo significativamente o número de solicitações enviadas para servidores de conteúdo localizados na nuvem e, consequentemente, o atraso necessário para a transmissão dos vídeos [Yao et al. 2019]. Trabalhos que combinam os dois conceitos (transmissão ciente de VP e cache na borda) buscam atender os VPs requisitados com *tiles* armazenados na cache e com a maior resolução possível, de forma a reduzir o tempo de entrega do vídeo. Neste caso, a comparação entre os *tiles* necessários para cobrir o VP requisitado e os *tiles* na cache é simples, uma vez que um acerto (*cache hit*) ocorre apenas se ambos foram idênticos em termos de localização e tamanho.

Recentemente, foi mostrado que a transmissão ciente de VP atinge melhores resultados quando usada com *tiles* dinâmicos, ou seja, *tiles* com tamanho e localização diferentes no quadro [Ozcinar et al. 2019, Madarasingha and Thilakarathna 2021]. Isso ocorre porque o uso de *tiles* de tamanho fixo dificulta a cobertura precisa do VP do usuário. Como consequência, nesta abordagem, normalmente são utilizados mais *pixels* que os necessários para cobrir o VP requisitado (também chamado de *pixels* redundantes), aumentando, de forma desnecessária, a quantidade de dados a serem transmitidos. Outro problema decorrente do uso de *tiles* de tamanho fixo é que as áreas distorcidas do VP re-

quisitado (geradas pelo ERP) são transmitidas com a mesma taxa de bits e com o mesmo tamanho de *tiles* das áreas não distorcidas, reduzindo as oportunidades de compressão de dados. *Tiles* dinâmicos, ao contrário, permitem uma cobertura mais flexível e precisa do VP do usuário. De fato, os resultados em [Madarasingha and Thilakarathna 2021] mostram que o uso de *tiles* dinâmicos na transmissão ciente de VP reduz em até 31% a quantidade de *pixels* a ser comprimida e em até 35% a largura de banda de rede consumida. No entanto, tal solução exige novos mecanismos para o cache de vídeos 360º. Isso ocorre porque a localização e o tamanho dos *tiles* nessa solução são altamente variáveis e, portanto, um casamento exato entre *tiles* requisitados e em cache é altamente improvável, resultando em uma baixa taxa de acerto e no redirecionamento de muitas requisições para o servidor de conteúdo, localizado na nuvem.

Até o momento, até onde sabemos, apenas um trabalho, denominado OpCASH [Madarasingha et al. 2022], propõe um mecanismo de cache para transmissão ciente de VP utilizando *tiles* dinâmicos. Neste contexto, neste trabalho, buscamos avançar o estado-da-arte propondo o ***D****ynamic* ***T****ile and* ***M****ulti-user Edge* ***Ca****che Utilization for 360-Degree vídeo* ***S(H)****treaming* (DTMCash). Até onde sabemos, nossa proposta é a primeira a considerar um cenário multi-usuário para o problema de mecanismos de cache para transmissão ciente de VP utilizando *tiles* dinâmicos. Comparado com duas versões adaptadas do OpCASH para o cenário multi-usuário, nossa solução reduz a largura de banda agregada do enlace de Internet em 49.7% e 48.2% nas versões 1 e 2 respectivamente, alcançando um ganho de 65.36% no enlace sem fio em comparação com a versão 2 e tendo uma perda de apenas 0.89% ao compararmos com a versão 1. Além disso, em todos experimentos realizados, o DTMCash promove uma ocupação mais eficiente da cache.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 apresenta fundamentos sobre *tiles* dinâmicos e trabalhos relacionados. A Seção 3 descreve o modelo do sistema, enquanto a Seção 4 descreve o funcionamento do DTMCash. Os resultados são apresentados na Seção 5. Finalmente, a Seção 6 apresenta as conclusões e trabalhos futuros.

2. Fundamentos e Trabalhos relacionados

O objetivo principal da transmissão ciente de VP é reduzir a quantidade de dados enviados pela rede e aprimorar a qualidade do conteúdo transmitido, alinhando-se mais precisamente ao VP do usuário. A Figura 1 mostra uma comparação entre um esquema de *tiles* de tamanho fixo (lado direito) e um esquema de *tiles* dinâmicos (lado esquerdo), ambos cobrindo regiões simétricas de um VP. Como mostrado na figura, uma limitação significativa dos *tiles* de tamanho fixo é que eles não conseguem proporcionar uma cobertura mais fina (precisa) para o campo de visão do usuário. Como consequência, a transmissão ciente de VP utilizando *tiles* de tamanho fixo gera uma quantidade considerável de *pixels* desnecessários (i.e., *pixels* fora do VP do usuário). Adicionalmente, em alguns casos, apenas um pequeno pedaço do *tile* está na área do VP do usuário. No lado esquerda da Figura 1, ao contrário, o VP é dividido em *tiles* de tamanho muito pequenos, denominados *tiles* básicos. Os *tiles* básicos são então combinados em áreas retangulares maiores, de tamanho variado, formando os *tiles* dinâmicos. Por terem tamanhos flexíveis (variados), os *tiles* dinâmicos conseguem cobrir o VP do usuário de maneira mais precisa. Uma vez que um esquema de *tiles* dinâmicos é proposto para um dado VP, ele pode ser mantido para

todos os quadros do mesmo *chunk* de vídeo. Isso significa que os *tiles* dinâmicos têm a mesma duração e a mesma quantidade de quadros que o *chunk* ao qual ele pertence. Esses *tiles* são então transmitidos do servidor de conteúdo para o usuário em *chunks* menores e de forma separada.

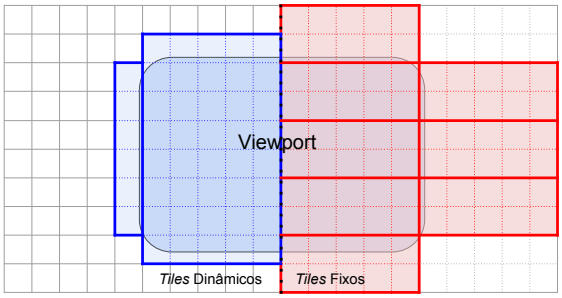

Figura 1. Formação de *tiles* dinâmicos a partir de *tiles* básicos (lado esquerdo) e comparação de cobertura de VP usando *tiles* de tamanho fixo (lado direito) e variado (lado esquerdo).

O conceito de *tile* dinâmico foi proposto em [Li et al. 2016], onde os autores dividem o quadro de um vídeo 360º em um número fixo de *tiles*, variando seu tamanho de acordo com a latitude. Posteriormente, os autores em [Zhou et al. 2018, Ozcinar et al. 2019] aperfeiçoaram o conceito, propondo a criação de *tiles* dinâmicos a partir da combinação de *tiles* básicos. Em [Madarasingha and Thilakarathna 2021], os autores propõem um mecanismo de geração de *tiles* dinâmicos para vídeos 360º denominado Vastile, onde um algoritmo de cobertura mínima sem sobreposição é utilizado para gerar um esquema de *tiles* dinâmicos a partir da combinação de *tiles* básicos. Para isso, o algoritmo considera um compromisso entre o tamanho dos *tiles* básicos, a quantidade deles presente em um único quadro e a minimização dos *pixels* desnecessários ao sobrepor os quadros do vídeo. Os autores demonstram que o Vastile pode prover esquemas de *tiles* dinâmicos para cobrir o VP do usuário em menos de 1 segundo. Adicionalmente, eles demonstram também que a transmissão ciente de VP utilizando *tiles* gerados pelo Vastile pode diminuir significativamente a quantidade de *pixels* desnecessários antes da compactação, reduzindo a largura de banda consumida durante a transmissão do vídeo.

Os impactos do armazenamento de *tiles* dinâmicos na cache foram estudados em [Madarasingha et al. 2022]. Para um mesmo vídeo e assumindo uma cache com tamanho infinito, os autores demonstram que, em geral, apesar de não serem idênticos, existe uma sobreposição razoável (em termos de *tiles* básicos) entre um *tile* dinâmico requisitado para cobrir um VP e algum *tile* dinâmico armazenado na cache. Apesar de tal sobreposição, técnicas de armazenamento em cache baseadas em *tiles* de tamanho fixo, quando usadas em cenários com *tiles* dinâmicos, resultam em baixa taxa de acerto, pois tais técnicas consideram ”hit” apenas se o *tile* requisitado é idêntico a algum *tile* em cache. Os autores então propõem o OpCASH, um mecanismo para aumentar a utilização da cache encontrando *tiles* dinâmicos em cache que não são exatamente iguais (em termos de tamanho e localização) aos requisitados, mas estão localizados próximos e apresentam tamanho quase similares. Para alcançar este objetivo, o OpCASH formula um Problema de Programação Linear Inteira (*Integer Linear Programming* - ILP) que busca atingir três objetivos: selecionar *tiles* da cache sempre que possível; reduzir a quantidade de *pixels*

desnecessários ao cobrir um VP com *tiles* da cache; e reduzir a quantidade de dados transmitida entre a cache e o servidor de conteúdo (ou seja, pelo enlace de Internet).

Uma limitação do OpCASH, no entanto, é que o mecanismo resolve esse problema considerando apenas uma requisição por vez. Ao contrário, o foco principal do DTM-Cash é atingir os três objetivos acima em um cenário de *tiles* dinâmicos onde múltiplos usuários fazem requisição para a cache ao mesmo tempo. Além de mais realista para um cenário multi-usuário, nossa proposta permite explorar regiões de intersecção entre VPs de usuários diferentes para otimizar a entrega de videos 360º, melhorando também a ocupação da cache.

3. Modelo do Sistema

Neste trabalho, consideramos um cenário onde múltiplos usuários consomem vídeos 360º de um serviço de vídeo sob demanda, como ilustrado na Figura 2. No restante deste trabalho, a menos que explicitamente especificado o contrário, usamos o termo *tile* para referirmos a *tiles* dinâmicos.

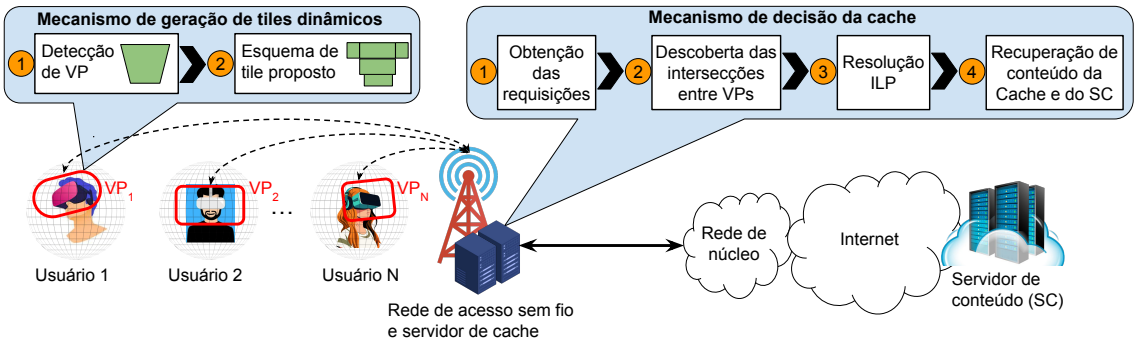

Figura 2. Múltiplos usuários requisitam *tiles* dinâmicos para o servidor de borda que hospeda a cache.

Consideramos um catálogo de vídeos 360º hospedado em um servidor de conteúdo localizado na nuvem. Assumimos que esse servidor é capaz de gerar *tiles* desses vídeos dentro do intervalo de tempo da entrega do próximo *chunk*. Denotamos por $\mathcal{DT} = \{dt_1, dt_2, \ldots, dt_{|\mathcal{DT}|}\}$ o conjunto de todos os *tiles* possíveis de serem gerados. Adicionalmente, denotamos por $\mathcal{BT} = \{bt_1, bt_2, \ldots, bt_{|\mathcal{BT}|}\}$ o conjunto de *tiles* básicos a partir dos quais os *tiles* (dinâmicos) podem ser gerados. Usando o acesso à Internet, o servidor de conteúdo pode atender as requisições de uma cache, co-localizada com uma estação 5G. Devido à capacidade limitada de armazenamento, em um determinado instante do tempo, a cache possui apenas um subconjunto dos *tiles* armazenados no servidor de conteúdo. Representamos esse subconjunto por $\mathcal{C} = \{dtc_1, dtc_2, \ldots, dtc_{|\mathcal{C}|}\}$.

Seja $\mathcal{U} = \{u_1, u_2, \ldots, u_{|\mathcal{U}|}\}$ o conjunto de usuários sob a cobertura da estação 5G. Como mostrado na Figura 2, esses usuários consomem conteúdo 360º do serviço de vídeo sob demanda usando um HMD que se conecta à rede 5G. No lado do usuário, assumimos que cada HMD executa um mecanismo (e.g., Vastile) capaz de particionar o VP do usuário em *tiles*. Seja $\mathcal{VP}_i = \{dtu_{i,1}, dtu_{i,2}, \ldots, dtu_{i,|\mathcal{VP}_i|}\}$ a configuração de *tile* proposta para cobrir o VP do usuário $u_i \in \mathcal{U}$, onde $dtu_{i,j} \in \mathcal{DT}$, $\forall j = 1, \ldots, |\mathcal{VP}_i|$. Uma vez que o HMD propõe tal configuração, ele requisita os *tiles* necessários para a cache usando o enlace 5G.

No lado da cache, múltiplas requisições de usuários, representadas por $\mathcal{VP} = \{\mathcal{VP}_1, \mathcal{VP}_2, \ldots, \mathcal{VP}_{|\mathcal{VP}|}\}$, podem chegar de forma concomitante. Neste contexto, alguns VPs requisitados podem conter regiões de intersecção. Dois VPs $\mathcal{VP}_i$ e $\mathcal{VP}_j$ possuem intersecção se ambos possuem um ou mais *tiles* básicos em comum, ou seja: $\mathcal{VP}_i \cap \mathcal{VP}_j \neq \emptyset \Leftrightarrow \exists\, bt_k \in \mathcal{BT} \wedge\ \exists\, dtu_{i,p} \in \mathcal{VP}_i\ \wedge\ \exists\, dtu_{j,r} \in \mathcal{VP}_j : I(bt_k, dtu_{i,p}) = I(bt_k, dtu_{j,r}) = 1$, onde I é uma função indicativa que retorna 1 caso um *tile* básico compõe um *tile* (dinâmico) e 0 caso contrário.

O objetivo do mecanismo de cache é atender o conjunto de requisições tentando encontrar, sempre que possível, *tiles* em cache que podem não ser exatamente idênticos aos *tiles* requisitados mas que contenham localização e tamanho próximos aos esperados. Ao mesmo tempo, é importante minimizar a quantidade de *pixels* redundantes. Por fim, é necessário também minimizar a quantidade de dados transferida entre a cache e o servidor de conteúdo. Para esse último objetivo, é importante explorar redundâncias (intersecções) de VPs para assegurar que os *tiles* que não estão em cache serão trazidos uma única vez do servidor de conteúdo.

4. DTMCash

Para atingir o primeiro objetivo descrito na Seção 3, é essencial que, ao receber um conjunto de requisições $\mathcal{VP}$, para cada $\mathcal{VP}_i \in \mathcal{VP}$, o mecanismo de cache busque localmente por *tiles* que possam cobrir adequadamente a área do VP requisitado. A adequação da área de cobertura de um *tile* $dtc_k \in \mathcal{C}$ em relação a um *tile* requisitado $dtu_{i,j} \in \mathcal{VP}_i$ pode ser calculada como:

$$c_k^r = \frac{f(dtc_k \cap dtu_{i,j})}{f(dtc_k)}, \tag{1}$$

onde a função f retorna a quantidade de *pixels* de um *tile* fornecido como entrada. De fato, c_k^r corresponde a um valor entre 0 e 1. Valores próximos de 0 significam que o *tile* da cache não provê uma cobertura adequada para o *tile* requisitado, seja porque a área de sobreposição é pequena, seja porque o *tile* da cache gera muitos *pixels* redundantes. Ao contrário, valores próximos de 1 implicam em uma cobertura adequada, indicando alta sobreposição com o *tile* requisitado, com poucos *pixels* redundantes.

Por outro lado, a sobreposição de um *tile* $dtc_k \in \mathcal{C}$ sobre um *tile* requisitado $dtu_{i,j} \in \mathcal{VP}_i$ pode gerar uma área residual, ou seja, uma área que não foi coberta pelo *tile* da cache e, portanto, deverá ser coberta com *tiles* do servidor de conteúdo. Ao buscar *tiles* na cache para cobrir um *tile* requisitado, é importante selecionar *tiles* cuja sobreposição gera uma área residual pequena. Isso reduz a quantidade de dados transmitida entre a cache e o servidor de conteúdo (i.e., o terceiro objetivo descrito na Seção 3). O tamanho da área residual gerada por um *tile* $dtc_k \in \mathcal{C}$ sobre um *tile* requisitado $dtu_{i,j} \in \mathcal{VP}_i$ pode ser calculada como:

$$c_k^s = f(dtu_{i,j} - (dtc_k \cap dtu_{i,j})). \tag{2}$$

Percebemos intuitivamente que o tamanho do *tile* $dtc_k \in \mathcal{C}$, denotado por c_k^e e calculado de acordo com a Equação 3, tem relação direta com a sobreposição com o *tile* requisitado e também com a área residual gerada. Tipicamente, quanto maior for c_k^e, maior será a sobreposição, o que é desejável, uma vez que um *tile* em cache com

pouca sobreposição gerará uma área residual maior. Contudo, *tiles* muito grandes (c_k^e altos) tendem a cobrir grandes regiões fora do *tile* requisitado, aumentando a quantidade de *pixels* redundantes. De acordo com o segundo objetivo descrito na Seção 3, esses *tiles* grandes devem ser removidos do processo de escolha, o que pode ser feito usando c_k^r. Observamos, portanto, que existe um compromisso entre c_k^r, c_k^s e c_k^e.

$$c_k^e = f(dtc_k). \tag{3}$$

Para ilustrar, na Figura 3(a), $dt1$ representa o *tile* requisitado. Esse *tile* é formado pelos *tiles* básicos $bt3$ a $bt11$. $dtc1$ (formado pelos *tiles* básicos de $bt1$ a $bt4$) e $dtc2$ (formado pelos *tiles* básicos $bt7$, $bt12$ e $bt13$) representam *tiles* na cache. Retângulos com cor de fundo rosa representam áreas do *tile* requisitado que são cobertas por *tiles* na cache, enquanto retângulos com cor de fundo azul representam áreas residuais. A borda preta em torno de $dtc1$ e $dtc2$ representa o tamanho do *tile* em cache.

Dada uma requisição para cobrir o VP de um único usuário, o OpCASH [Madarasingha et al. 2022] propõe um modelo de ILP para encontrar uma configuração ótima de *tiles* em cache (e no servidor de conteúdo) para atender a requisição, como mostrado na Equação 4.

$$maximizar \quad (w_1\mathbf{c^r} + w_2\mathbf{c^e} \; - \; w_3\mathbf{c^s})\mathbf{x}^T \tag{4}$$

sujeito a:

$$\mathbf{D}\mathbf{x}^T \leq 1 \tag{5}$$

$$x \in \{0, 1\}, \; \forall x \in \mathbf{x} \tag{6}$$

Na Equação 4, $\mathbf{x}^T \in \mathbb{R}^{|\mathcal{C}|\times 1}$ representa um vetor contendo variáveis de decisão. Cada $x_k \in \mathbf{x}^T$ é uma variável binária, de forma que $x_k = 1$ se o *tile* $dtc_k \in \mathcal{C}$ é escolhido para cobrir o VP do usuário e $x_k = 0$, caso contrário. $\mathbf{c^r}$, $\mathbf{c^e}$ e $\mathbf{c^s}$ representam vetores contendo, respectivamente, a adequação de cobertura, o tamanho do *tile* e o tamanho da área residual de cada *tile* armazenado na cache. As constantes w_1, w_2 e w_3 estabelecem pesos para a relevância de cada vetor. Essas constantes são obtidas empiricamente, devendo garantir que $w_1 + w_2 + w_3 = 1$ e $w_1, w_2, w3 > 0$. A restrição 5 estabelece que os *tiles* selecionados da cache não podem ter sobreposição, ou seja, não podem ter *tiles* básicos em comum. Para isso, a matriz D mantém informações de todos os *tiles* da cache e quais *tiles* básicos os compõem, como ilustrado na Figura 3(b). A Inequação 5 permite deixar alguns *tiles* básicos não cobertos, caso a cache não possua uma cobertura ótima para eles. Nesse caso, esses *tiles* básicos serão cobertos por *tiles* do servidor de conteúdo.

Como mencionado anteriormente, o OpCASH considera que uma única requisição chega à cache a cada instante de decisão. No entanto, ao fazer tal suposição, o OpCASH deixa de explorar possíveis intersecções de *tiles* que possam existir nos VPs dos múltiplos usuários. Ao contrário, ao receber um conjunto de requisições $\mathcal{VP}$, o DTMCash encontra uma configuração ótima de *tiles* em cache (e no servidor de conteúdo) para atender a todas as requisições, de forma a garantir que não exista redundância na recuperação de regiões residuais providas pelo servidor de conteúdo. Para ilustrar, suponha dois VPs distintos (requisitados concomitantemente) $\mathcal{VP}_i$ e $\mathcal{VP}_j$ com uma sobreposição $\mathcal{VP}_i \cap \mathcal{VP}_j \neq \emptyset$.

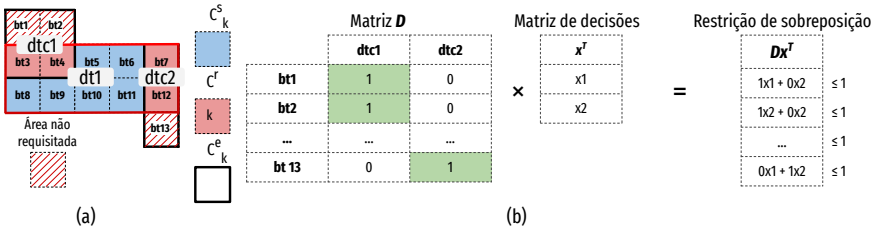

Figura 3. Variáveis para modelar a decisão da cache como um problema ILP. (a) Ilustração dos conceitos relacionados com c_k^r, c_k^s e c_k^e. (b) Geração da matriz D de modo a atender a restrição 5.

Caso não seja possível cobrir a região correspondente a $\mathcal{VP}_i \cap \mathcal{VP}_j$ totalmente com *tiles* da cache, O DTMCash busca os *tiles* residuais do servidor de conteúdo uma única vez para atender as duas requisições.

Para atingir esse objetivo, ao receber um conjunto de requisições $\mathcal{VP}$, o DTMCash precisa identificar todas as sobreposições de VPs existentes em $\mathcal{VP}$. Para identificar tais sobreposições de forma eficiente, o DTMCash rotula todos os *tiles* básicos $bt_k \in \mathcal{BT}$ de acordo com o conjunto potência de $\mathcal{VP}$, denotado por $\mathcal{P}$. Suponha, por exemplo, $\mathcal{VP} = \{\mathcal{VP}_1, \mathcal{VP}_2\}$, $\mathcal{VP}_1 = \{dtu_{1,1}\}$ e $\mathcal{VP}_2 = \{dtu_{2,1}\}$. O conjunto potencia é dado por $\mathcal{P} = \{\emptyset, \mathcal{VP}_1, \mathcal{VP}_2, (\mathcal{VP}_1, \mathcal{VP}_2)\}$. Um *tile* básico $bt_k \in \mathcal{BT}$ é rotulado como $\emptyset$ se $I(bt_k, dtu_{1,1}) = I(bt_k, dtu_{2,1}) = 0$. $bt_k \in \mathcal{BT}$ é rotulado como $\mathcal{VP}_1$ se $I(bt_k, dtu_{1,1}) = 1$ e $I(bt_k, dtu_{2,1}) = 0$. Analogamente, $bt_k \in \mathcal{BT}$ é rotulado como $\mathcal{VP}_2$ se $I(bt_k, dtu_{1,1}) = 0$ e $I(bt_k, dtu_{2,1}) = 1$. Finalmente, $bt_k \in \mathcal{BT}$ é rotulado como $(\mathcal{VP}_1, \mathcal{VP}_2)$ se $I(bt_k, dtu_{1,1}) = I(bt_k, dtu_{2,1}) = 1$.

Uma vez que cada *tile* básico $bt_k \in \mathcal{BT}$ é rotulado, podemos recuperar os *tiles* básicos que possuem o mesmo rótulo. Esses *tiles* formam uma região específica. Considerando o exemplo anterior, os *tiles* básicos com rótulos $\mathcal{VP}_1$ formam uma região requisitada apenas pelo VP $\mathcal{VP}_1$. Similarmente, os *tiles* básicos rotulados como $(\mathcal{VP}_1, \mathcal{VP}_2)$ formam a região que corresponde à intersecção dos VPs $\mathcal{VP}_1$ e $\mathcal{VP}_2$. Cada região possível pode então ser recuperada e passada, por exemplo, para o Vastile que irá propor um novo esquema de cobertura de *tile* (dinâmico) para cada região, gerando um novo conjunto de *tiles* a serem requisitados, denotado por $\mathcal{VP}' = \{\mathcal{VP}'_1, \mathcal{VP}'_2, \ldots, \mathcal{VP}'_{|\mathcal{VP}'|}\}$. O Algoritmo 1 ilustra esse procedimento.

Algorithm 1 FindIntersection

Require: conjunto de *tiles*:DTs, $\mathcal{BT}$
1: $\mathcal{P} \leftarrow PowerSet(DTs)$
2: $\mathcal{M} \leftarrow label(\mathcal{P}, \mathcal{BT})$
3: $\mathcal{A} \leftarrow getAreas(\mathcal{M})$
4: $\mathcal{VP}' \leftarrow \emptyset$
5: **for each** $a \in \mathcal{A}$ **do**
6: $\quad \mathcal{VP}' \leftarrow \mathcal{VP}' \cup Vastile(a)$
7: **end for**
8: $return\ \mathcal{VP}'$

De fato, cada requisição $\mathcal{VP}'_i \in \mathcal{VP}'$ é um conjunto de *tiles* que cobre o VP ori-

ginal do usuário u_i, porém com um esquema de *tiles* diferente do original. Então, para cada requisição $\mathcal{VP}'_i \in \mathcal{VP}'$, o DTMCash usa o modelo ILP proposto pelo OpCASH para encontrar o conjunto de *tiles* na cache e (no servidor de conteúdo) que atendam, de forma ótima, cada requisição. No entanto, ao invés de buscar no servidor de conteúdo os *tiles* que cobrem uma região residual assim que ela é gerada pela modelo ILP, esses *tiles* residuais são identificados e agrupados em um conjunto $\mathcal{R}$. Quando todas as regiões residuais de todas as requisições forem identificadas, o conjunto $\mathcal{R}$ é passado para o procedimento ilustrado no Algoritmo 1. Isso garante que, se houver intersecções entre os *tiles* correspondentes às regiões residuais, eles serão buscados apenas uma vez do servidor de conteúdo, minimizando, portanto, a quantidade de dados transferida entre o servidor de conteúdos e a cache. Uma vez buscados, os *tiles* correspondentes às regiões residuais são armazenados na cache. O Algoritmo 2 mostra o funcionamento completo do DTMCash.

Algorithm 2 DTMCash

Require: $\mathcal{VP}, \mathcal{U}, \mathcal{BT}$
1: $\mathcal{VP}' \leftarrow FindIntersection(\mathcal{VP}, \mathcal{BT})$
2: $\mathcal{R} \leftarrow \emptyset$
3: **for each** $u \in \mathcal{U}$ **do**
4: $\quad \mathcal{VP}'_u \leftarrow \emptyset$
5: $\quad$ **for each** $dtu \in \mathcal{VP}_u$ **do**
6: $\quad\quad$ **for each** $dt \in \mathcal{VP}'$ **do**
7: $\quad\quad\quad$ **if** $\exists\, bt_k \in \mathcal{BT} : I(bt_k, dtu) = I(bt_k, dt) = 1$ **then**
8: $\quad\quad\quad\quad \mathcal{VP}'_u \leftarrow \mathcal{VP}'_u \cup dt$
9: $\quad\quad\quad$ **end if**
10: $\quad\quad$ **end for**
11: $\quad$ **end for**
12: $\quad compute\ \mathbf{c^r}, \mathbf{c^e}, \mathbf{c^s}\ \ for\ \mathcal{VP}'_u$
13: $\quad cachedDTs \leftarrow ILP(\mathbf{c^r}, \mathbf{c^e}, \mathbf{c^s})$
14: $\quad \mathcal{R} \leftarrow \mathcal{R} \cup (\mathcal{VP}'_u - cachedDTs)$
15: **end for**
16: $newDTs \leftarrow FindIntersection(\mathcal{R}, \mathcal{BT})$
17: $newDTs \leftarrow requestContentServer(newDTs)$
18: $updateCache(newDTs)$

5. Avaliação Experimental

Nesta seção, apresentamos uma avaliação envolvendo o DTMCash e duas variações do OpCASH. A Figura 4 ilustra as três soluções consideradas. Como o OpCASH considera apenas uma requisição de VP a cada instante de decisão, a partir da implementação disponibilizada pelos autores da solução, implementamos duas versões distintas do OpCASH, como descrito a seguir.

A primeira versão, denominada OpCASH v1 e ilustrada no lado direito da Figura 4, aceita várias requisições de VPs (de usuários diferentes) ao mesmo tempo e as resolve, separadamente, em paralelo, de forma que a atualização da cache por uma requisição não afeta a solução das outras. Dessa forma, nesta solução, se dois VPs requisitados compartilham uma área em comum (passo 1), caso essa área em comum não

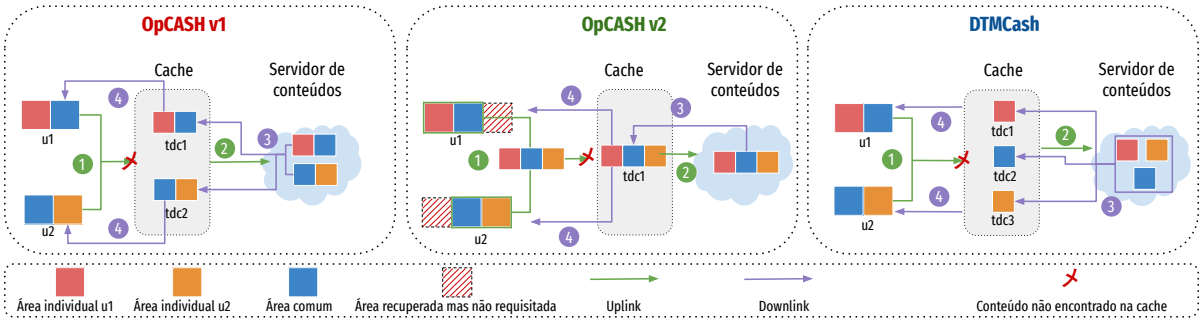

Figura 4. Comparação do DTMCash com ambas as versões do OpCASH.

possa ser completamente coberta por *tiles* da cache (passo 2), os *tiles* residuais serão requisitados duas vezes para o servidor de conteúdo, uma para cada requisição. Esses *tiles* são então transmitidos para a cache (passo 3), onde são armazenados. Em seguida, para cada requisição, os *tiles* em cache e os vindos do servidor de conteúdo são combinados para cobrir o VP requisitado e transmitidos para o usuário (passo 4). A segunda versão, denominada OpCASH v2 e ilustrada na parte central da Figura 4, aceita várias requisições de VPs (de usuários diferentes) ao mesmo tempo, unindo todos os VPs requisitados em um VP único global, o qual é atendido pela solução. Nesta versão, caso dois VPs requisitados compartilhem uma área em comum, ela é adicionada uma única vez ao VP global (passo 1). Em seguida, a cache procura localmente por *tiles* que possam cobrir o VP global. Caso não seja possível cobrir completamente o VP global com *tiles* na cache, os *tiles* residuais são requisitados para o servidor de conteúdo (passo 2), que os transmite para a cache (passo 3), onde são armazenados. Em seguida, os *tiles* em cache e os vindos do servidor de conteúdo são combinados para cobrir o VP global, que é então transmitido para os usuários (passo 4). Todos os usuários recebem os *tiles* necessários para cobrir o VP global. Por fim, o lado direito da Figura 4 corresponde ao DTMCash.

Implementamos as três soluções usando a linguagem Python (versão 3.8.10) e o *solver* glpk (versão 0.4.7). Todos os experimentos foram realizados em uma máquina contendo o sistema operacional Ubuntu 20.04 com 8GB de memória RAM e processador i5-10210U CPU @ 1.60GHz. As implementações e os resultados estão disponíveis no github[1].

Utilizamos um vídeo 360º com resolução HD (1920x1080), duração de 1 minuto e taxa de quadros de 30 FPS, coletado por [Nasrabadi et al. 2019]. Esse vídeo é dividido em 120 *chunks*, cada um de 0,5 segundos. Utilizamos também alguns esquemas de VPs gerados pelo Vastile a partir de VPs de usuários reais que assistiram o vídeo. Esses esquemas também foram gerados e disponibilizados em [Nasrabadi et al. 2019]. Como em [Madarasingha et al. 2022], assumimos $w_1 = 0{,}60$, $w_2 = 0{,}25$ e $w_3 = 0{,}15$.

Em termos de infraestrutura, assumimos uma cache com capacidade total de armazenamento de 15 Mbits que implementa uma política *Least Frequently Used* (LFU) para substituição de *tiles*, removendo o *tile* mais antigo em caso de empate. Apenas com o propósito de exercitar as soluções, assumimos que o enlace sem fio (que conecta os usuários à cache) e o enlace de Internet (que conecta a cache ao servidor de conteúdo)

[1]https://github.com/LABORA-INF-UFG/paper-GLKS-2024.

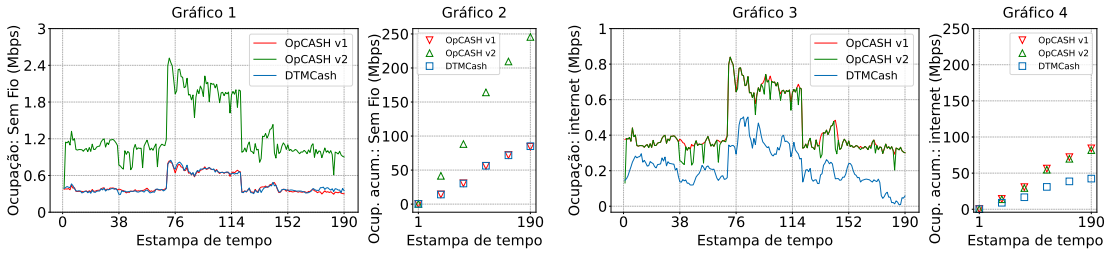

Figura 5. Ocupação dos enlaces de internet e enlace sem fio em cada estampa de tempo. Gráficos 1 e 3 representam ocupação total, enquanto gráficos 2 e 4 a ocupação acumulada.

possuem uma largura de banda de 100 Mbps.

Para simular as requisições de VPs, distribuímos um total de seis usuários em dois grupos, cada um composto por três usuários. Todos os usuários do mesmo grupo requisitam o mesmo *chunk* do vídeo, embora com VPs diferentes. Para cada requisição de VP, um esquema de VP é atribuído ao usuário. O primeiro grupo começa a assistir o video, a partir do primeiro *chunk*, na estampa de tempo 0. O segundo grupo, por sua vez, começa a assistir o vídeo, a partir do primeiro *chunk*, na estampa de tempo 70. Portanto, a partir da estampa de tempo 70, temos dois grupos de usuários consumindo o vídeo, cada grupo requisitando *chunks* diferentes. Ambos os grupos assistem os 120 *chunks* do vídeo disponibilizado.

A Figura 5 apresenta a ocupação dos enlaces sem fio e de Internet ao executar cada solução. O gráfico 1 da figura representa a ocupação do enlace sem fio em cada estampa de tempo, durante todo o experimento, enquanto o gráfico 2 representa a ocupação agregada do enlace sem fio até um certo instante de tempo, para todo o experimento. Os gráficos 3 e 4 da figura representam métricas similares porém para o enlace de Internet. Podemos perceber que o DTMCash e o OpCASH v1 geram um consumo de banda muito parecido no enlace sem fio. Contudo, o consumido de banda gerado pelo OpCASH v2 nesse enlace é significativamente maior que as outras duas soluções. Isso ocorre porque o OpCASH v2 envia, para cada usuário, os *tiles* que cobrem o VP único global, o qual pode conter *tiles* que cobrem áreas não requisitadas pelo usuário. Como consequência, o OpCASH v2 envia mais *pixels* redundantes para os usuários. Em relação ao enlace de Internet, o DTMCash reduz em até em 49.7% e 48.2% a largura de banda consumida quando comparado com as soluções OpCASH v1 e OpCASH v2, respectivamente. O ganho em relação ao OpCASH v1 é esperado, uma vez que essa versão não identifica *tiles* residuais que atendem mais de um VP, requisitando tais *tiles* mais de uma vez ao servidor de conteúdo. Por outro lado, os ganhos do DTMCash em relação ao OpCASH v2 é menos intuitivo, uma vez que ambas as soluções requisitam *tiles* residuais que atendem mais de um VP apenas uma vez ao servidor de conteúdo. Esse ganho pode ser explicado pelo VP global gerado pelo OpCASH v2, que tem um tamanho maior que os VPs individuais. Como resultado, muitos *tiles* da cache cobrem uma porção pequena do VP global, obrigando o mecanismo a buscar outros *tiles* no servidor de conteúdo.

A Figura 6 mostra a latência de comunicação fim-a-fim (i.e., a soma das latências de comunicação no enlace sem fio e no de Internet), em cada estampa de tempo, ao executar cada solução. Percebemos que o DTMCash e o OpCASH v1 promovem latências

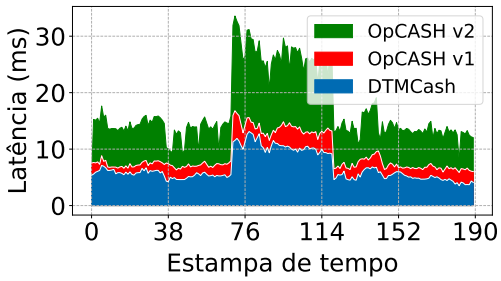

Figura 6. Latência de comunicação fim-a-fim durante a recuperação de *tiles*, por estampa de tempo

menores, com uma pequena vantagem para o DTMCash. Isso ocorre porque ambas as soluções conseguem atender muitas requisições usando *tiles* da cache, ao contrário do OpCASH v2.

A Figura 7 apresenta a taxa de acerto (primeiro gráfico da figura) e a ocupação da cache (segundo gráfico da figura), em cada estampa de tempo, ao executar cada solução. Observamos que, até a entrada do segundo grupo de usuários, que ocorre no instante 70, nenhuma requisição de VP consegue ser atendida, ainda que parcialmente, com *tiles* da cache. Isso ocorre porque todos os usuários do primeiro grupo estão requisitando os mesmos *chunks*, os quais ainda não foram requisitados anteriormente. A partir do instante 70, o segundo grupo de usuários começa a assistir o vídeo, requisitando *tiles* para *chunks* que já foram acessados anteriormente. No entanto, como mostrado no segundo gráfico, próximo ao instante 38, o OpCASH v1 e o OpCASH v2 atingem o limite de capacidade da cache, o que leva a cache a remover *tiles* mais antigos. Como consequência, a taxa de acerto dessas soluções ao longo do experimento, se mantém próxima de zero. Ao contrário, o DTMCash, é mais eficiente na ocupação da cache e atinge a capacidade máxima de armazenamento próxima do instante 76. Isso permite ao DTMCash ter uma taxa de acerto melhor que os outros dois modelos, principalmente no final do experimento.

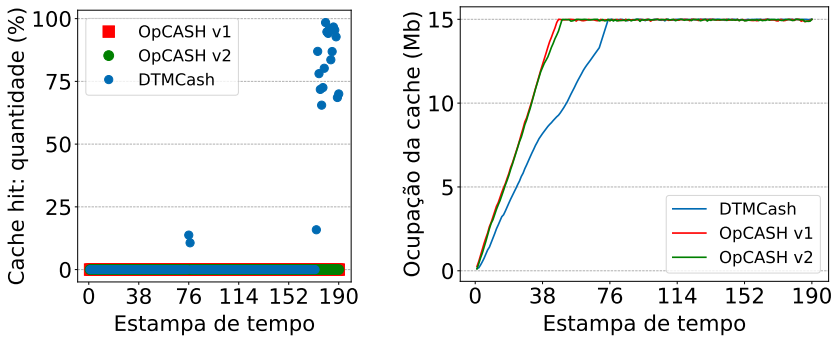

Figura 7. (1) Cache hit por quantidade de *tiles* básicos; (2) Ocupação da cache em cada estampa de tempo.

Em resumo, comparado com as duas versões do OpCASH, observamos que o DTMCash provê melhor utilização dos enlaces de Internet e sem fio, reduzindo a quantidade de dados transmitidos nesses enlaces. O DTMCash também provê menor latência de comunicação uma vez que ele consegue atender mais requisições usando *tiles* da cache.

Por fim, o DTMCache provê uma taxa de acerto maior, mostrando uma ocupação mais eficiente da cache.

6. Conclusão e Trabalhos Futuros

Neste artigo, propomos o DTMCash, uma abordagem eficiente de utilização de cache na borda para fornecer cobertura ótima de *tiles* em cache para VPs de múltiplos usuários com tiles dinâmicos em streaming de vídeo 360º. DTMCash tenta minimizar a quantidade de *pixels* redundantes e a quantidade de *tiles* residuais ao cobrir os VPs de múltiplos usuários com *tiles* da cache, minimizando a quantidade de dados requisitada para o servidor de conteúdo pela identificação de áreas redundantes entre os VPs solicitados. Comparamos, então, o DTMCash com duas versões multi-usuário do OpCASH. Os resultados mostram que a nossa solução demonstra uma redução significativa na largura de banda consumida no enlace de Internet enquanto mantém praticamente o mesmo consumo de banda que a melhor versão do OpCASH para o mesmo problema. Adicionalmente, o DTMCash supera as versões do OpCASH em relação à utilização da cache, apresentando melhor ocupação da cache ao longo do tempo. Portanto, o DTMCash representa um avanço importante na busca por soluções eficazes para a transmissão de vídeos 360º em redes sem fio.

Como trabalhos futuros, pretendemos estender o DTMCash e incorporar a seleção de resolução de *tiles* no problema de seleção ótima de *tiles* dinâmicos armazenados na cache.

Reconhecimentos

Este trabalho foi apoiado pela CAPES; pela RNP/MCTIC, através do projeto No. 01245.010604/2020-14 – Sistemas de Comunicações Móveis 6G; pela Anatel e FAPEG, através do projeto “Avaliação de Impacto da Web 3.0: Descentralizada, Imersiva, Semântica, Centrada no Usuário e Conectada com o Mundo Ciberfísico”.

Referências

Cao, J., Lam, K.-Y., Lee, L.-H., Liu, X., Hui, P., and Su, X. (2023). Mobile Augmented Reality: User Interfaces, Frameworks, and Intelligence. *ACM Comput. Surv.*, 55(9).

Hsu, C.-H. (2020). MEC-Assisted FoV-Aware and QoE-Driven Adaptive 360° Video Streaming for Virtual Reality. In *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*, pages 291–298.

Li, J., Wen, Z., Li, S., Zhao, Y., Guo, B., and Wen, J. (2016). Novel tile segmentation scheme for omnidirectional video. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 370–374.

Liu, L., Zhong, R., Zhang, W., Liu, Y., Zhang, J., Zhang, L., and Gruteser, M. (2018). Cutting the Cord: Designing a High-Quality Untethered VR System with Low Latency Remote Rendering. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys ’18)*, page 68–80.

Lo, W.-C., Huang, C.-Y., and Hsu, C.-H. (2018). Edge-Assisted Rendering of 360° Videos Streamed to Head-Mounted Virtual Reality. In *2018 IEEE International Symposium on Multimedia (ISM)*, pages 44–51.

Madarasingha, C. and Thilakarathna, K. (2021). Vastile: Viewport adaptive scalable 360-degree video frame tiling. In *Proceedings of the 29th ACM International Conference on Multimedia*, MM '21, page 4555–4563, New York, NY, USA. Association for Computing Machinery.

Madarasingha, C., Thilakarathna, K., and Zomaya, A. (2022). Opcash: Optimized utilization of mec cache for 360-degree video streaming with dynamic tiling. In *2022 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 34–43. IEEE.

Nasrabadi, A. T., Samiei, A., Mahzari, A., McMahan, R. P., Prakash, R., Farias, M. C. Q., and Carvalho, M. M. (2019). A taxonomy and dataset for 360° videos. In *Proceedings of the 10th ACM Multimedia Systems Conference*, MMSys '19, page 273–278, New York, NY, USA. Association for Computing Machinery.

Ozcinar, C., Cabrera, J., and Smolic, A. (2019). Visual Attention-Aware Omnidirectional Video Streaming Using Optimal Tiles for Virtual Reality. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(1):217–230.

Papaioannou, G. and Koutsopoulos, I. (2019). Tile-based Caching Optimization for 360° Videos. In *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 171–180.

Qian, F., Han, B., Xiao, Q., and Gopalakrishnan, V. (2018). Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18)*, page 99–114.

van der Hooft, J., Amirpour, H., Vega, M. T., Sanchez, Y., Schatz, R., Schierl, T., and Timmerer, C. (2023). A Tutorial on Immersive Video Delivery: From Omnidirectional Video to Holography. *IEEE Communications Surveys & Tutorials*, 25(2):1336–1375.

Wu, D., Yang, Z., Zhang, P., Wang, R., Yang, B., and Ma, X. (2023). Virtual-Reality Interpromotion Technology for Metaverse: A Survey. *IEEE Internet of Things Journal*, 10(18):15788–15809.

Yao, J., Han, T., and Ansari, N. (2019). On mobile edge caching. *IEEE Communications Surveys & Tutorials*, 21(3):2525–2553.

Yaqoob, A., Bi, T., and Muntean, G.-M. (2020). A Survey on Adaptive 360° Video Streaming: Solutions, Challenges and Opportunities. *IEEE Communications Surveys & Tutorials*, 22(4):2801–2838.

Zhou, C., Xiao, M., and Liu, Y. (2018). ClusTile: Toward Minimizing Bandwidth in 360-degree Video Streaming. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 962–970.