雾无线接入网中的多层协作缓存方法

蒋雁翔[1]，夏骋宇[1,2]

（1. 东南大学移动通信国家重点实验室，江苏 南京 210096；2. 香港科技大学电子与计算机工程系，香港 999077）

摘 要：针对边缘缓存和雾无线接入技术中降低后传链路负载的问题，提出了一种雾无线接入网中的多层协作缓存方法。考虑网络架构、文件流行度估计、链路容量等因素，将相关的优化问题分解为每个层中缓存布置的背包问题，并分别使用贪心算法求解。仿真结果表明，所提出的多层协作缓存方法能够有效降低后传链路负载，同时具有较高的缓存命中率。

关键词：雾无线接入网；协作缓存；后传链路负载；文件流行度

中图分类号：TN915

文献标识码：A

doi: 10.11959/j.issn.1000−436x.2019093

Multi-tier cooperative caching in fog radio access network

JIANG Yanxiang[1], XIA Chengyu[1,2]

1. National Mobile Communications Research Laboratory, Southeast University, Nanjing 210096, China

2. Electronic and Computer Engineering Department, HongKong University of Science and Technology, HongKong 999077, China

Abstract: Aiming at the problem of reducing the load of the backward link in the edge buffer and fog wireless access network technology, a multi-tier cooperative caching scheme in F-RAN was proposed to further reduce the backhaul traffic load. In particular, by considering the network topology, content popularity prediction and link capacity, the optimization problem was decomposed into knapsack subproblems in multi-tiers, and effective greedy algorithms were proposed to solve the corresponding subproblems. Simulation results show that the proposed multi-tier cooperative caching scheme can effectively reduce the backhaul traffic and achieve relatively high cache hit rate.

Key words: fog radio access network, cooperative caching, backhaul traffic load, content popularity

1 引言

近年来，移动网络中对于多媒体业务的需求呈现爆炸式增长，由此带来的传输链路负载，尤其是无线接入网络和后传链路中的负载成为移动运营商们亟待解决的问题[1-2]。为了解决这些问题，研究者们提出了许多适用于未来新一代移动通信系统的新型网络架构及创新的文件传输方式[3-7]，其中一种比较流行的方法就是将流行文件缓存在离用户较近的地方，这种方法主要包括以下 2 种方案。

1) 在基站部署缓存，以有效降低用户对于流行内容的重复下载而带来的后传链路负担，同时还可以提高移动网络的服务质量，如降低时延等[2-7]。

2) 采用新型网络的架构，将基站小型化，使整个通信网络都更加靠近用户。在雾无线接入网中，有较为分散部署的云节点（CU, cloud unit）、大量密集部署的小型雾接入点（F-AP, fog access point）等，形成了一种多层的拓扑结构。由于减小了接入节点和用户的距离，雾无线接入网架构可以很有效地提高无线网络的区域频谱效率及网络容量[4-6]。

然而，在对雾网缓存布置的研究中，始终有 2 个很重要的问题[4-6]，一是底层的 F-AP 缓存容量都较小，无法缓存大量的流行文件；二是 F-AP 覆盖的用户很少，导致他们产生的文件请求无法反映文件的聚集效应，因此无法准确估计出在本地流行的文件。考虑到上述 2 个问题，本文基于 F-AP 成簇的场景，提出了一种多层协作缓存方法。单个 F-AP

收稿日期：2019−01−05；修回日期：2019−06−27

覆盖用户很少，但是一个 F-AP 簇可以覆盖拥有相似文件喜好的一群用户（例如一幢写字楼中的员工、一个体育场中的观众等），将 F-AP 适当成簇，可以通过更多的文件请求更准确地估计出文件的本地流行度；而多层的协作缓存则可以提高系统的协作度，弥补 F-AP 缓存容量小的问题。

本文考虑了在 F-AP 成簇的雾无线接入网中的多层协作缓存，提出了一种有效的缓存布置策略和层间、层内的协作策略。本文的贡献如下。

1) 相对于其他文献考虑每个 F-AP 单独缓存，本文考虑 F-AP 成簇场景下的多层缓存布置，优化问题更为复杂，但更加具有实际意义。

2) 综合考虑了传输信道容量及多层缓存的协作机制，提出了高效的缓存布置及协作算法，降低整个网络尤其是后传链路的负载。

3) 将缓存布置优化问题分解为每层内的缓存布置子问题，从而简化了问题求解的复杂度，并将每层的优化问题都转化成背包问题，进而采用贪心算法求解。

4) 仿真结果表明，本文提出的多层协作缓存算法可以显著降低雾无线接入网中的后传链路负载及提高缓存命中率。

2　系统建模

2.1　多层缓存架构

图 1 展示了一种雾无线接入网中的多层缓存的网络模型，其中，CU 为云节点，ue 为用户设备。最顶端为云端，存储整个系统中的文件库。CU 有一定容量的缓存，并通过后传链路和云端相连，通过前传链路与网络边缘的 F-AP 相连。网络边缘的 F-AP 也带有一定大小的缓存，且形成了很多簇。由于 CU 的覆盖范围较广，同一个 CU 可以覆盖多个 F-AP 簇。本文将同一个簇中的 F-AP 定义为相邻的，相邻 F-AP 之间可以通过簇头实现协作缓存。而同一个 CU 覆盖下的 F-AP 簇间可以通过 CU 实现协作缓存。如前文所述，网络中的 CU 和 F-AP 都能缓存一些文件，以此来降低用户下载文件的时延及重复从云端下载流行文件所带来的链路负载。由于 CU 和 F-AP 的缓存，以及 F-AP 簇的存在，整个网络形成了一种三层的缓存拓扑结构，Tire 0 为 CU 层，Tire 1 为 F-AP 簇间（通过 CU 控制协作），Tire 2 为 F-AP 簇内（通过簇头控制协作）。

网络中每个簇头都存有一个本簇内其他 F-AP 的缓存文件表，每个 CU 都存有一个它所覆盖的所有簇的缓存文件表，根据相关研究，由此带来的负载可以忽略不计。这样，簇头就能为簇内的其他 F-AP 做出缓存决定。CU 则可以为每个簇做出缓存决定，并控制簇间的协作。用户向 F-AP 申请文件时，F-AP 首先检查本地缓存，如果文件在本地缓存中，则直接发送给用户；如果本地没有缓存，F-AP 则会将申请发送到簇头，如果簇头仍没有缓存，簇头会控制簇内其他 F-AP 协作；若整个簇内都没有缓存此文件，则簇头会将申请发送到 CU，如果在 CU 仍没有命中，则 CU 会控制其他簇的协作；如果 CU 覆盖的所有簇内都没有缓存这个文件，则从云端下载。

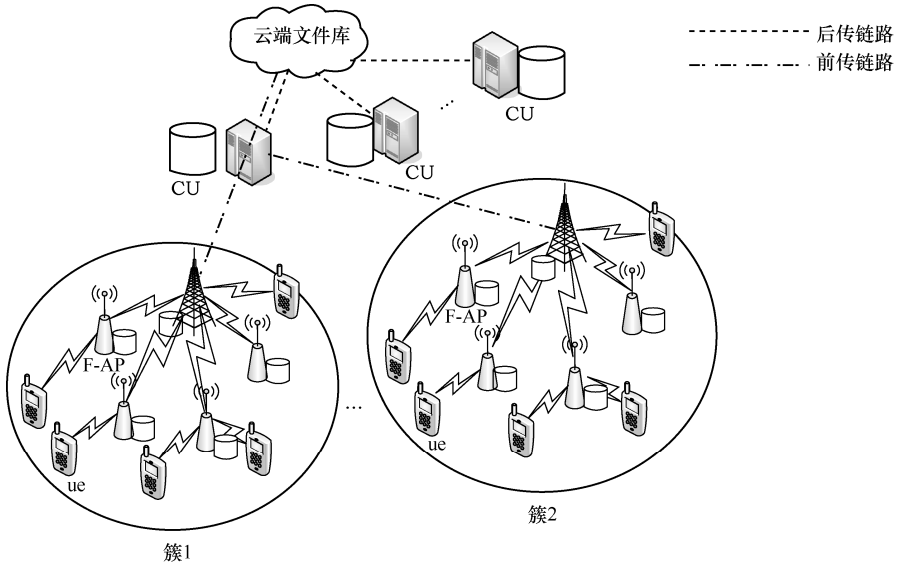


图 1　雾无线接入网中的多层缓存网络模型

本文假设文件的流行度变化很慢，系统可以在负载较低的时候（例如深夜）缓存那些流行文件。由于一段时间内文件的流行度可以看成是固定不变的，本文不考虑更新文件所带来的负载。文件的流行度可以通过机器学习及分析用户的行为或偏好等得到[4]，因此本文假设文件流行度是已知的。

2.2 系统模型

本文在建模中考虑只有一个 CU 的系统，包括一个 CU、M 个簇，第 m 个簇中有 N_m 个 F-AP。Q_0 表示 CU 的缓存容量，$\{Q_1,Q_2,Q_3,\cdots,Q_m,\cdots,Q_M\}$ 表示 M 个簇的缓存容量，$\{Q_{m1},Q_{m2},Q_{m3},\cdots,Q_{mN_m}\}$ 表示第 m 个簇中的 F-AP 的缓存容量。F-AP_n^m 表示第 m 个簇中的第 n 个 F-AP，C_n^m 为 F-AP_n^m 和簇头之间的链路容量，C_m 为第 m 个簇和 CU 之间的链路容量。

假设系统中共有 F 个文件，用 $\{o_1,o_2,o_3,\cdots,o_f,\cdots,o_F\}$ 表示。考虑到实际情况，所有文件的大小都是不相同的，用 $\{s_1,s_2,s_3,\cdots,s_f,\cdots,s_F\}$ 表示文件的大小。第 m 个簇中的 F-AP 的缓存布置由一系列二值数 $\{x_{m1}^f,x_{m2}^f,x_{m3}^f,\cdots,x_{mn}^f,\cdots,x_{mN_m}^f\}$ 表示，例如 $x_{m1}^f=1$ 表示文件 o_f 缓存在 F-AP_1^m 中，$x_{m1}^f=0$ 没有缓存。类似地，$\{x_1^f,x_2^f,x_3^f,\cdots,x_m^f,\cdots,x_M^f\}$ 表示 M 个簇头中的缓存布置。一旦 F-AP 层的缓存布置确定后，F-AP 簇内的缓存布置也就确定了。定义 λ_{mn}、λ_m、λ_0 分别为文件请求到达 F-AP_n^m、第 m 个簇头及 CU 的平均到达率。需要注意的是，到达第 m 个簇头的文件请求可能来自簇头本身覆盖的用户或者簇内的 F-AP。因此，本文定义 β_m 为簇头 m 的用户发送的文件请求的平均到达率。对文件 o_f 的请求在 F-AP_n^m 及簇头 m 的到达率分别为 λ_{mn}^f 和 λ_m^f。考虑到簇头 m 处对文件 o_f 的请求可能来自直接相连的用户，也可能来自簇内的 F-AP，因此定义 β_m^f 为簇头 m 的用户对文件 o_f 的请求的到达率。定义文件 o_f 在 F-AP_n^m 处的流行度和归一化流行度分别为 p_{mn}^f 和 r_{mn}^f，类似地，簇头 m 的用户申请文件 o_f 的流行度定义为 q_m^f。因此，可以得到

$$r_{mn}^f=\frac{p_{mn}^f}{\sum_{i=1}^{F}p_{mn}^i} \tag{1}$$

$$\lambda_{mn}^f=r_{mn}^f\lambda_{mn} \tag{2}$$

$$\beta_m^f=\frac{q_m^f\beta_m}{\sum_{i=1}^{F}q_m^i} \tag{3}$$

本文假设文件的全局流行度 $\{P_1,P_2,P_3,\cdots,P_f,\cdots,P_F\}$ 满足 Zipf 分布，显然有 $P_f=\sum_{m=1}^{M}\sum_{n=1}^{N_m}p_{mn}^f+\sum_{m=1}^{M}q_m^f$。

3 优化问题

本节研究文件的最优缓存策略，从而减小网络后传链路的负载。由于整个系统涉及文件在三层缓存中的纵向分布及每层内的横向分布，较为复杂，因此，考虑将优化问题分解为三层内的缓存分布问题，并分别求解。

3.1 F-AP 簇内协作缓存

对于用户发出的文件请求，如果本地的 F-AP 正好缓存了被请求的文件，则请求可以被立即满足；如果簇内其他 F-AP 缓存了这个文件，则文件请求会被发送到簇头，再由簇头控制簇内协作，实现用户的下载。

$y_{kn}^{fm}\in\{0,1\}$ 表示簇头 m 是否将文件 o_f 从 F-AP_k^m 取出并发送到 F-AP_n^m，并且认为 $y_{nn}^{fm}=0$。$\lambda_{mn}^f s_f$ 表示在 F-AP_n^m 处请求文件 o_f 所需要的信道容量。因此，在 F-AP_n^m 处，本地可以满足的信道容量表示为

$$\sum_{f=1}^{F}x_{mn}^f\lambda_{mn}^f s_f \tag{4}$$

而 F-AP_n^m 本地没有命中缓存，需要簇内其他 F-AP 协作满足的信道容量表示为

$$\sum_{f=1}^{F}\sum_{k=1}^{N_m}y_{kn}^{fm}\lambda_{mn}^f s_f \tag{5}$$

在不同 F-AP 簇中的协作是可以同时进行、互不干扰的，因此，得到如式(6)所示的优化问题来表示第 m 个簇中的协作。

$$\max_{x_{mn}^f,y_{kn}^f}\left\{\sum_{f=1}^{F}\sum_{n=1}^{N_m}x_{mn}^f\lambda_{mn}^f s_f+\sum_{f=1}^{F}\sum_{n=1}^{N_m}\sum_{k=1}^{N_m}y_{kn}^{fm}\lambda_{mn}^f s_f\right\} \tag{6}$$

$$\text{s.t.}\quad\sum_{f=1}^{F}x_{mn}^f s_f\leqslant Q_{mn},\quad\text{for all } n \tag{7}$$

$$\sum_{f=1}^{F}\sum_{n=1}^{N_m}y_{kn}^{fm}\lambda_{mn}^f s_f\leqslant C_k^m,\quad\text{for all } k \tag{8}$$

$$\sum_{f=1}^{F}\sum_{n=1}^{N_m} y_{kn}^{fm}\lambda_{mn}^{f}s_f \leqslant C_n^m, \quad \text{for all } n \tag{9}$$

$$y_{kn}^{fm} \leqslant x_{mk}^{f} \,\|\, x_{mn}^{f} - x_{mn}^{f}, \quad \text{for all } n,k,f \tag{10}$$

$$\sum_{k=1}^{N_m} y_{kn}^{fm} \leqslant \bigcup_{k=1}^{N_m} x_{mk}^{f} - x_{mn}^{f}, \quad \text{for all } n,f \tag{11}$$

其中，$\bigcup_{k=1}^{N_m} x_{mk}^{f} - x_{mn}^{f} = x_{m1}^{f} \| x_{m2}^{f} \| x_{m3}^{f} \| \cdots \| x_{mN_m}^{f} - x_{mn}^{f}$，$\|$表示逻辑或运算。

式(7)限制了每个 F-AP 中缓存的文件大小；式(8)和式(9)限制了 F-AP 和簇头之间的信道容量；式(10)限制了若本地 F-AP 缓存了被请求文件，则协作不会发生；式(11)限制了每次最多只会有一个 F-AP 与请求文件的 F-AP 协作，而不会出现多个 F-AP 同时向请求文件的 F-AP 传输，避免了重复传输带来的资源浪费。

式(10)和式(11)可以分别通过放缩转化为线性不等条件，而且可以证明，这种转化是等价的，即式(10)和式(11)分别等价于式(12)和式(13)。

$$y_{kn}^{fm} \leqslant x_{mk}^{f}, \quad \text{for all } n,k,f \tag{12}$$

$$\sum_{k=1}^{N_m} y_{kn}^{fm} \leqslant 1 - x_{mn}^{f}, \quad \text{for all } n,f \tag{13}$$

证明　如附录所示。

这样，优化问题就变成了一个整数线性规划问题，这是一个 NP-hard 问题，考虑到其指数式的复杂度不可能求得它的最优解。因此，本文参考主元的思想，提出一种较为简单的算法，希望求得该优化问题的次优解。首先，将优化问题分成 2 个子问题，具体如下。

子问题 1　优化 x_{mn}^{f}。

$$\max_{x_{mn}^{f}} \sum_{f=1}^{F}\sum_{n=1}^{N_m} x_{mn}^{f}\lambda_{mn}^{f}s_f$$
$$\text{s.t.} \quad \text{式(7)} \tag{14}$$

子问题 2　在 x_{mn}^{f} 的基础上，优化 y_{kn}^{fm}。

$$\max_{y_{kn}^{f}} \sum_{f=1}^{F}\sum_{n=1}^{N_m}\sum_{k=1}^{N_m} y_{kn}^{fm}\lambda_{mn}^{f}s_f$$
$$\text{s.t.} \quad \text{式(7)~式(10)} \tag{15}$$

由于先优化 x_{mn}^{f}，可以认为此时簇内 N_m 个 F-AP 的缓存布置是独立的，因此可以使用贪心算法求解子问题 1。在确定了 x_{mn}^{f} 后，子问题 2 也可用类似的贪心算法求解，从而解决整个优化问题。解决 2 个子问题的贪心算法如算法 1、算法 2 所示。算法 1 和算法 2 的总体复杂度为 $O(T1\text{b}T)$，其中 $T = FN_mN_m$ 为总迭代次数。在本文所提算法中，解决子问题 1 实际上就是解决文件在 F-AP 簇中的分布；解决子问题 2 实际上就是在已有的缓存分布上实现了通过簇头进行簇内协作缓存。

算法 1　解决子问题 1 的贪心算法

1) 输入 $m,F,N_m,\lambda_{mn}^{f},s_f,C_n^m,Q_{mn}$
2) 设 S_{mn} 为 F-AP$_n^m$ 的缓存空间
3) 初始化 $x_{mn}^{f}=0, S_{mn}=0$
4) for $f \in \{1,2,3,\cdots,FN_m\}$ do
5)　　计算 $\lambda_{mn}^{f}s_f$
6) end for
7)　将 $\lambda_{mn}^{f}s_f$ 降序排列，$W_i \leftarrow f$
8)　$j=1$
9)　while $S_{mn} < Q_{mn}$ do
10)　　$x_{mn}^{W[j]}=1$
11)　　$S_{mn} = S_{mn} + s_{W[j]}$
12)　　$j = j+1$
13)　end while
14) 输出 x_{mn}^{f}

算法 2　解决子问题 2 的贪心算法

1)　输入 $m,F,N_m,\lambda_{mn}^{f},s_f,C_n^m,Q_{mn}$
2)　初始化 $y_{kn}^{fm}=0$
3)　执行算法 1（优化 x_{mn}^{f}）
4)　　设　$\text{Cl}_n = \text{Cd}_n = C_n^m$；　$A_{FN_m\times 1} = \pi_{FN_m\times 1} = \omega_{FN_m\times 1} = 0$
5)　$A_j = \lambda_{mn}^{f}s_f$，并使 $f = \text{mod}(j-1,F)+1$ 且 $n = \dfrac{j-f}{F}+1$
6)　将 A 降序排列，$\omega_i \leftarrow j$
7)　for $j=1$ to FN_m do
8)　　计算 $f = \text{mod}(j-1,F)+1$ 和 $n = \dfrac{j-f}{F}+1$
9)　　if（$x_{mn}^{f} == 0$）&&（$\sum_{i=1}^{N_m} x_{mi}^{f} \geqslant 1$）&&（$A_j \leqslant \text{Cd}_n$）then
10)　　找出所有的 $n \in \{1,2,\cdots,N_m\}$ 满足 $x_{mn}^{f}=1$
11)　　找出 $i \in n$，满足 $\arg\min\left(\dfrac{\lambda_{mi}^{f}s_f}{\text{Cl}_i}\right)$
12)　　if $\lambda_{mi}^{f}s_f \leqslant \text{Cl}_i$ then

13) $y_{in}^{fm}=1$

14) end if

15) end if

16) end for

17) 输出 y_{kn}^{fm} 和 x_{mn}^{f}

3.2 F-AP 簇间协作缓存

3.1 节已经考虑了簇内如何实现协作，并得到了 x_{mn}^{f} 和 y_{kn}^{fm}，则对文件 o_f 的申请在簇头 m 本地的到达率为

$$\lambda_m^f=[\sum_{n=1}^{N_m}\lambda_{mn}^f(1-x_{mn}^f-\sum_{k=1}^{N_m}y_{kn}^{fm})]+\beta_m^f \tag{16}$$

簇头处的文件请求包含了簇头用户发送的文件请求及簇内 F-AP 发送的文件请求。

与 3.1 节类似，本文定义 $y_{mt}^{f}\in\{0,1\}$ 来表示簇 m 是否将文件 o_f 通过协作发送到簇 t，并且认为 $y_{mm}^{f}=0$，$\lambda_m^f s_f$ 来表示在簇头 m 处申请文件 o_f 的信道容量。由于簇间的协作通过 CU 来控制实现，考虑到 CU 具有充分的计算资源，以及簇和 CU 间的前传链路负载，本文定义一个增益函数用来表示将文件 o_f 存放在簇 m 中或者它相邻的簇中对于信道容量使用率的增益，如式(17)所示。

$$G_m^f=\frac{G_{\mathrm{loc},m}^f+G_{\mathrm{nei},m}^f}{s_f} \tag{17}$$

其中，$G_{\mathrm{loc},m}^f$ 表示文件 o_f 存放在簇头 m 中对本簇中的请求所带来的增益，$G_{\mathrm{nei},m}^f$ 表示对相邻的簇中的请求所带来的增益；分母为文件大小 s_f 是为了优先选择增益较大且体积较小的文件，进一步提高缓存命中率。很显然，本簇的到达率 λ_m^f 越高，则 $G_{\mathrm{loc},m}^f$ 越大；相邻簇的到达率 λ_{nei}^f 越高，则 $G_{\mathrm{nei},m}^f$ 越大。而如果一个 CU 的蜂窝中，只有一个簇 m，很显然簇 m 没有相邻簇，此时有最大 $G_{\mathrm{loc},m}^f$，$G_{\mathrm{loc},m}^f=x_m^f\lambda_m^f s_f$，而 $G_{\mathrm{nei},m}^f=0$。

因此，定义增益函数的表达式为

$$G_{\mathrm{loc},m}^f=x_m^f\lambda_m^f s_f-\frac{\sum_{\{t|t\in M\&t\neq m\}}y_{mt}^f\lambda_t^f s_f(1-\eta)}{M} \tag{18}$$

$$G_{\mathrm{nei},m}^f=\sum_{\{t|t\in M\&t\neq m\}}y_{tm}^f\lambda_t^f s_f(1-\eta) \tag{19}$$

其中，η（$0\leqslant\eta\leqslant1$）是本文引入的一个系数，用于表征 CU 所控制协作的成功率，η 越小，则 $G_{\mathrm{nei},m}^f$ 越大，表示协作的成功率越高。例如，当 $\eta=0$ 时，有 $G_{\mathrm{nei},m}^f=\sum_{\{t|t\in M\&t\neq m\}}y_{tm}^f\lambda_t^f s_f$，这时簇头 m 能与所有请求此文件的簇协作；而当 $\eta=1$ 时，$G_{\mathrm{nei},m}^f=0$，簇间没有协作发生。由此可得簇间协作的优化问题为

$$\max_{x_m^f,y_{mt}^f}\left(\sum_{f=1}^{F}G_m^f\right) \tag{20}$$

$$\text{s.t.}\quad\sum_{f=1}^{F}x_m^f s_f\leqslant Q_m,\quad\text{for all } m \tag{21}$$

$$\sum_{\{t|t\in M\&t\neq m\}}y_{mt}^f\leqslant1-x_m^f,\quad\text{for all } m,f \tag{22}$$

$$y_{mm}^f=0,\quad\text{for all } m,f \tag{23}$$

$$\sum_{f=1}^{F}y_{mt}^f\lambda_t^f s_f\leqslant\min(C_m,C_t) \tag{24}$$

其中，式(21)限制了簇头的缓存容量；式(22)和式(23)保证了当本簇中已经缓存了被请求文件时，就不会再向其他簇请求文件。显然式(20)也是一个整数线性规划问题，同样也是 NP-hard 问题。与 3.1 节类似，本文依然希望采用贪心算法求出次优解。

首先，式(20)所示的问题可以展开为

$$\max_{x_m^f,y_{mt}^f}\left\{x_m^f\lambda_m^f-\frac{1}{M}\sum_{\{t|t\in M\&t\neq m\}}y_{mt}^f\lambda_t^f(1-\eta)+\sum_{\{t|t\in M\&t\neq m\}}y_{tm}^f\lambda_m^f(1-\eta)\right\} \tag{25}$$

式(20)可以进一步被分解成 2 个子问题，即子问题 3 和子问题 4。

子问题 3 优化 x_m^f，如式(26)所示。

$$\max_{x_m^f}(x_m^f\lambda_m^f)$$
$$\text{s.t.}\quad\text{式(21)} \tag{26}$$

子问题 4 在式(26)结果的基础上优化 y_{mt}^f，如式(27)所示。

$$\max_{y_{mt}^f}\{x_m^f\lambda_m^f-\frac{1}{M}\sum_{\{t|t\in M\&t\neq m\}}y_{mt}^f\lambda_t^f(1-\eta)+\sum_{\{t|t\in M\&t\neq m\}}y_{tm}^f\lambda_m^f(1-\eta)\}$$
$$\text{s.t.}\quad\text{式(21)}\sim\text{式(24)} \tag{27}$$

与 3.1 节类似，提出求解子问题 3 和子问题 4 的贪心算法，如算法 3 和算法 4 所示。算法 3 和算法 4 的总体复杂度为 $O(FM^2\mathrm{lb}(FM^2))$。

算法 3　解决子问题 3 的贪心算法

1) 输入 $M,F,\eta,\lambda_m^f,s_f,Q_m$

2) 设 S_m 为簇头 m 的缓存空间

3) 初始化 $x_m^f=0,S_m=0$

4) for　$f=1$ to F　do

5)　　计算 $x_m^f\lambda_m^f$

6) end for

7) 倒序排列 $x_m^f\lambda_m^f$ 并记录原下标 f，$W_i\leftarrow f$

8)　$j=1$

9)　while　$S_m<Q_m$　do

10)　　if　（s_j 在簇 m 的缓存表中）　//保证簇头不会缓存簇中 F-AP 已缓存的文件

11)　　$j=j+1$

12)　　else

13)　　$x_m^{W[j]}=1$

14)　　$S_m=S_m+s_{W[j]}$

15)　　$j=j+1$

16)　　end if

17)　end while

18) 输出 x_m^f

算法 4　解决子问题 4 的贪心算法

1) 输入 $M,F,\eta,\lambda_m^f,s_f,Q_m$

3) 初始化 $x_m^f=0,y_{mt}^f=0$

4) 执行算法 3，得到 x_m^f

5) 将算法 3 结果中 x_m^f=1 对应的 (m,f) 放入中间变量集合 $\mathbb{R}$

6) 设 $A_{mt}^f=\frac{1}{M}y_{mt}^f\lambda_t^f(1-\eta)$，　$B_{tm}^f=y_{tm}^f\lambda_m^f(1-\eta)$

7) while　$\mathbb{R}\neq\varnothing$ do

8)　　找到 (m,f,t)（其中 $(m,f),(m,t)\in\mathbb{R}$），$\arg\max\limits_{(m,f,t)}\{B_{tm}^f-A_{mt}^f\}$

9)　　if　$(\lambda_t^f s_f<C_t\ \&\&\lambda_m^f s_f<C_m)$　then

10)　　$y_{mt}^f=1,C_t=C_t-\lambda_t^f s_f,C_m=C_m-\lambda_m^f s_f$

11)　　$\mathbb{R}\leftarrow\frac{\mathbb{R}}{(m,f,t)}$

12)　　else

13)　　$\mathbb{R}\leftarrow\frac{\mathbb{R}}{(m,f,t)}$

14)　　end if

15) end while

16) 输出 x_m^f，　y_{mt}^f

3.3　CU 层缓存分布

通过上述的分析，得到了 F-AP 簇内的缓存分布、协作方式，以及簇头的缓存分布和簇间协作方式。因此，可以得到从簇层到达 CU 层的文件请求到达率，如式(28)所示。

$$\lambda_0^f=\sum_{m=1}^{M}\lambda_m^f\Big(1-x_m^f-\sum_{\{t|t\in M\,\&\,t\neq m\}}y_{tm}^f\Big)\tag{28}$$

类似地，对于 CU 层的缓存分布同样可以找到优化问题来最大化链路容量，如式(29)所示。

$$\max_{x^f}\sum_{f=1}^{F}x_f\lambda_0^f s_f,\quad \text{s.t.}\sum_{f=1}^{F}x_f s_f\leqslant Q_0\tag{29}$$

其中，x_f 表示 CU 层的缓存分布，$x_f=1$ 表示文件 o_f 被缓存在 CU 中，否则表示没有被缓存在 CU 中。CU 层的优化问题是一个简单的背包问题，本文用复杂度很低的背包算法[8]来解决。

4　仿真结果与分析

首先，假设有 500 个文件，网络中有一个 CU，它通过前传链路连接到 2 个簇的簇头，每个簇中有 5 个 F-AP。同时，假设 F-AP、簇头、CU 的缓存大小关系为 $Q_{mn}:Q_m:Q_0=1:4:10$；假设每层之间的链路容量为 $C_n^m=1$ Gbit/s，$C_m=2$ Gbit/s，系数 $\eta=0.9$。

为了便于仿真，假设每个簇中有 50 个用户，共 100 个用户。在每个簇中，簇头覆盖了 30 个用户，5 个 F-AP 中的每一个覆盖 5 个用户，并且其覆盖区域互不重叠。文件 o_f 的流行度 p_{mn}^f 和 q_m^f 可以通过机器学习的办法获得，为了便于仿真，假设 p_{mn}^f 和 q_m^f 服从 Zipf 分布。

为了便于横向比较算法性能，本文选取了另外 2 种缓存分布算法：部分协作的多层缓存算法[9]和基于 BP（belief propagation）算法的分布式缓存算法[10]，具体如下。

1) 部分协作的多层缓存算法。这种算法也是一种多层缓存布置算法，与本文所提算法的区别在于这种算法没有考虑层内的缓存协作，只有层间的协作。

2) 基于 BP 算法的分布式缓存算法。这种算法中只有最底层的 F-AP 带有缓存，采用了置信度传播算法，使每个 F-AP 可以独自做出缓存决定。根据其主要思想，将这种算法应用到本文的模型中，并进行仿真，便于同本文算法进行比较。

4.1 缓存总容量和缓存命中率的关系

本文分别改变缓存布置算法中缓存总容量的大小，对于本文算法、本文算法中只有某一层协作、部分协作式多层缓存、基于 BP 算法的单层缓存分别绘制了缓存命中率和缓存总容量的关系曲线，如图 2 所示。在本文算法中，当缓存总容量只占到文件总大小的10%时，命中率就已经达到了 51.8%，此时，对于部分协作式多层缓存和基于 BP 算法的单层缓存，本文提出的算法分别有 16.5%和 11.5%的提高，这是由于本文的缓存中同时存在垂直方向和水平方向的协作。而观察本文算法中只有某一层协作时的性能曲线，可以发现，最底层 F-AP 层（Tire 2）的协作对于系统命中率的增益最大，而簇间（Tire 1）协作的增益较小，但相对于部分协作式多层缓存，在缓存总容量为 10%时都至少有 12.4%的增益。仿真结果显示，当缓存容量在较小的范围内增长时，本文算法的命中率提高速率也是最快的，这是因为本文算法通过整个簇内用户的申请来估算流行度，这样得到的流行度要比另外 2 种算法更为精确。

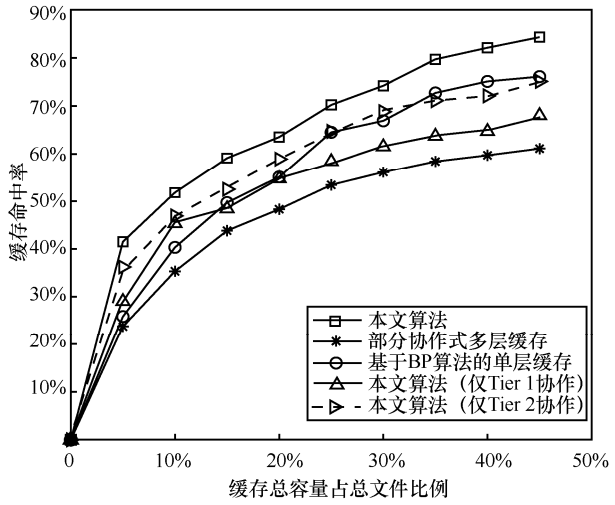


图 2 各算法缓存总容量与缓存命中率关系

4.2 缓存容量和后传链路负载关系

本文分别改变 3 种算法中缓存容量的大小，绘制了后传链路负载与缓存容量的关系曲线，如图 3 所示。在本文算法中，当缓存总容量为总文件大小的 15%时，后传链路负载已经降低为 40.9%，增益明显高于另外 2 种算法，这主要是因为本文的算法引入了更高的协作度，因此，更充分地利用了 CU 到簇的前传链路，有效降低了后传链路的负载。

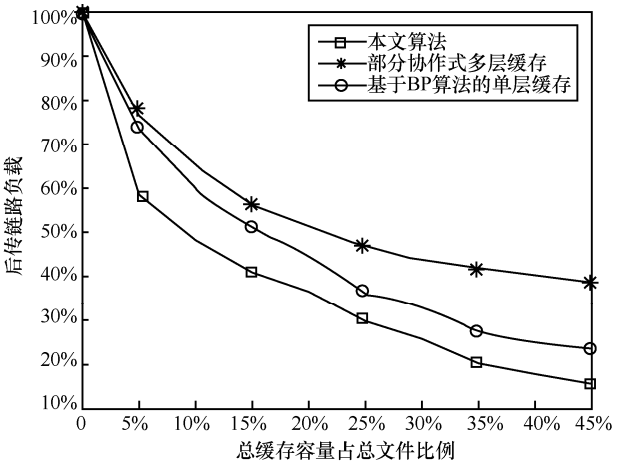


图 3 缓存总容量和后传链路负载关系

4.3 链路使用率和申请次数的关系

本文将缓存总容量固定为文件总大小的 5%，通过改变用户的申请次数，绘制了不同申请次数下，不同层内的链路占用率和申请次数的关系，如图 4 所示。从图 4 中可以看出，簇内传输链路的占用率始终高于簇间传输链路占用率，这是因为簇内 F-AP 层的水平协作、簇头和 F-AP 的垂直协作及簇间协作后传输到 F-AP 都需要占用簇内传输链路。在申请数量在 1.0×10^4 次以上时，两层的链路使用率都在 90%以上，这表示这种算法的协作机制十分有效；当申请数量继续增加时，两层的链路占用率最终都达到了 100%，协作度达到最大。但是由于缓存总容量只有文件总大小的 5%，无法缓存所有流行文件，所以，此后后传链路的负载将会快速增加。

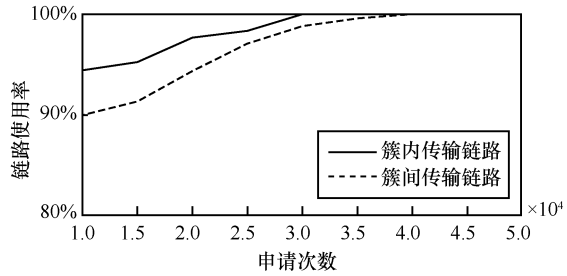


图 4 链路使用率和申请次数关系

5 结束语

本文主要研究了雾无线接入网中的缓存布置问题。为了降低网络的后传链路负载以及提高链路利用率，本文提出了 CU 层、F-AP 簇间、F-AP 簇内的多层协作缓存方法，使 F-AP 簇内缓存可以在

簇头的指导下协作，F-AP 簇间可以在 CU 的指导下协作，而 CU 能分布式地做出缓存决定，并提出了缓存布置策略。特别地，为了简化问题，在考虑了层内协作、层间协作、链路容量的基础上，本文将优化问题分解为三层内的缓存布置问题，并分别转化为背包问题，采用贪心算法求解。仿真结果显示，所提出的多层协作缓存方法可以显著降低后传链路负载。

附录　式(10)、式(11)与式(12)、式(13)等价的证明

证明　式(10)、式(11)到式(12)、式(13)的转化是等价的。

$$\begin{aligned}&\because x_{mk}^{f}, x_{mn}^{f} \in \{0,1\}\\&\therefore x_{mk}^{f} \vee x_{mn}^{f} - x_{mn}^{f} \leqslant x_{mk}^{f}\end{aligned} \tag{30}$$

当 $x_{mn}^{f}=0$ 时，式(30)取“=”，此时显然式(10)与式(12)等价。

当 x_{mn}^{f}=1 时，表明 F-AP$_n^m$ 已经缓存了文件 o_f，y_{kn}^{fm} 应为 0，而根据式(13)，此时 $y_{kn}^{fm}=0$，因此，式(10)与式(12)是等价的。

$$\begin{aligned}&\because x_{mk}^{f}, x_{mn}^{f} \in \{0,1\}\\&\therefore \bigcup_{k=1}^{N_m} x_{mk}^{f} - x_{mn}^{f} \leqslant 1 - x_{mn}^{f}\end{aligned} \tag{31}$$

当 $\exists x_{mk}^{f}=1, \text{for all } k$ 时，式(31)取“=”，此时，显然式(11)和式(13)等价。

当 $\forall x_{mk}^{f}=0, \text{for all } k$ 时，表示相邻 F-AP 都没有缓存文件 o_f，无法协作，y_{kn}^{fm} 应为 0。而根据式(12)，此时，$y_{kn}^{fm}=0$。因此，式(11)与式(13)是等价的。证毕。

参考文献：

[1] LI L, ZHAO G, BLUM R S. A survey of caching techniques in cellular networks: research issues and challenges in content placement and delivery strategies[J].IEEE Communications Surveys & Tutorials, 2018, 20(3): 1710-1732.

[2] PASCHOSG S, IOSIFIDISG, TAO M, et al. The role of caching in future communication systems and networks [J]. IEEE Journal on Selected Areas in Communications, 2018,36(6):1111-1125.

[3] LI X , WANG X , ZHU C , et al. Caching-as-a-service: virtual caching framework in the cloud-based mobile networks[C]// Computer Communications Workshops. IEEE, 2015: 372–377.

[4] JIANG Y, MA M, BENNIS M, et al. User preference learning based edge caching for fog radio access network [J]. IEEE Transactions on Wireless Communications, 2019,67(2):1268-1283.

[5] JIANG Y, HUANG W, BENNIS M, et al. Decentralized asynchronous coded caching design and performance analysis in fog radio access networks[J].IEEE Trans on Mobile Computing (Early Access), 2019(1): 1-12.

[6] CUI X, JIANG Y, CHEN X, et al. Graph-based cooperative caching in Fog-RAN[C]//IEEE ICNC Workshops. IEEE, 2018: 1-6.

[7] 钱志鸿,王雪. 面向 5G 通信网的 D2D 技术综述[J].通信学报,2016,37(7):1-14.
QIAN Z H, WANG X. Reviews of D2D technology for 5G communication network[J].Journal on Communications, 2016,37(7):1-14.

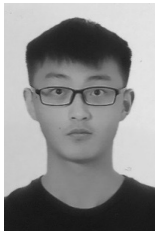
[8] MARTELLO S, TOTH P. Knapsack problems: algorithms and computer implementations[M]. New York: Wiley Press, 1990.

[9] ZHOU Y, ZHAO Z, LI R, et al. Cooperation-based probabilistic caching strategy in clustered cellular networks[J]. IEEE Communications Letters, 2017, 21(9):1.

[10] LIU J, BO B, ZHANG J, et al. Content caching at the wireless network edge: a distributed algorithm via belief propagation[C]// IEEE International Conference on Communications. IEEE, 2016: 1-6.

[作者简介]

蒋雁翔（1977− ），男，山东烟台人，博士，东南大学副教授，主要研究方向为 B5G/6G 移动通信理论与关键技术、无线大数据智能通信理论与关键技术。



夏骋宇（1997− ），男，江苏丹阳人，香港科技大学硕士生，主要研究方向为雾无线接入网中的边缘缓存技术等。