# Milking the Cache Cow With Fairness in Mind

Liang Wang, Gareth Tyson, Jussi Kangasharju, and Jon Crowcroft, *Fellow, IEEE, ACM*

***Abstract*— Information-centric networking (ICN) is a popular research topic. At its heart is the concept of in-network caching. Various algorithms have been proposed for optimizing ICN caching, many of which rely on collaborative principles, i.e. multiple caches interacting to decide what to store. Past work has assumed altruistic nodes that will sacrifice their own performance for the global optimum. We argue that this assumption is insufficient and oversimplifies the reality. We address this problem by modeling the in-network caching problem as a Nash bargaining game. We develop optimal and heuristic caching solutions that consider both performance and *fairness*. We argue that only algorithms that are fair to all parties involved in caching will encourage engagement and cooperation. Through extensive simulations, we show our heuristic solution, FairCache, ensures that all collaborative caches achieve performance gains without undermining the performance of others.**

***Index Terms*— Information-centric networking, resources allocation, game theory, algorithm design, modeling, optimisation.**

## I. Introduction

INFORMATION-CENTRIC Networking (ICN) [1] has been proposed to exploit the observation that much of today's Internet traffic is content distribution. ICN replaces the existing location-based Internet model with a content request/response model. One feature this enables is the capacity to cache content within the network. Whereas initial ICN caching approaches used traditional algorithms (*e.g.* Least Recently Used), there has been a number of novel proposals that attempt to specifically target ICN environments. These algorithms exploit things like inter-AS cooperation, request prediction and a priori topology maps to optimise performance [2]–[4]. A key outcome of this work has been the observation that collaborative caching usually outperforms locally optimised algorithms [4]–[7]. This is primarily caused by the nature of ubiquitous ICN caching, where nearby caches will often wastefully store the same objects [6]. To address this, a simple collaborative algorithm might involve two nodes strategically caching distinct objects [3], [8], [9].

Cache collaboration is therefore likely to play a role in any future ICN deployments [10]. In tandem, another trend we are witnessing is the *fragmentation* of cache ownership in the Internet. This involves large operators deploying separate cache infrastructures. Some of these exclusively host their own content (*e.g.* Google, Netflix), whilst other aggregate content from multiple content sources (*e.g.* Akamai, ChinaCache). Increasingly, we also see individual content producers placing their content across *multiple* third party caching infrastructures. For example, Twitter spreads content across three separate content delivery networks, whilst MultiCDN offers aggregation services to allow any website to achieve this easily. This trend adds an extra layer of complexity, as it means that (even on an intra-domain level) we are beginning to see multiple competing stakeholders operating caches that potentially serve the same content. This trend will likely be accelerated by the growth of network function virtualisation, which will allow anybody to "spin up" caches within a network (we already see the availability of virtual cache services, *e.g.* Fortinet Virtual Cache). Hence, ensuring the cache collaboration can also work in this setting will become increasingly important.

A more extreme example of this fragmentation is within the expanding number of wireless community mesh networks, *e.g.* Guifi [11]; these are deployed by groups of individuals who each contributes wireless routers (*e.g.* mounted on their property). In a community network, *every* router/cache is operated by a separate individual.

Consequently, we predict that future ICNs will use caches that are provisioned not just by network operators, but also various distinct stakeholders at strategic locations. These observations, however, have the potential to undermine the key tenets of caching in ICNs: What if caches operated by separate entities pursue policies that do not include collaboration, the storage of competitor's content or the serving of specific users? This is currently the situation online today, and it is unlikely to change with the advent of ICN. Despite this, most ICN collaborative algorithm assumes altruistic nodes that simply strive to reach a global optimum [3], [4], [6], [7], [9], [12]–[14]. Whereas this might be acceptable in scenarios where a single organisation operates all caches, it ceases to be suitable in scenarios where caches are owned and operated by multiple parties.

The reasons why a non-collaborative policy may be implemented are diverse. However, in this paper, we explore the topic from a utilitarian perspective. Intuitively, caches would wish to engage in a collaborative algorithm if they attain greater utility than if they were not to engage. This observation mandates some concept of *fairness*, where benefits are spread fairly across caches, and individuals are not expected to sacrifice personal performance by collaborating. Imagine, for instance, the above community network example; an

Manuscript received October 11, 2016; revised February 3, 2017 and March 22, 2017; accepted April 27, 2017; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. X. Liu. Date of publication June 9, 2017; date of current version October 13, 2017. *(Corresponding Author: Liang Wang.)*

L. Wang and J. Crowcroft are with University of Cambridge, Cambridge CB2 1TN, U.K. (e-mail: liang.wang@cl.cam.ac.uk).

G. Tyson is with the Queen Mary University of London, London E1 4NS, U.K.

J. Kangasharju is with the University of Helsinki, 00100 Helsinki, Finland

Digital Object Identifier 10.1109/TNET.2017.2707131

TABLE I
TABLE OF MAIN NOTATION USED IN THE PAPER

| Notation | Meaning |
|---|---|
| $G = (V, E)$ | a graph G of node set $V$ and edge set $E$ |
| $O$ | content set, $o_k$ represents the $k^{th}$ item |
| $O'$ | a reduced content set of $O$ by removing unpopular items |
| $\mathbf{s}$ | $s_k$ is the size of the $k^{th}$ content item $o_k$ |
| $\mathbf{w}$ | demand matrix, $w_{i,k}$ is the demand of $o_k$ on $v_i$ |
| $\mathbf{U}$ | utility vector, $U_i$ is the utility of node $v_i$ |
| $\mathbf{u^0}$ | initial disagreement vector, $u_i^0$ is the disagreement value of $v_i$ |
| $\mathbf{x}$ | $x_{i,k}$ represents the decision whether $v_i$ caches $o_k$ locally |
| $\mathbf{y}$ | $y_{i,j,k}$ represents the decision whether $v_i$ retrieves $o_k$ from $v_j$ |
| $\mathbf{\Omega}$ | solution space of all caching games, $\mathbf{\Omega^e}$ is the Pareto frontier |
| $\mathbf{\Psi}$ | solution space of all fair caching games |
| $\boldsymbol{\lambda}$ | dual variable associated with the constraint (7): $y_{i,j,k} \leq x_{j,k}$ |
| $\mathcal{L}(\cdot)$ | Lagrangian associated with the objective function (4) |
| $d(\cdot)$ | Lagrangian dual function of the objective function (4) |
| $h(\cdot)$ | subgradient of the dual function $d(\cdot)$ |
| $\mathbf{\Phi}$ | overall communication complexity at system level |
| $N_i$ | neighbourhood of $v_i$, average size is denoted as $\lvert\overline{N}\rvert$ |
| $N_i^+$ | set of nodes having $v_i$ in their neighbourhoods |
| $r$ | average search radius, $r_i$ uniquely defines $N_i$ of $v_i$ |
| $n_1$, $n_2$ | average size of one-hop and two-hop neighbourhood |
| $t$ | current iteration index while running subgradient descent |
| $\xi$ | $\xi_t$ is the step size of a subgradient algorithm at iteration $t$ |
| $\theta'$, $\theta$ | current utility improvement and the stopping threshold |

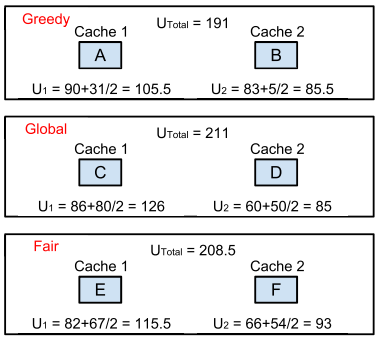
individual who sees his/her own performance consistently detrimented because of collaboration would (rationally) cease. We therefore argue that collaboration should be based on fairness, which may or may not reduce global performance. While a global optimum sounds attractive, we argue it is more important, from a practical perspective, that every node is better off by collaborating together than working alone. In this paper, we design a collaborative caching algorithm that embraces both high performance and fairness. Our focus is not to build a protocol that forces nodes to collaborate, or provides protection against malicious behaviour but, rather, to design underlying algorithms that can fairly share cache space across trusted collaborators. We first formulate the fair in-network caching problem as a Nash bargaining game (§III) before describing optimal algorithms for allocating objects to caches (§IV). We then propose a heuristic collaborative caching algorithm (§V) with fairness at its core: FairCache. Through extensive simulations, we show that FairCache achieves in excess of 90% accuracy compared to the optimal solution, at a fraction of the overheads (§VII). Importantly, we show that, when using FairCache, *all* nodes improve their performance via cooperation. It can be deployed across small subsets of collaborating caches or, alternatively, globally without change to design. We conclude by extracting key lessons learnt (§X).

## II. MOTIVATION AND SYSTEM MODEL

To underpin our design, we begin with a motivational example before outlining our system model. For convenience, Table I contains the notations used throughout the paper.

### A. Motivational Example

We use the simple toy caching system described in Figure 1 as a motivating example. Imagine two routers with a cache capacity of one object. They each serve a nearby set of users




Demand Matrix

| Object | Requests at Cache 1 | Requests at Cache 2 |
|---|---|---|
| A | 90 | 5 |
| B | 31 | 83 |
| C | 86 | 50 |
| D | 80 | 60 |
| E | 82 | 54 |
| F | 67 | 66 |

Fig. 1. A mini caching system with two caches and six objects. Three strategies (Greedy, Global and Fair) are presented.

and, consequently, it is desirable that they collaborate to decide which objects should be cached (*e.g.* to avoid storing the same object). To decide which object to store, the caches locally inspect the request rates they receive, depicted in Figure 1 (as a Demand Matrix). Intuitively, each cache would wish to selfishly optimise some concept of individual "utility". For simplicity, we measure their utility as the number of cache hits they get. We also allow nodes to redirect requests to the other cache; if a hit is attained there, a utility of 0.5 is given to the node performing the redirect (factored down due to the extra delay, overhead etc.). We consider three caching strategies:

*Case 1: Greedy Strategy*, where each cache locally and selfishly optimises its performance. As our comparison baseline, Greedy strategy is a perfect LFU that keeps track of all the objects. Cache 1 chooses to hold $A$ since it is the most popular content of demand 90, which leads to $U_1 = 90 + \frac{31}{2} = 105.5$. Similarly, Cache 2 caches $B$ which leads to $U_2 = 83 + \frac{5}{2} = 85.5$. Therefore, we have the aggregated utility $U_{Total} = U_1 + U_2 = 105.5 + 85.5 = 191$.

*Case 2: Global Strategy*, where each cache tries to maximise the aggregated utility $U_{Total}$ of the whole system. By caching $C$ and $D$ on Cache 1 and 2 respectively, $U_{Total}$ reaches its theoretical maximum, namely $U_{Total} = 126 + 85 = 211$. However, if we examine the individual performance and compare them to the Greedy Strategy, we notice that the increase in utility for Cache 1 results in a utility decrease for Cache 2.

*Case 3: Fair Strategy*, where caches attempt to collaborate fairly, in a way that does not reduce utility for any party. Cache 1 stores $E$ and Cache 2 stores $F$. Although this does not achieve the global optimum (*i.e.* the aggregated utility $U_{Total}$ drops from 211 to 208.5), it ensures that *both* caches improve their respective performance whilst also improving upon the local Greedy Strategy. This solution is Pareto efficient, and ensures both parties are incentivised.

The above reveals a stark mismatch. Attaining a global optimum often disadvantages some parties [15]. Thus, nodes that are unfairly exploited by other caches' redirects (at the cost of their own performance) are unlikely to continue collaboration: Caching should balance the need for high performance against the need for fair usage across caches.

### B. System Model

We model the network as a graph, $G = (V, E)$, where $V$ is the set of nodes and $E$ is the set of edges. $V$ could consist of all caches in a network or, alternatively, a subset of collaborating partners. These could be owned by one or more separate organisations. We follow an NDN [16] model, whereby hosts generate requests that get deterministically routed to sources that reply with content objects. Each node in the network, $v_i \in V$, is equipped with cache of size $C_i$. We denote $O$ as the global set of content objects. For each $o_k \in O$, we associate two parameters: $s_k$, which is the object size and $w_{i,k}$, which is its aggregated demand (requests per second) observed from all the clients connected to $v_i$. We focus on a subcategory of caching: collaborative algorithms. Because of resource constraints, we assume that nodes are limited in the number of nearby nodes they can cooperate with. We use $r_i$ to represent $v_i$'s search radius measured in hops. $r_i$ defines a neighbourhood for each $v_i$, which we denote as $N_i = \{v_j | l^*_{i,j} \leq r_i, \forall v_j \in V,\ v_i \neq v_j\}$, where $l^*_{i,j}$ measures the length of the shortest path between $v_i$ and $v_j$.

A collaborative caching algorithm can be decomposed into "caching decisions" and "retrieval decisions". These two parts solve "what to cache" and "where to fetch". The latter is necessary to allow nodes to redirect requests to other caches (opposed to forwarding it along the default route to the original source). This means that caches that do not locally store an object retain the flexibility to exploit objects stored elsewhere (*i.e.* collaboration). To model such a caching strategy, we use two vector decision variables: $\mathbf{x}$ and $\mathbf{y}$. $x_{i,k} \in \{0, 1\}$ denotes whether $v_i$ caches $o_k$, and $y_{i,j,k} \in \{0, 1\}, \forall i \neq j$ denotes whether $v_i$ retrieves the object $o_k$ from $v_j$. Formally, we say:

*Definition 1: A caching strategy for a network $G$ is a tuple of functions $(\mathbf{x}, \mathbf{y})$ where $\mathbf{x} : V \times O \to \{0, 1\}$ and $\mathbf{y} : V \times V \times O \to \{0, 1\}$. The family of all such tuples is denoted as $\Psi$, which represents the whole space of all caching strategies.*

*Definition 2: A caching strategy for a node $v_i$ is defined as $(\mathbf{x_i}, \mathbf{y_i})$, where $\mathbf{x_i} : \{v_i\} \times O \to \{0, 1\}$ and $\mathbf{y_i} : \{v_i\} \times V \times O \to \{0, 1\}$ are the partial functions of $\mathbf{x}$ and $\mathbf{y}$ with domains restricted to $\{v_i\} \times O$ and $\{v_i\} \times V \times O$ respectively.*

Note that $\times$ above represents the Cartesian product. We strive for a caching strategy that is (*i*) Pareto efficient; (*ii*) has well-defined fairness achieved amongst the nodes; and (*iii*) attains high performance. From a utilitarian perspective, this combination of attributes will lead to stable cooperation.

## III. Fundamentals of Bargaining Games

A bargaining game is a model for analysing how parties collaborate to obtain certain utility values. We model collaborative caching as a bargaining game, in which we aim to achieve both high performance (utility) and fairness. Ideally, a solution is considered fair if it satisfies certain axioms [17], [18]: (*i*) Pareto optimality; (*ii*) Scale invariance; (*iii*) Symmetry; (*iv*) Independence of the irrelevant alternatives; (*v*) Monotonicity. Nash proved that there is one unique solution which satisfies axiom (*i*)-(*iv*), termed the Nash Bargaining Solution (NBS) [17]. The NBS can be extended to multiple players. On the other hand, the Kalai-Smorodinsky Solution (KS) [18] satisfies axiom (*i*)-(*iii*)and (*v*). These two solutions lead to two fairness metrics. Compared to NBS, KS often does not have a closed-form expression. Hence, we focus on NBS.

### A. Game Definitions

In game theory, each node attempts to optimise its personal "utility". In caching, utility for a node, $U_i$, can be measured by the delay to respond to a client. Each cache aims to serve its clients with the lowest possible delay. Consequently, serving a request from the local cache produces the greatest utility, but redirecting a request to another nearby neighbour also increases it (rather than forwarding to the original source). As such, a selfish cache strives to maximise its utility through a combination of local caching and redirects to nearby neighbours.[1] Of course, if utility can be maximised solely through local caching then a node will cease to collaborate. NBS is an axiomatic solution for solving the following problem:

$$\max \prod_{v_i \in V} (U_i - u_i^0) \tag{1}$$

Eq. (1) is called the Nash product. As mentioned, $U_i$ is node $i$'s utility. $u_i^0$ is the initial disagreement value of $i$. The disagreement value is defined as the worst utility payoff a node would accept for collaboration. In practice, a node sets its disagreement value to the maximum value achieved by optimising locally as a standalone cache, *e.g.* using Least Recent Used. In the following, we give the formal definition of our in-network caching game and its solution.

*Definition 3: An in-network caching game is a tuple $(\mathbf{\Omega}, \mathbf{u^0})$, where $\mathbf{\Omega} \subset \mathbb{R}^{|V|}$ contains all the utility values obtainable via collaboration, and $\mathbf{u^0} \subset \mathbb{R}^{|V|}$ contains all the disagreement values leading to a negotiation breakdown.*

Let $\mathbf{\Omega^e} \subset \mathbf{\Omega}$ be the Pareto frontier of set $\mathbf{\Omega}$, *i.e.* the potential Pareto efficient solutions. We assume that $\mathbf{\Omega^e}$ is also a concave function with a closed compact convex domain. A game is considered fair iff its outcome is fair. Therefore, we have:

*Definition 4: A fair caching game is a game $(\mathbf{\Omega}, \mathbf{u^0})$ with a Nash bargaining solution, i.e. a function $f : \mathbf{\Omega^e} \to \mathbf{\Psi}$ such that $f(\mathbf{\Omega}, \mathbf{u^0}) = (\mathbf{x}, \mathbf{y})$ uniquely maximises $\prod_{v_i \in V} (U_i - u_i^0)$.*

By taking the logarithm of the objective function (1), we have $\ln(\max \prod_{v_i \in V} (U_i - u_i^0)) = \max \ln(\prod_{v_i \in V} (U_i - u_i^0))$. By taking the negation, NBS can be obtained equivalently by:

$$\max \sum_{v_i \in V} \ln(U_i - u_i^0) \Longrightarrow \min \sum_{v_i \in V} -\ln(U_i - u_i^0). \tag{2}$$

### B. Fairness Definitions

We argue that collaboration should follow the intuitive concept of fairness, such that all caches receive fair utility improvements through collaboration. This is critical to ensure that node owners do not feel exploited and do not disengage

[1] Here, we assume that each individual node selfishly optimises. However, our model can also support collective self interest amongst multiple nodes, *e.g.* if several caches are owned by a single organisation.

from the collaboration. Being Pareto efficient, alone, does not achieve this. To attain fairness, it is necessary to formalise the concept. Three well-defined fairness metrics are often referred to in the literature [19]–[21], *i.e. Egalitarian (EF)*, *Max-min (MF)* and *Proportional (PF)* fairness. *EF* pursues an equal amount of improvement on every node, which usually creates Pareto inefficiency (and is thus seldom used in practice). Both *MF* and *PF* have axiomatic foundations and are widely used, *e.g.* in traffic engineering. *MF* is a generalisation of KS, while *PF* is a generalisation of NBS. Thus, we only focus on *PF*:

*Definition 5: Proportional Fairness (PF): A caching strategy* $(\mathbf{x}^*, \mathbf{y}^*)$ *is PF iff* $\forall\ (\mathbf{x}, \mathbf{y}) \neq (\mathbf{x}^*, \mathbf{y}^*) \Rightarrow \sum_{v_i \in V} \frac{U_i - U_i^*}{U_i^* - u_i^0} < 0$.

A cache allocation is considered *PF* if the re-allocation of any object would decrease the proportional utility gain (from collaboration) of a node by *less* than the respective aggregated increase for others. For example, imagine an object is re-allocated from Cache 1 to Cache 2. It would not be fair if this re-allocation reduces Cache 1's utility by 20%, so that Cache 2 could increase its utility by just 3%. However, it would be considered fair if Cache 2 could increase its utility by 60%. Importantly, to be considered *PF* (and to incentivise uptake), it is necessary for *all* caches to improve their performance over local optimisation (*e.g.* Least Recently Used). Otherwise collaboration would immediately breakdown in favour of local algorithms. If this were to occur, each node would simply select its own preferred algorithm (*e.g.* LRU). However, in our caching games, *PF* is guaranteed by NBS (the proof is trivial and available in [22]). It is also trivial to show whenever a Pareto efficient solution achieves EF, it also achieves MF, *i.e.* a fair solution achieves all three fairness metrics in NBS given it is EF. We later show that being collaborative brings benefits to almost all nodes, indicating that the number of (rational) nodes who would revert to a local algorithm are very limited.

## IV. Solving a Fair In-Network Caching Game

We next devise both centralised and decentralised optimal solutions for achieving fair caching. We later use these to evaluate our heuristic solution, *FairCache*. We avoid presenting all the standard mathematical details but rather focus on the key mechanisms (remaining are available in [22]).

### A. Defining a Utility Function

We assume that a cache's utility is generated from serving its users' demand with low delay. For edge nodes, this demand comes directly from clients, whereas for core nodes this comes from their downstream customers. In either case, utility could be improved by a router using its local cache, or by redirecting a request to a nearby collaborative cache. Both improve delay compared to following the deterministic route to the origin. More precisely, $v_i$'s utility is defined as:

$$U_i = \sum_{o_k \in O} s_k w_{i,k} x_{i,k} + \sum_{o_k \in O} \sum_{v_j \in N_i} \frac{s_k w_{i,k}}{l_{i,j}^* + 1} y_{i,j,k} \tag{3}$$

Both terms show that the utility is a non-decreasing function of demand and content size. The second term shows that the utility of retrieving remote content decreases as the distance increases. It indicates that a node prefers fetching from the closest source to reduce latency and traffic footprint. Although this affine utility function is used throughout the paper, any other metric (*e.g.* bandwidth) or affine function can be used to model the utility without change to our model.

### B. Centralised Solution

We begin by outlining the optimal solution, which can be computed centrally (*e.g.* on a controller). Without loss of generality, we assume unit object size $s_k = 1$,[2] also let $l_{i,j} \triangleq l_{i,j}^* + 1$ for simplicity of expression. Plugging in Eq. (3) and Eq. (2), we define the optimisation problem as:

$$\min \sum_{v_i \in V} -\ln\Big(\sum_{o_k \in O} w_{i,k} x_{i,k} + \sum_{o_k \in O} \sum_{v_j \in N_i} \frac{w_{i,k}}{l_{i,j}} y_{i,j,k} - u_i^0\Big). \tag{4}$$

Subject to

$$\sum_{o_k \in O} x_{i,k} \leq C_i, \quad \forall v_i \in V \tag{5}$$

$$\sum_{v_j \in N_i} y_{i,j,k} \leq 1, \quad \forall v_i \in V,\ \forall o_k \in O \tag{6}$$

$$y_{i,j,k} \leq x_{j,k}, \quad \forall v_i,\ v_j \in V,\ o_k \in O \tag{7}$$

$$x_{i,k} \in \{0,1\}, \quad \forall v_i \in V,\ o_k \in O \tag{8}$$

$$y_{i,j,k} \in \{0,1\}, \quad \forall v_i,\ v_j \in V,\ o_k \in O \tag{9}$$

Constraint (5) means the content stored at a node cannot exceed its cache capacity. Constraint (6) simplifies the data scheduling and avoids requesting redundant content by constraining a node to retrieve a maximum of one complete object in a cache period. Constraint (7) says $v_i$ can retrieve $o_k$ from $v_j$ only if $v_j$ has cached it; it also says $v_i$ cannot get more than $v_j$ can offer. Constraints (8) and (9) impose the domain of decision variables.

The above optimisation problem is a typical *Integer Programming* program which is NP-Complete. By applying *Linear Programming relaxation*, we relax constraints (8) and (9) by letting $x_{i,k} \in [0,1]$ and $y_{i,j,k} \in [0,1]$. We later round up/down $x_{i,k}$ and $y_{i,j,k}$ to construct caching strategies. Such relaxation renders a suboptimal solution hence is considered as the lower bound of the actual performance. Regarding Eq. (4), since all the affine functions are log-concave, their composite with logarithmic functions preserves concavity. Thus, the problem (4) becomes a convex optimisation problem defined over a set of compact and convex constraints, which leads to the unique Pareto efficient solution, which is followed by the existence of the equilibrium in NBS by definition [17]. The centralised solution can be derived by applying standard convex optimisation techniques (see [22]). The solver needs the demand matrix of each cache, cache size, content object set, whole network topology *etc.* as inputs. The whole equation system has $3|O|\cdot|V|^2 + 2|O|\cdot|V| + |V|$ variables and the same

[2]In practice, object size could either be varied per-object or, alternatively, objects can be separated into smaller fixed size units

number of equations. Thus, the computation of solving such a large system is non-trivial.

### C. Distributed Solution

The optimal centralised solution has obvious drawbacks in its actual use: (*i*) it suffers from high computation complexity; (*ii*) it creates a single point of failure; and (*iii*) it is not adaptive under network dynamics. Hence, we next translate it into a distributed solution using decomposition techniques.

To solve an equation system, each node can be viewed as a subsystem. If they simply optimise locally, all the calculations in each subsystem are independent from those in others. However, due to collaboration, there are variables and constraints, which are referred to as *complicating variables and constraints* [23]. These make calculations interdependent and couple a subsystem with others. In problem (4), the only complicating constraint is (7).

To decompose Eq. (4), we apply Lagrangian dual relaxation. Lagrangian dual relaxation provides a non-trivial lower-bound of a primal. The difference between the dual and the primal is called the *duality gap*, which can be zero if certain conditions are met as we show below. The Lagrangian $\mathcal{L}(\cdot) : \mathbb{R}^{2|O||V|^2} \rightarrow \mathbb{R}$ associated with objective (4) is defined as follows:

$$\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}) = \sum_{v_i \in V} [-\ln(U_i - u_i^0) + \sum_{v_j \in N_i} \sum_{o_k \in O} \lambda_{i,j,k}(y_{i,j,k} - x_{j,k})]. \tag{10}$$

$\boldsymbol{\lambda} \succeq 0$ is the dual variable associated with constraint (7) of objective function (4). Then the Lagrangian dual function $d(\cdot) : \mathbb{R}^{|O||V|^2} \rightarrow \mathbb{R}$ is as follows:

$$d(\boldsymbol{\lambda}) = \inf_{\mathbf{x} \in X, \mathbf{y} \in Y} \mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}). \tag{11}$$

Given $\boldsymbol{\lambda}$, let $\mathbf{x}^*$ and $\mathbf{y}^*$ be the unique minimizers for the Lagrangian (10) over all $\mathbf{x}$ and $\mathbf{y}$. Then the dual function (11) can be rewritten as $d(\boldsymbol{\lambda}) = \mathcal{L}(\mathbf{x}^*, \mathbf{y}^*, \boldsymbol{\lambda})$. By maximising the dual function, we can reduce the duality gap. The Lagrangian dual problem of the primal (4) is defined as:

$$\max_{\boldsymbol{\lambda} \in \mathbb{R}^{|O||V|^2}} d(\boldsymbol{\lambda}) = \mathcal{L}(\mathbf{x}^*, \mathbf{y}^*, \boldsymbol{\lambda}). \tag{12}$$

The constraints for the dual are the same as those of the primal except constraint (7) which has already been included in the dual objective function (12). Because (4) is convex and all the constraints (5)(6)(8) and (9) are affine, Slater's condition holds given a solution exists, and the duality gap is zero. Thus, when the dual (12) reaches its maximum, the primal also reaches its minimum. The optimal solution for primal problem (4) can be derived from the optimal solution for dual problem (12).

After decomposition, each node $v_i$ now only needs to optimise its utility locally for a given $\boldsymbol{\lambda}$ by calculating:

$$\min \mathcal{L}_i\ (\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}) = -\ln(U_i - u_i^0) + \sum_{v_j \in N_i} \sum_{o_k \in O} \lambda_{i,j,k}(y_{i,j,k} - x_{j,k}).$$

We use the standard projected subgradient method [23] to derive the algorithm. Let $h(\boldsymbol{\lambda})$ and $\partial d(\boldsymbol{\lambda})$ denote the subgradient and subdifferential of dual function $d(\cdot)$ at point $\boldsymbol{\lambda}$ respectively. Then for every $h_{i,j,k} \in h(\boldsymbol{\lambda})$ we have:

$$h_{i,j,k} = y^*_{i,j,k} - x^*_{j,k} \Longrightarrow h(\boldsymbol{\lambda}) \in \partial d(\boldsymbol{\lambda}).$$

Gradient $\mathbf{h} \triangleq h(\boldsymbol{\lambda})$ points to the direction where $d(\cdot)$ increases fastest. In each iteration, node $v_i$ solves the local subsystem (13) to update the dual variable $\boldsymbol{\lambda}$. $t$ represents the $t^{th}$ iteration. $\xi_t$ is the step-size in the $t^{th}$ iteration which can be determined by several standard methods [23]. The projected subgradient method projects $\boldsymbol{\lambda}$ on its constraint (*i.e.* $\boldsymbol{\lambda} \succeq 0$) in each iteration, and we use $(\cdot)_+$ as a shorthand for the Euclidean projection. Eventually $\boldsymbol{\lambda}^{(t)} \rightarrow \boldsymbol{\lambda}^*$ when $t \rightarrow \infty$. The primal solution can be constructed from the optimum $\boldsymbol{\lambda}^*$. Combining the above, we refer to Eq. (13) as the *distributed optimal algorithm*:

$$\begin{cases} \mathbf{x}_i^{(t)}, \mathbf{y}_i^{(t)} = \arg\min_{\mathbf{x},\mathbf{y}} \mathcal{L}_i\ (\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}^{(t)}) \\ \mathbf{h}^{(t)} = -(\mathbf{x}_i^{(t)} - \mathbf{y}_i^{(t)}) \\ \boldsymbol{\lambda}^{(t+1)} = (\boldsymbol{\lambda}^{(t)} + \xi_t \sum_{v_j \in N_i \cup \{v_i\}} \mathbf{h}_j)_+ \end{cases} \tag{13}$$

*Theorem 1: Optimal algorithm converges to its optimum as the sequence* $\{\boldsymbol{\lambda}^{(1)}, \boldsymbol{\lambda}^{(2)}\ ...\ \boldsymbol{\lambda}^{(t)}\}$ *converges, if a diminishing step size is used such that* $\lim_{t\rightarrow\infty} \xi_t = 0$ *and* $\sum_{t=1}^{\infty} \xi_t = \infty$.

The above theorem guarantees convergence [22]. $\lambda_{i,j,k}$ can be viewed as the *"shadow price"* of transferring $o_k$ from $v_j$ to $v_i$, which is a "cost" for $v_i$ but an "income" for $v_j$. Given an ideal network, the distributed optimal algorithm will converge faster than the centralised one due to its parallel computations. It is worth emphasising that although the optimal algorithm above distributes the calculations over nodes, the overall computations are not reduced. At the same time, the communication cost increases due to exchanging "shadow price" information. However, the overall communication complexity remains the same as that of the centralised solution as we will later show in Section VI.

## V. FairCache: A Low-Complexity Heuristic Design

The distributed algorithm, although optimal, comes with high overheads. To mitigate this, we propose FairCache, a heuristic algorithm which does not require global knowledge regarding the content and network topology. We emphasise that FairCache is a decentralised algorithm for fairly sharing cache capacity across multiple *trusted* stakeholders. It is not intended to be a protocol, by which malicious behaviour (*e.g.* falsifying content demand) can be prevented. Hence, we assume trusted parties who faithfully execute the algorithm, much like is assumed within existing Internet routing schemes.

### A. Overview of Heuristics

To understand the rationality behind our heuristic, we first give a verbal explanation on the mechanisms of the optimal algorithm expressed in Eq. (13). Recall $\boldsymbol{\lambda}$ represents the shadow price of transmitting an object between two nodes. Each node hence maintains a list of prices for any given object from any given node. In each iteration of the optimisation,

a node tries to minimise its total cost using $\boldsymbol{\lambda}^{(t)}$. During the optimisation, the node adjusts its local caching strategy (via $\mathbf{x_i}$ and $\mathbf{y_i}$) and price list on other nodes (via $\boldsymbol{\lambda}$ and $\mathbf{h}$). Namely, a node may decide to cache an object if it brings significant improvement, or stop retrieving an object from another node due to high cost. Meanwhile the node adjusts the price on how to charge its neighbours by offering help. Then the node collects the price adjustments from *all others in the network* and updates its own list. The procedure continues until the performance converges based on certain well-defined criteria (as described next). Future updates are periodically shared to address changes in content popularity. As, generally, popularity changes are relatively slow to occur (hours, rather than minutes), this does not create considerable overheads. The mechanisms above indicate that we can approximate the optimal algorithm in the following ways:

*(i) Cut out unpopular content*: This approximation takes advantage of the highly skewed content popularity distribution. It is well-known that the popularity distribution has a long and heavy tail and most content fall into the tail. Removing the tail can significantly reduce the size of the exchanged messages. Meanwhile, the results will not be significantly influenced because of their marginal contribution to the overall utility (whilst also reducing signalling overheads dramatically). Thus, requests for unpopular content will be forwarded towards the origin (as with vanilla NDN [16]).

*(ii) Cut out distant nodes*: This approximation takes advantage of topological locality. Since the utility of retrieving distant content is a decreasing function of the hop count between two nodes, the value quickly diminishes as path length increases. It is more likely to find the requested content in nearby nodes due to content spatial locality [24], [25]; removing remote nodes should not have significant impact on the result.

*(iii) Reverse direction*: This approximation takes advantage of the behaviours of gradient methods. In the optimal algorithm, the neighbourhood ($r$) gradually shrinks from the network diameter to its optimum (as a result of minimising the cost function). However, most elements in $\mathbf{y_i}$ are already set to zero by the gradient method in the beginning phase of the optimisation. Exchanging messages between nodes that are not going to collaborate is a waste of resources. By growing the neighbourhood set outwardly, instead of shrinking it, we can avoid unnecessary message exchange.

### B. FairCache Algorithm

We embed the above heuristics in our algorithm, FairCache, presented in Algorithm 1. It takes several inputs. $\mathbf{w}$ is the local demand matrix. $r$ is for tracking the current number of hops that defines a node's neighbourhood radius. $\theta'$ is used for recording the utility improvement by increasing the radius from $r$ to $r+1$, while $\theta$ is the threshold below which FairCache should stop growing the neighbourhood size. $\boldsymbol{\lambda}$ is the list for tracking the shadow prices; this needs to be exchanged amongst nodes (via price adjustment $\mathbf{h}$) in a neighbourhood. Algorithm 2 shows how heuristic (*i*) is implemented to derive a reduced content set.

**Algorithm 1** Fair In-Network Caching (FairCache) on $v_i$

1: **Input:**
2: Demand matrix $\mathbf{w}$
3: Dual variables $\boldsymbol{\lambda}$
4: Search radius $r = 0$
5: Improvement threshold $\theta, \theta'$ $(\theta = 10^{-2}; \theta < \theta')$
6: **Output:**
7: Caching decision $\mathbf{x}_i$
8: Collaboration decision $\mathbf{y}_i$
9: **while** $\theta' \geq \theta$ **and** $r <$ network diameter **do**
10: $r = r + 1$; $t = 0$;
11: **while** $t < t_{stop}$ **do**
12: $\mathbf{x}_i$, $\mathbf{y}_i = \arg\min_{\mathbf{x},\mathbf{y}} \mathcal{L}_i(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda})$
13: $\mathbf{h} = \mathbf{y}_i - \mathbf{x}_i$; trim $\mathbf{h}$ for $\forall v_j \in N_i^+$
14: $\mathbf{h} = \mathbf{h} + \sum_{\forall v_j \in N_i} \mathbf{h}_j$
15: $\boldsymbol{\lambda} = (\boldsymbol{\lambda} + \xi \mathbf{h})_+$
16: $t = t + 1$
17: **end while**
18: Update $\theta'$ with current improvement
19: **end while**

**Algorithm 2** Construct Reduced Content Set $O_i'$ on $v_i$

1: **Input:**
2: Demand matrix $\mathbf{w}$
3: Complete content set $O$
4: **Output:**
5: Reduced content set $O_i'$
6: Sort $O$ in decreasing order based on $\mathbf{w}$;
7: set $O_i' = \emptyset$;
8: **for each** $o \in O$ **do**
9: **if** size ($O_i'$) + size ($o$) $\leq$ $v_i$'s cache capacity
10: **then** add $o$ to $O_i'$
11: **else** break
12: **end for each**

To apply the approximations, for node $v_i$, instead of making a complete price list, $\boldsymbol{\lambda}$, containing all the content and nodes in the network, node $v_i$ makes a partial $\boldsymbol{\lambda}$ which only includes: (*i*) the most popular content that can be fit into its local cache (*i.e.* heuristic (*i*)); and (*ii*) the nodes in the neighbourhood defined by $r$ (*i.e.* heuristic (*ii*)). It is possible that $v_i$ observes other content in the $\mathbf{h}_j$, collected from neighbours while $r$ grows (*i.e.* heuristic (*iii*)), then $v_i$ dynamically adds those content into its own $\boldsymbol{\lambda}$. $v_i$ can also remove items from $\boldsymbol{\lambda}$ if they are too expensive to retrieve. After local optimisation in each iteration, the price adjustment $\mathbf{h}$ will be trimmed before exchange by removing information that is not included in $\boldsymbol{\lambda}$; and removing the unchanged items, *i.e.* the zero values. Essentially, $v_i$ only exchanges the trimmed $\mathbf{h}$ within its neighbourhood and $\boldsymbol{\lambda}$ only contains the aggregated popular content in the neighbourhood. Algorithm 3 and 4 detail the logic in lines 13–14 in Algorithm 1 whenever sending and receiving price updates in each iteration. Obviously, these approximations render incomplete information (due to removing unpopular content and distant nodes). To handle

**Algorithm 3** On Sending the Price Update $\mathbf{h}_i$ on $v_i$

1: **Input:**
2: Neighbourhood $N_i^+$
3: Caching strategy $\mathbf{x}_i$, $\mathbf{y}_i$
4: set $\mathbf{h}_i = \mathbf{y}_i - \mathbf{x}_i$;
5: **for each** $v_j \in N_i^+$ **do**
6: construct $\mathbf{h}'_i$ from $\mathbf{h}_i$ based on $(O'_i \cup O'_j)$;
7: send $\mathbf{h}'_i$ to $v_j$;
8: **end for each**

**Algorithm 4** On Receiving All the $\mathbf{h}'_j$ on $v_i$

1: **Input:**
2: Neighbourhood $N_i$
3: Caching strategy $\mathbf{x}_i$, $\mathbf{y}_i$
4: set $\mathbf{h}_i = \mathbf{y}_i - \mathbf{x}_i$;
5: **for each** $v_j \in N_i$ **do**
6: receive $\mathbf{h}'_j$ from $v_j$;
7: set $\mathbf{h}_i = \mathbf{h}_i + \mathbf{h}'_j$;
8: **end for each**

the missing $\lambda_{i,j,k}$ in the local optimisation, we let missing $\lambda_{i,j,k} = \infty$ $(i \neq j)$, which indicates that the algorithm should neither exchange unpopular content nor exchange content with distant nodes.

Looking more closely, FairCache consists of two loops. The outer loop (lines 9–19) increases the search radius $r$ by one hop in each iteration. The outer loop stops when the current improvement, $\theta'$, drops below the threshold $\theta$ (*i.e.* $\theta' < \theta$) due to enlarging the neighbourhood. The inner loop (lines 11–17) finishes the local optimisation —- this is the calculation in Eq. (13) for the given neighbourhood defined by the current radius $r$. The communication overhead come from the operation in line 14 which collects the price adjustments $\mathbf{h}_j$ from the neighbourhood $N_i$. Line 15 adjusts the local shadow price list and updates the $\boldsymbol{\lambda}$ by removing or adding items.

## VI. FairCache Complexity Analysis

### A. Overview

The distributed solution requires exchange of control messages between nodes, whilst the centralised solution requires the distribution of the derived solution to *all* nodes (once it has been centrally computed). The optimal solution comprises of $|O|$ matrices of size $|V|^2$. The aggregation of each row $i$ (in all matrices) represents one specific optimal strategy for the corresponding node $v_i$ in the system. To transmit these rows to the nodes, the aggregated communication complexity is $\Theta(|O| \cdot |V|^2)$. This is the same for the distributed solution, as this performs the same computation (shared across multiple nodes). The rest of this section will delineate the complexity of FairCache. Table II presents an overview of the computational, communication, and space complexity of all three solutions.

### B. FairCache Heuristic Complexity

FairCache's computation, communication, and space complexities all correlate with the size of the content set and the neighbourhood set. Herein we evaluate the complexity savings of FairCache's three heuristics.

TABLE II
THE OVERALL AND (WORST) INDIVIDUAL COMPLEXITY OF THE THREE SOLUTIONS. THE COMMUNICATION COMPLEXITY OF THE CENTRALISED SOLUTION DOES NOT OCCUR DURING THE COMPUTATION BUT COMES AFTERWARDS WHEN DISTRIBUTING THE SOLUTION TO ALL NODES. THE DISTRIBUTED SOLUTION DOES NOT REDUCE ANY COMPLEXITY OF THE CENTRALISED SOLUTION BUT, INSTEAD, SHARES IT ACROSS MULTIPLE NODES. FAIRCACHE SIGNIFICANTLY REDUCES ALL THREE COMPLEXITIES BY APPLYING ITS HEURISTICS

| Type of complexity | Centralised | Distributed | FairCache |
|---|---|---|---|
| Computation | $\Theta(\lvert O\rvert\lvert V\rvert^2)$ | $\Theta(\lvert O\rvert\lvert V\rvert^2)$ | $\Theta(\lvert O'\rvert\lvert V\rvert\lvert\overline{N}\rvert)$ |
| Communication | $\Theta(\lvert O\rvert\lvert V\rvert^2)$ | $\Theta(\lvert O\rvert\lvert V\rvert^2)$ | $\Theta(\lvert O'\rvert\lvert V\rvert\lvert\overline{N}\rvert)$ |
| Memory space | $\Theta(\lvert O\rvert\lvert V\rvert^2)$ | $\Theta(\lvert O\rvert\lvert V\rvert^2)$ | $\Theta(\lvert O'\rvert\lvert V\rvert\lvert\overline{N}\rvert)$ |
| Individual (worst) | $\Theta(\lvert O\rvert\lvert V\rvert^2)$ | $\Theta(\lvert O\rvert\lvert V\rvert)$ | $\Theta(\lvert O'\rvert\lvert\overline{N}\rvert)$ |

Heuristic (*i*) is used to reduce the content set size ($|O'|$ from $|O|$). The level of reduction depends on both the popularity distribution of content and the percentage of requests we want to capture in the request streams. It is well documented that content popularity follows a Zipf-like distribution [26], [27]. If we assume a Zipf distribution with $\alpha = 1.0$, for a content set of $10^6$ equal sized objects, we are able to cover $72.8\%$ of the requests by caching only $2\%$ of $|O|$. In other words, $|O'| = 0.02 \cdot |O|$ indicates a $98\%$ reduction. Even if we only cache $1000$ objects (*i.e.* $0.1\%$ of $|O|$), we can still capture $52\%$ of requests, whilst leading to a significant reduction in $|O'|$, *i.e.* up to $99.9\%$.

Heuristic (*ii*) is used to reduce the size of collaboration neighbourhood, which is $|V|$ for the distributed solution. By investigating the FairCache algorithm, we can see that the communication complexity is due to exchanging $\mathbf{h}$ in order to update the local shadow price $\boldsymbol{\lambda}$, *cf.* the last equation in Eq.(13) and its corresponding line 14 in Algorithm 1. The overhead therefore consists of two parts. The first part is induced by replying the queries on $\mathbf{h}$ from the nodes having $v_i$ in their neighbourhood, namely $N_i^+$. The second part is induced by collecting $\mathbf{h}$ from the nodes in $v_i$'s own neighbourhood, namely $N_i$. Given that the communication complexity is measured by the number of exchanged messages, the complexity $\phi_i$ of node $v_i$ can be calculated as

$$\phi_i = c \cdot |O'| \cdot (|N_i^+| + |N_i|) \tag{14}$$

Scalar $c$ in Eq. (14) represents a constant factor for communication complexity, and can be understood as message size or other protocol-dependent factors. Now we show how to calculate the overall complexity $\Phi$ at system level from $\phi_i$ in the following.

Given any $v_j \in N_i$, a neighbourhood relation can be written as a tuple $(v_i, v_j)$. Calculating $\sum_{v_i \in V} |N_i|$ is equivalent to counting how many tuples there are in the whole system. Obviously, $v_i \in N_j \iff v_j \in N_i^+, \forall v_i, v_j \in V$, in other words, as long as there is a tuple $(v_i, v_j)$ for $N_i$, there must be a tuple $(v_j, v_i)$ for some $N_j^+$, and vice versa. Hence we can show $\sum_{v_i \in V} |N_i| = \sum_{v_i \in V} |N_i^+|$. Because one tuple $(v_i, v_j)$

represents one message exchange from $v_i$ to $v_j$, by using double counting technique, it is obvious that each message will be counted twice in the calculation if we try to aggregate all $\phi_i$: *i.e.* $(v_i, v_j)$ is in both $v_i$'s $N_i$ and $v_j$'s $N_j^+$. Therefore, overall complexity $\Phi$ can be calculated as half of the aggregated $\phi_i$ of all $v_i$ in $V$ as below.

$$\Phi = \frac{1}{2}\sum_{v_i \in V} \phi_i = \frac{c}{2} \cdot |O'| \cdot \Big(\sum_{v_i \in V} |N_i^+| + \sum_{v_i \in V} |N_i|\Big) \quad (15)$$

$$= c \cdot |O'| \cdot \sum_{v_i \in V} |N_i| \quad (16)$$

The intuitive explanation of eliminating $N_i^+$ in calculating $\Phi$ is that we only need to count the aggregated messages sent out from a node (*i.e.* the size of a node's own neighbourhood) since for each message there is always a correspondence in the network to receive it. Clearly, if we denote the average neighbourhood size as $|\overline{N}|$, then the system level communication complexity of FairCache is $\Phi = \Theta(|O'| \cdot |V| \cdot |\overline{N}|)$ as below.

$$\Phi = c \cdot |O'| \cdot \sum_{v_i \in V} |N_i| = c \cdot |O'| \cdot \sum_{v_i \in V} |\overline{N}| \quad (17)$$

$$= \Theta(|O'| \cdot |V| \cdot |\overline{N}|) \quad (18)$$

Similarly in the distributed solution Eq.(13), since each node has the whole network of size $|V|$ as its neighbourhood and uses the complete content set $O$ in the optimisation. Then its complexity can be calculated as $\Phi = \Theta(|O| \cdot |V|^2)$ by plugging the term $|\overline{N}| = |V|$ and $|O'| = |O|$ into Eq.(18). As we can see, the derived complexity is exactly the same as that of the centralised solution.

Heuristic (*iii*) is used to minimise the size of the neighbourhood by increasing, rather than decreasing, the neighbourhood size. As we can see from the analysis above, the complexity of FairCache is proportional to its average neighbourhood size $|\overline{N}|$. Reducing its size is therefore beneficial. The authors in [24] have proposed a neighbourhood model to calculate $|\overline{N}|$ in its closed form on any general network topologies. With its most general form, the overall complexity of the network of average $r$-hop neighbourhood can be calculated as

$$\Phi = |O| \cdot |V| \cdot n_1 \cdot \Big(1 + \Big[\frac{n_2}{n_1}\Big] + \Big[\frac{n_2}{n_1}\Big]^2 + \cdots + \Big[\frac{n_2}{n_1}\Big]^{r-1}\Big)$$

$$= \Theta\Big(|O| \cdot |V| \cdot \Big[\frac{n_2}{n_1}\Big]^r\Big) \quad \forall n_1, n_2 \in \mathbb{N}, n_2 > n_1$$

where $n_1$ and $n_2$ denote the average number of one-hop and two-hop neighbourhoods, and $r$ represents the search radius. If we assume $n_2 > n_1$ (which holds for all scale-free networks), the complexity of FairCache will grow exponentially if the search radius $r$ increases.[3] Hence, by ensuring a small $r$, Heuristic (*iii*) dramatically improves scalability. Importantly, it has been shown that $r$ is very small in most network optimisation problems [24] (confirmed in Section VII).

Besides the memory for storing the actual content objects, extra space is needed to store the input parameters of the optimisation problem (*i.e.* $\mathbf{w}$ and $\lambda$). The space complexity specifically refers to such extra space. Obviously, the space complexity is also decided by the size of content set and the size of neighbourhood. Hence FairCache on each node needs $\Theta(|O'||\overline{N}|)$ memory space to keep track of the information needed for optimisation. If we assume the average object size is 800 KB (much smaller than the average Youtube video size: 8 MB [26]) and each node has 8 GB memory installed, each node needs to maintain roughly $10^4$ objects. If we allocate 64-bit space to store the information for each object, given a neighbourhood of 10 nodes, even in the worst case wherein all these content sets are disjoint, each node only needs approximately 800 KB extra memory. However, spatial locality is very common [24] in content networks, which means there is a significant overlap regarding the stored content among nearby routers. If we assume that only 10% of objects are overlapping, the 800 KB can be further reduced to 651 KB (by summing up a geometric sequence to calculate). In the end, the extra memory overhead (*i.e.* the ratio between the memory space for parameters and memory space for objects) is less than 0.01%.

### C. Complexity Summary

Table II summarises our previous analysis of the complexity of the three different solutions. We wish to highlight the following key points:

- The computation and space complexity of the centralised and distributed solutions are identical. It is always $\Phi = \Theta(|O| \cdot |V|^2)$. However, because the distributed solution is able to share the complexity across each node in the network, in practice, it leads to a more scalable design and avoids the single point of failure.
- FairCache is able to reduce all complexities by limiting the sizes of the content set and its collaborative neighbourhood. The complexity of FairCache is only a small fraction of the distributed solution, more precisely $\frac{|O'|}{|O|} \cdot \frac{|\overline{N}|}{|V|}$. We have shown $|O| \gg |O'|$, also in reality $|V| \gg |\overline{N}|$ (as showed in [24]). We also confirm this experimentally in Section VII), hence the improvement is significant and improvements should grow as the network size increases.
- FairCache on each node needs $\Theta(|O'||\overline{N}|)$ memory space to keep track of the information needed for optimisation (*i.e.* $\mathbf{w}$ and $\lambda$). In practice this introduces only negligible overhead. As the average object size is bigger, the overhead becomes even smaller.

## VII. FairCache Evaluation

### A. Methodology

To evaluate FairCache, we perform extensive simulations using the publicly available LiteLab platform [29]. We use several topologies. First, we use real topologies collected by the Rocketfuel project [28]; namely, two ISP router-level topologies: Sprint (604 nodes, 2,279 edges) and AT&T (631 nodes, 2,078 edges). This embodies the use case where an individual network has allowed the deployment of

[3]Note that most natural graphs like Internet, ISP networks, and social networks are all scale-free [28]

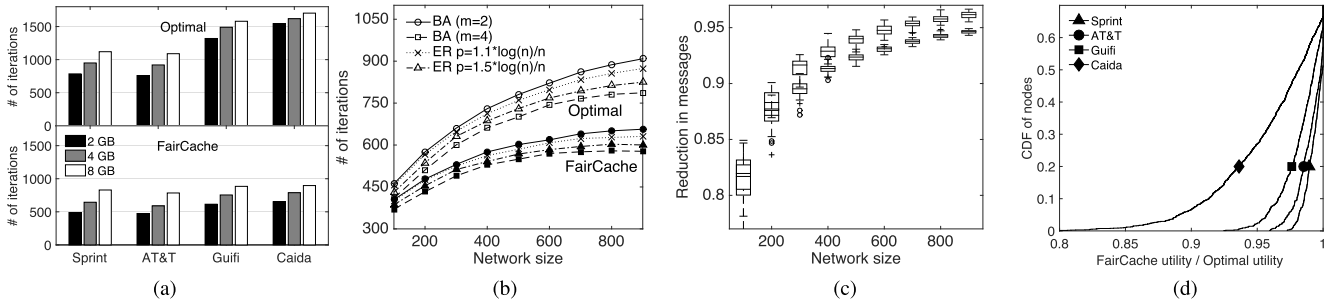



Fig. 2. Compared to the optimal algorithm, FairCache is more scalable on both real and synthetic networks. FairCache has a faster convergence rate and generates less traffic overhead than the optimal. Meanwhile FairCache achieves high accuracy. (a) Convergence on ISP networks. (b) Convergence on synthetic networks. (c) Reduction of traffic overhead. (d) The accuracy of FairCache.

caches in their network by multiple parties (*e.g.* via network function virtualisation). Second, we use traces from Guifi [11]. Guifi is the largest open wireless community mesh network in the world. It allows any user to purchase equipment and become part of the network. We use its core network topology in the Catalunya region (735 nodes, 1,059 edges). Third, we use the CAIDA ITDK trace [30] which takes a snapshots of the Internet AS topology every 24 hours. We use the 7/01/2017 trace (25,107 nodes, 49,458 edges). This captures the situation where each AS operates its own caches, but chooses to collaborate with neighbouring ASes. Fourth, to allow us to vary key graph parameters, we also generate synthetic networks based on two models: the Barabási-Albert (BA) model and the Erdős-Rényi (ER) model. Four parameter sets are used for these synthetic networks: $\{BA_1 : m = 2\}$, $\{BA_2 : m = 4\}$, $\{ER_1 : p = 1.1 \cdot \log(n)/n\}$ and $\{ER_2 : p = 1.5 \cdot \log(n)/n\}$. If there are multiple components in a synthetic network, we only use the largest one. $\log(n)/n$ in the setting guarantees there is only one single giant component in the network with high probability [31]. We have also analysed many recent AS-level topologies using the CAIDA ITDK traces [30]. Since most AS networks have an average degree of $4$, we use $BA_2$ configuration to reflect this fact. Therefore, we note that our synthetic networks represent both a per-router and per-AS topologies. For each topology we attach a single client to each edge router (*i.e.* with degree of 1). For example, this results in 161 clients in Sprint; 207 clients in AT&T; and 200 in Guifi. We then randomly select between 10 and 20 distinct routers to attach a source to. Each router is then allocated a given cache capacity, which we vary; the default is 4 GB, which is $< 0.1\%$ of the corpus. We select a low value to be representative of feasible cache capacity in a wide area network with a large corpus.

Using the above topologies, clients generate requests at each simulation tick, which are then routed through the network to either a content source or an intermediate cache based on the strategy employed. We base our content set on the Youtube trace from [26]. This contains 1,687,506 objects (average size is $8.0$ MB and aggregated size is $12.87$ TB). We use the view count information to fit a *Zipf*$(\alpha)$ distribution $(\alpha = 0.9537)$ to model the overall content popularity. To explore the impact of different request patterns, we also perform sensitivity analysis on $\alpha$. Throughout this section, we use our distributed optimal algorithm (*i.e.* Eq. (13)) as an optimal benchmark to compare FairCache against. Each result is averaged over 50 runs; errorbars are not plotted if they are sufficiently small ($< 5\%$).

### B. Scalability

We start by exploring scalability, measured by FairCache's convergence rate, *i.e.* how many rounds of message exchange it takes the algorithm to bootstrap. This happens once at initiation: future dynamics are addressed using periodic low cost updates that are algorithmically trivial. Figure 2a compares the convergence rates of the optimal (the upper figure) and FairCache (the lower figure) on the various topologies. The optimal needs significantly longer time to bootstrap than FairCache. Unsurprisingly, larger cache sizes also lead to a longer convergence time, as more state must be exchanged. To investigate how network size impacts the convergence rate, we use synthetic topologies with $4$ GB caches. The lines in Figure 2b are clearly divided into two groups: the upper one is the optimal (with hollow markers) and the lower one is FairCache (with filled markers). The convergence rate degrades as the network size grows. Importantly, though, the increase in convergence time only grows sub-linearly, stabilising at networks of size 1k; we experimented with topologies of up to 9k nodes to find consistent results.

We also measure FairCache's scalability by its traffic overheads. Clearly, it is undesirable to generate large amounts of control messages to bootstrap. In this experiment, we measure the aggregated size of control messages for both the distributed optimal and FairCache as $C_O$ and $C_F$ respectively. We then calculate the traffic reduction as $\frac{C_O - C_F}{C_O}$. Figure 2c presents a box plot of the results for both $BA_1$ (upper boxes) and $ER_1$ (lower boxes) topologies. It shows that FairCache is able to achieve over 80% traffic reductions, even on small networks of 100 nodes. As the network size increases, the benefit of using FairCache becomes more obvious. In a network of 900 nodes, FairCache attains 95% reductions. This equates to significant traffic volumes; in one iteration, a 500 node network with $10^3$ objects can save 887 MB of control traffic via FairCache (leaving only 66.8 MB). With FairCache, on average, each cache only introduces 136 KB traffic overhead in an iteration. We can therefore combine the above message overhead and convergence measurements to calculate the convergence time. If we configure the rate of

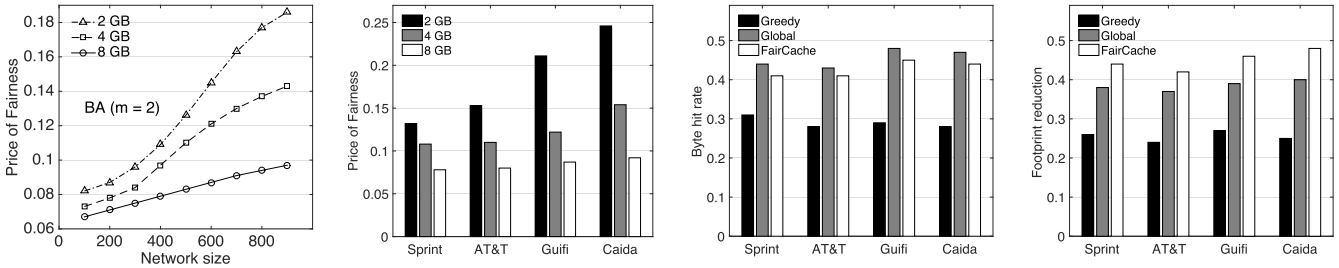



Fig. 3. FairCache achieves fairness by trading off some efficiency. However, a large cache size can effectively reduce PoF. In reality, FairCache is able to achieve very similar performance as Global, and is superior to Greedy in all cases. (a) PoF on synthetic networks. (b) PoF on ISP networks. (c) Byte hit rate, 4GB cache. (d) Footprint reduction, 4GB cache.

control messages to 100 KB/s, FairCache takes 11 minutes to bootstrap. This is just 4.6% of the time taken by the distributed optimal algorithm. Given a saturated 54 Mbit link, the FairCache control messages would therefore consume just 1.4% of bandwidth. Importantly, this is only a bootstrap process; changes in request patterns are addressed with low cost updates within each node's neighbourhood. Even in highly dynamic situations where demands are volatile, we can let nodes exchange demand information in the background while running FairCache. Once the new demand matrix is constructed, FairCache simply re-calculates the new solution. Recall our setting in Section VI where each node needs to maintain $651$ KB demand information. To guarantee FairCache continuously runs, a total of $651 \times 10$ KB needs to be exchanged among $10$ nodes within $11$ minutes, these updates constitute under $10$ KB/s. These values can be configured to reflect the operating environment (we anticipate that in many cases operators would transfer state at much higher rates). Overall, we believe these overheads are more than acceptable for the overall performance gains (§VII-E).

### C. Accuracy

FairCache significantly reduces the convergence time and messaging overhead of fairly allocating caching responsibilities. These improvements potentially come at the cost of accuracy (*i.e.* lower utility than optimal solution). We next inspect the accuracy sacrifice required to obtain these improvements.

To measure the accuracy of FairCache, we compare it against the optimal algorithm using multiple topologies of different sizes. We first run the optimal algorithm and measure the utility $U_i$ for every node $i$. Similarly, we run FairCache and measure the utility $U_i'$. We then calculate the accuracy of FairCache as its ratio to the optimal for every node, *i.e.* $\frac{U_i'}{U_i}$. Figure 2d plots the per node CDF of this ratio. We can see that FairCache achieves very high accuracy. For large networks like Guifi, all the nodes achieve an accuracy of over 92%. For medium size networks like Sprint, all the nodes have at least 97% accuracy and about 50% of the nodes reach 100% accuracy. Besides Figure 2d, we also measure the *aggregated* accuracy using $\frac{\sum_i U_i'}{\sum_i U_i}$. We find it is always above $95\%$ for medium-sized networks, whilst it decreases slightly for larger networks (*i.e.* 3% drop from Sprint to Guifi). Even for the very large CAIDA topology (over 25k nodes), the average accuracy is still $\approx 95\%$ with fewer than $6\%$ nodes that have an accuracy drop of $10\% \sim 20\%$. These findings are consistent across the other topologies. To validate that these benefits continue to be enjoyed by large topologies, we also repeated the experiments presented in Figure 2b on topologies ranging from 1k–9k nodes. Again, we find high levels of accuracy, stabilising at 95%.

TABLE III
KEY METRICS FOR INCREASING NETWORK SIZES (295 - 25k NODES). NETWORKS ARE REAL ISP TOPOLOGIES TAKEN FROM ROCKETFUEL AND CAIDA AS-LEVEL TRACE

| ASN | #nodes | #edges | Accuracy | PoF (2GB) | PoF (4GB) |
|---|---|---|---|---|---|
| #1755 | 295 | 544 | 97.3% | 16.5% | 11.3% |
| #3356 | 1620 | 6743 | 96.2% | 20.8% | 13.7% |
| #1221 | 2669 | 3181 | 95.4% | 22.6% | 14.9% |
| #2914 | 4670 | 7618 | 94.8% | 23.9% | 15.4% |
| #1239 | 7337 | 9924 | 94.7% | 24.3% | 15.6% |
| #7018 | 9430 | 11682 | 94.8% | 24.1% | 15.2% |
| Caida | 25107 | 49458 | 94.6% | 24.4% | 15.4% |

The results confirm the rationale behind the FairCache heuristics. The approximation introduces almost negligible degradation in the accuracy. The main reason is that the highly skewed content popularity means that the bulk of caching decisions are limited to the most popular objects. This means that FairCache can attain high accuracy without sharing information about all objects (unlike the optimal). Further, by localising interactions to nearby nodes, FairCache can scale-up without being overly affected by increasing network sizes.

### D. Price of Fairness

FairCache aims to realise fair collaboration amongst nodes, which could cause a degradation in aggregated global utility. We use the *Price of Fairness* (PoF) to measure the loss in utility. The PoF is calculated as the ratio between the aggregated utility loss of all nodes using FairCache and the global optimal that does not consider fairness [15]. A higher PoF value indicates a larger utility sacrifice.

Figure 3a and 3b plot the PoF results of using both real and synthetic networks with three cache sizes. Both figures convey the same information, which is that the PoF increases as network size increases. We experiment with both realistic and synthetic networks of up to 9k nodes (Table III summarises the results), to find that the PoF stabilises after reaching a

size of ≈3k nodes with a maximum PoF of 24% (and a maximum of 20% in the real topologies, *e.g.* Guifi). This is not negligible, but is likely not significant enough to dissuade caches that are interested in fairness from using FairCache. Interestingly, our results also show that increasing the cache size is an effective way to ameliorate the loss in efficiency. In Figure 3a, a 4 GB cache significantly improves the PoF. Using a 2 GB cache, the PoF increases by 11% when the network size increases from 100 to 900, whereas the PoF only increases by 3% if a 4 GB cache is used. Figure 3b shows similar properties with, for example, a 57% improvement in PoF when increasing the cache size from 2 GB cache to 4 GB in Guifi.

Overall, we believe that an average PoF of $< 8\%$ is a cost worth paying for those concerned by a need for fairness. Moreover, FairCache exhibits good scalability regarding both accuracy and PoF, as the results in Table III show. The accuracy only slightly degrades (1.5%) from 1000 nodes to 9000 nodes, and is always above 94.7%. The accuracy stabilises after 2000 nodes. Similarly, PoF stabilises after reaching around 3000 nodes. For the 2GB cache configuration, PoF is capped by 24.3%; 4GB by 15.6%; 8GB by 11.4% (not included in Table III due to space limit). The scalability of FairCache can be explained as follows: only small neighbourhoods play an important role in deciding a node's overall performance. As the size of the neighbourhood is relatively unaffected by the network size, this property always exists.

### *E. Caching Performance*

The previous section has shown that utility is reduced by considering fairness. Next, we explore performance from the perspective of traditional metrics: byte hit rate and footprint reduction. Hit rate is a conventional metric to measure saving on inter-domain traffic, whilst footprint reduction is the reduction on the product of traffic volume and distance.

We compare FairCache against two other strategies: (*i*) *Greedy*, which computes the local optimal for each cache without collaboration; and (*ii*) *Global*, which maximises the aggregated utility. Figure 3c and 3d plot the results on the real networks using 4 GB caches. Naturally, Figure 3c shows that Global achieves the best hit rates due the fact that it optimises the overall network. That said, FairCache only performs slightly worse, with a 5%–10% performance degradation. Compared to Greedy, FairCache is consistently superior with at least a 28% improvement. This shows that, regardless of fairness, FairCache can offer significant performance improvements over local algorithms (note that Greedy is the theoretical upper bound of algorithms such as Least Recently Used). When inspecting the traffic footprint reduction, performance is even higher. FairCache is superior in all networks. Although the reasons are intuitive for Greedy, which sees nodes locally optimising, it is more surprising in Global. The reason is that FairCache only requests from nearby caches (limited by $r$). In contrast, Global uses *any* node in the network. This increases hit rates, but results in more traffic.

To have a closer look how utility is spread across caches, we select the AT&T network and study the utility distribution

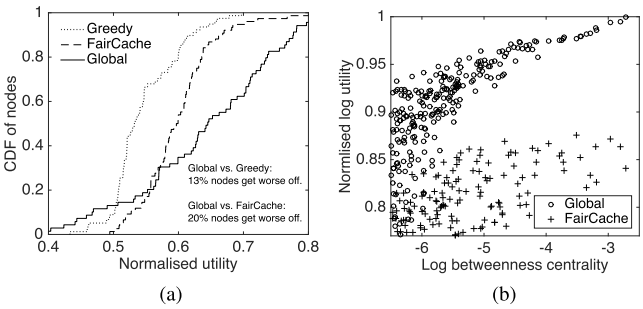



Fig. 4. Comparison of strategies on AT&T, 4 GB cache size. (a) Cumulative distribution of utilities. (b) Betw. centrality vs. utility.

in the network (*i.e.* how are traffic savings distributed across caches). Figure 4a plots the CDF of normalised utility values across each node (normalised by the top value per simulation). By comparing Greedy and FairCache, we see that *every* node is better off through collaboration using FairCache (note this is also the case across all other topologies and cache sizes). On the other hand, the Global strategy intersects with both Greedy and FairCache, *i.e.* some caches in Global get lower utility than Greedy. The area between the lines indicates the percentage of caches that are worse off due to global optimisation. The Global strategy leads to 13% of nodes getting worse off compared to Greedy, and 20% compared to FairCache. With Global, these nodes should rationally cease to cooperate. Again, regarding the aggregated utility, Global is only about 5% better than FairCache. Moreover, the CDF curve of Global is more stretched than that of FairCache, which indicates there are much larger variations in nodes' utilities when using the Global strategy, *i.e.* benefits are not evenly distributed.

Figure 4b shows the log-log plot of nodes' normalised utility as a function of betweenness centrality [6], [13]. Nodes with a high betweenness are core routers, whilst those with low betweenness are usually found at the edge. Interestingly, when nodes use the Global strategy, a node's utility strongly correlates with its position in the network: core nodes gain the highest utility. This is because the Global optimal tends to place all the popular (*i.e.* high value) content at the core to reduce duplicates — a theoretically attractive, but practically infeasible approach. In contrast, FairCache significantly weakens this correlation. This is beneficial as it means that utility is also increased at the edge caches. As well as improving fairness, it also reduces load in the backbone and provides consumers with lower delay access to object. This also contributes to FairCache's high traffic reductions, as hits are pushed closer to clients.

### *F. Sensitivity Analysis of Spatial/Content Locality*

FairCache's heuristics take advantage of highly skewed spatial and content popularity localities. A natural question is how these localities impact the algorithm. To explore this, we perform sensitivity analysis across these two parameters to measure the robustness of our heuristics. Here, we solely present the Guifi topology due to space constraints. The reason we select Guifi is that the dataset contains geographic coordinates of each node, allowing much more fine grained

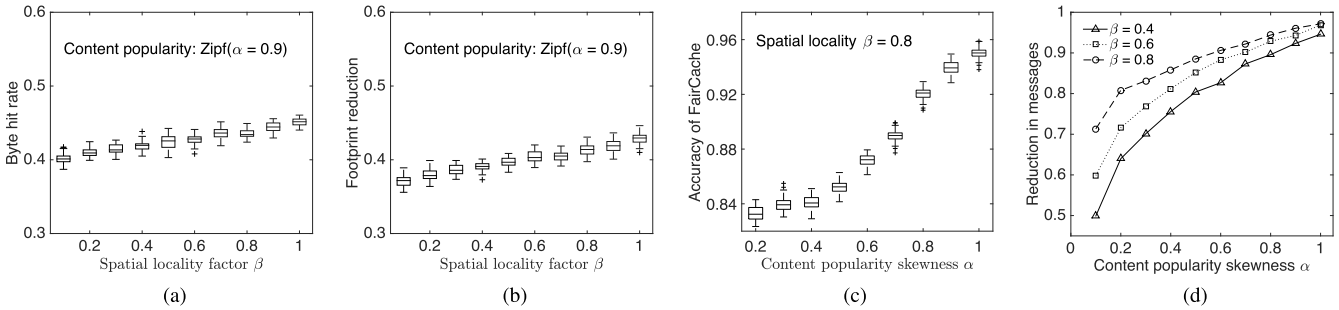



Fig. 5. Experiments on the Guifi network, 4 GB cache size. We vary both content popularity skewness $\alpha$ and spatial locality factor $\beta$ from 0.1 to 1. We observe a gradual and slow improvement in caching performance as spatial locality factor increases. Both spatial locality and content popularity skewness have significant impacts on the accuracy and the traffic reduction. (a) Byte hit rate vs. spatial locality. (b) FP reduction vs. spatial locality. (c) Accuracy vs. popularity skew. (d) Msg reduction vs. popularity skew.

analysis of spatial locality. We have confirmed that the results are representative of the other topologies.

We use a Hawkes process-based algorithm [24], [25] to generate a user request trace. The algorithm is controlled by two parameters: a *content popularity skewness* $\alpha$ and a *spatial locality factor* $\beta$. $\alpha$ controls the overall content popularity which follows *Zipf*$(\alpha)$. The spatial locality factor, $\beta = 0$, means the request pattern reduces to an *Independent Reference Model*; whilst $\beta = 1$ indicates very high spatial localisation (*i.e.* requests for an object often occur in the same locale).

First, we inspect their impact on the caching performance metrics. Figure 5 presents the results by varying both $\alpha$ and $\beta$ in $(0, 1]$. From Figure 5a and 5b, we observe a shallow improvement on byte hit rate and footprint reduction as $\beta$ increases. Specifically, they increase by only $6\%$ and $8\%$ respectively when increasing $\beta$ from $0.1$ to $1$. This suggests that spatial locality is not a critical requirement for FairCache.

On the other hand, the popularity skew, $\alpha$, has a more significant impact on the accuracy and message reduction of FairCache. Figure 5c shows that the average accuracy of FairCache improves from $85\%$ to $97\%$ by increasing $\alpha$ from $0.2$ to $1$. The speed of degradation of accuracy by decreasing $\alpha$ also slows down at certain point ($\alpha = 0.4$). The reason is because the general popularity distribution gets closer to a uniform distribution (due to a small $\alpha$). Thus, items are randomly requested, which means that each object has a similar utility when being cached. Interestingly, this means the overall utility of a cache will not vary much, though the solution can be quite different from the optimal one.

Last, we inspect the messaging overhead of running FairCache, presented as the reduction in comparison to the distributed optimal solution again. In Figure 5d, we see that both $\alpha$ and $\beta$ have a notable impact. Higher $\alpha$ and $\beta$ both result in lower overheads (*i.e.* higher reductions). The reason is that a smaller $\alpha$ value leads to a more uniform popularity distribution, which makes the demand matrices deviate more from each other, which further leads to larger exchanged messages for $\boldsymbol{\lambda}$ values. The smaller $\beta$ values have almost the same effect on demand matrices as that of $\alpha$. However, we also notice that $\beta$ has more significant impacts when $\alpha$ is small.

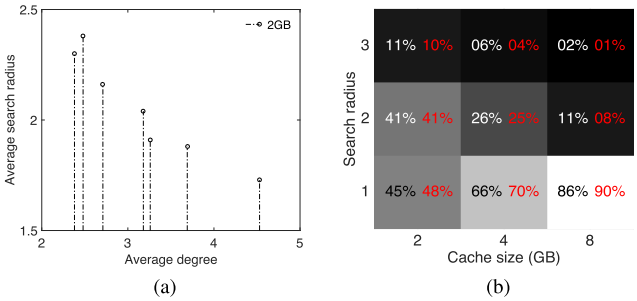



Fig. 6. The distribution of neighbourhood size (in terms of $r$) after FairCache converges. $(a)$ uses seven ISP topologies; there is a relatively strong negative correlation between nodes' average degree and their average neighbourhood size. $(b)$ uses Guifi topology with three cache configurations; the numbers in the grid show the percentage of the nodes and percentage of the control messages (in red).

### G. Neighbourhood Size Distribution

As previously stated in § VI, maintaining some neighbourhoods is critical for ensuring scalability. This is because (*i*) it dictates the communication overhead; and (*ii*) it justifies the effectiveness of the heuristic (*iii*). In this section, we present our empirical study on the neighbourhood size. We launch a number of experiments using the default setup, whilst varying the topology (as this is what dictates the neighbourhood size). We measure the neighbourhood sizes after FairCache converges (represented by the search radius $r$).

Figure 6a plots the average search radius, $r$, as a function of the average degree of the topology. We use seven $r1$-level ISP topologies (*i.e.* ISP router-level topology with one-hop clients included), with a 2GB cache size. As the average degree increases, $r$ decreases, thereby reducing network overhead. This is because network density increases, it becomes less necessary to create multi-hop neighbourhoods. Most important is the fact that even with a low degree, the search radius is very small on all topologies (around 2 hops).

Figure 6b presents results for experiments performed on the Guifi network. It shows a heatmap, which reveals the percentage of nodes that have a certain radius (across three cache sizes). Each point in the grid shows the percentage of nodes in a simulation that have a certain search radius (lighter colour means more), as well as the percentage of control

messages generated within each neighbourhood (shown in red). Figure 6b conveys two pieces of important information. First, most of the nodes end up with a neighbourhood of fewer than 3 hops. Even for the small 2GB cache sizes, $87\%$ of the nodes have no more than 2-hop search radius. Second, using larger cache size makes the distribution even more skewed, which further leads to even smaller search radius. When we increase the cache size from 2GB to 8GB, the percentage of nodes with a 1-hop radius increases from $45\%$ to $86\%$. With 8 GB cache, $97\%$ of nodes' final neighbourhoods are no more than 2 hops. In addition, we also provide the distribution of control messages in each $r$-hop neighbourhood, plotted in red colour in the same grid. As we can see, the traffic distribution is even more condensed within a very small neighbourhood, which indicates most interactions are between directly connected neighbours. We have confirmed that these results are mirrored across all other topologies.

## VIII. Discussion and Limitations

Deployment of FairCache raises a number of interesting questions. A key practical concern is the potential for parties to participate in FairCache in a malicious manner. This is a possibility as nodes are expected to report shadow prices, which they could be manipulated. FairCache is not intended to force stakeholders to collaborate, or to protect against cheating; hence, we have assumed that all nodes adhere to the FairCache algorithm. However, if deployed, such complexities would need to be addressed. In current inter-domain network protocols this problem is handled using out-of-band trust establishment, alongside signing authorities to bind autonomous systems to trusted identities (*e.g.* RPKI [32]). Equally, we envisage FairCache could rely on similar principles, in which legally formed (potentially transitive) collaboration agreements are underpinned by public key cryptography.

There are also a number of alternative practical concerns. Of course, highly dynamic content popularity may lead to volatile state information in caches, which can further increase the operational cost. We therefore implicitly assumed that the demand matrix is stationary, whereas in reality the demand will change over time. Recall that the space complexity (in Section VI) of storing an individual demand matrix in FairCache is $\Theta(|O'||\overline{N}|)$, hence the complexity of updating the whole matrix is bounded by $\Theta(|O'||\overline{N}|)$. Two other facts further help to reduce this overhead: (*i*) the spatial locality we have mentioned; and (*ii*) the various research [26] that has reported that such changes are gradual. Hence we can incrementally update the demand matrix to avoid unnecessary traffic. Furthermore, such occasional and incremental updates do not necessarily need to be synchronised especially when FairCache is deployed as an ever-running background process. In theory, the stale information may slow down the convergence in an iterative optimisation process. Yet, in practice, less frequent and incremental updates can ameliorate such impacts [23]. Moreover, due to the nature of FairCache, the updates mostly affect the local neighbourhood and their cascading effects will drastically decrease out of the neighbourhood.

Although we have focussed on building an efficient algorithm, we also acknowledge that FairCache treats storage as the key bottleneck. There are also a number of other constraints that could be included [33]. For instance, hardware bottlenecks can render servers useless even whilst in possession of content (*e.g.* CPU, I/O bus, congestion collapse). Thus, deployment would probably involve the introduction of such considerations into our model of fairness and utility. This could, for example, result in caches actively storing the *same* object in an attempt to share heavy load. Another simplifying assumption is the modelling of delay using hop count (like BGP); whereas this is a useful abstraction, it does not consider the variability introduced by realtime congestion. We consider this an acceptable sacrifice, as introducing such realtime considerations would introduce burdensome overheads. Another point worth highlighting is that we base caching decisions on per-object popularity counts, therefore introducing greater memory overheads than algorithms like Least Recently Used. We emphasise, however, that our heuristic removes all unpopular content, making such counts highly feasible.

## IX. Related Work

There are three key related areas of work: collaborative caching, content delivery networks (CDNs) and game theoretical studies of caching. Collaborative in-network caching has been proven as an effective methodology to improve system performance in various contexts [3]–[7], [9], [12], [34], even though edge caching has also been shown to be effective [35]. Previously proposed solutions are either limited by a centralised solver [5], [36] which makes scalability difficult, or limited by distributed heuristics [3], [4], [6], [7], [9], [12], [34], which neither guarantees a global optimum nor Pareto efficiency. FairCache is most related to the latter in that we do not guarantee a global optimum; however, we build on their contributions by introducing the concept of fairness and ensuring Pareto efficiency. Importantly, we also reveal the need for fairness to encourage engagement by cache operators. Recently, fair cache sharing also attracts enough attention in cloud computing and system research, *e.g.* [37] proposes FairRide using blocking to achieve isolation-guarantee and strategy-proofness properties.

The current solution used for Internet-scale content delivery are CDNs. They hold many similarities to ICNs [35], however, unlike our proposal, they are not collaborative entities. Typically, they are operated by distinct companies that deploy independent infrastructures. Some, like Akamai, sell their capacity to third party content providers (arguably a form of collaboration), whilst others build dedicated infrastructures for their own content (*e.g.* Google, Facebook, Netflix). Recent work within the IETF has endeavoured to support inter-CDN cache sharing [38], however, this only provides protocol support, rather than algorithms to decide when, where and how caches should be shared. Hence, our work is orthogonal, and could be applied to CDNs as a recent proposal in [39].

Game theory is an effective tool to analyse the effects of individual behaviours in a complex system; *e.g.* prior work [40] analyses the fairness achieved in bandwidth allo-

cation by coordinated and uncoordinated rate control over multiple links in peer-to-peer networks. Recent work [2], [36], [41]–[44] applies game theory to study in-network caching. In these papers, the caching problem is modelled as non-cooperative, pure strategic games and the equilibrium is analysed. Unlike us, this work takes a system-level utilitarian approach that aims to achieve a global optimum. In contrast, we focus on attaining fairness amongst nodes. More related to us is [2], [36], [42], which look at how selfishness drives nodes to act. These studies show how selfishness impact the equilibrium and efficiency in cache systems (measured by the *Price of Anarchy*). They also show that the global optimum is seldom achieved due to lack of coordination and nodes' inherent selfishness. Again, fairness is overlooked; we introduce this as an integral requirement of cooperation. To the best of our knowledge, no prior work has tried to solve the collaborative caching as a bargaining game and has devised a low-complexity heuristic to embrace both efficiency and fairness.

## X. Conclusion

To date, studies of collaborative caching have focussed on metrics such as hit rate, assuming that nodes are happy to contribute to achieving a global optimum. In this paper, we have argued that practical situations are unlikely to adhere to this model. Instead, caches operated by separate stakeholders will expect a reasonable level of *fairness*, where they are not penalised for cooperating with others. We began by delineating an optimal solution, which ensures no node attains lower utility by collaborating. To address its high complexity, we have also proposed a heuristic algorithm, FairCache, which we have shown achieves high performance at a fraction of the cost. Unlike past work, FairCache offers Pareto efficiency and proportional fairness, ensuring that *all* nodes are incentivised to collaborate. As well as helping to promote cooperation, our results show that proportional fairness plays a key role in balancing network traffic too. It helps maintain more hits at the edge, rather than globally optimal solutions that centralise hits in the core. We are not prescriptive in how FairCache is deployed and have ensured that it can be used either globally or amongst a subset of collaborating nodes. Hence, our key take-home message is that future collaborative caching designs should cease to assume purely altruistic cooperation and, instead, be explicitly built around the concept of fairness.

There are several lines of potential future work. First, we plan to build a wireline protocol to implement FairCache's design. The most prominent challenge in this regard is implementing a FairCache protocol that is robust against cheating nodes. This is a fascinating area of future work; currently, we assume trusted certified parties that do not lie, however, we wish to expand this to cover more dynamic arrangements in which trust can be formed on-the-fly. Note that this would not necessarily involve significant changes to FairCache — simply extra functions, *e.g.* key exchange. Clearly, this should be underpinned by a hardware implementation for exploring practical feasibility at line rates. There are various other real-world concerns that could also be integrated into FairCache too. For instance, dynamics regarding link availability, congestion and request patterns could be explored. This should extend to integrating new constraints (*e.g.* bandwidth, power, CPU), as well as alternate forms of fairness (*e.g.* bandwidth fairness, user-centric fairness). Lastly, we wish to expand FairCache to consider situations in which caches have external influences (*e.g.* business arrangements) that modify their behaviours. As of yet, little work has considered exogenous incentives that drive caching collaboration. We therefore see this as a fruitful line of study.

## References


[1] G. Xylomenos *et al.*, "A survey of information-centric networking research," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 1024–1049, 2nd Quart., 2014.

[2] V. Pacifici and G. Dán, "Selfish content replication on graphs," in *Proc. 23rd Int. Teletraffic Congr. (ITC)*, 2011, pp. 119–126. [Online]. Available: http://dl.acm.org/citation.cfm?id=2043468.2043488

[3] E. Rosensweig and J. Kurose, "Breadcrumbs: Efficient, best-effort content location in cache networks," in *Proc. INFOCOM*, Apr. 2009, pp. 2631–2635.

[4] J. Dai, Z. Hu, B. Li, J. Liu, and B. Li, "Collaborative hierarchical caching with dynamic request routing for massive content distribution," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 2444–2452.

[5] M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson, "Cooperative caching: Using remote client memory to improve file system performance," in *Proc. 1st USENIX Conf. Oper. Syst. Design Implement. (OSDI)*, Berkeley, CA, USA, 1994, Art. no. 19. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267638.1267657

[6] W. K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache 'less for more' in information-centric networks," *Comput. Commun.*, vol. 36, no. 7, pp. 758–770, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S014036641300025X

[7] S. Saha, A. Lukyanenko, and A. Yla-Jaaski, "Cooperative caching through routing control in information-centric networks," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 100–104.

[8] W. Wong, L. Wang, and J. Kangasharju, "Neighborhood search and admission control in cooperative caching networks," in *Proc. IEEE GLOBECOM*, Dec. 2012, pp. 2852–2858.

[9] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *Proc. 2nd ed. ICN Workshop Inf. Centric Netw. (ICN)*, New York, NY, USA, 2012, pp. 55–60. [Online]. Available: http://doi.acm.org/10.1145/2342488.2342501

[10] L. Wang, "Content, topology and cooperation in in-network caching," Dept. Comput. Sci., Univ. Helsinki, Helsinki, Finland, Tech. Rep. A-2015-1, Mar. 2015, pp. 472–484. [Online]. Available: http://urn.fi/URN:ISBN:978-951-51-0825-8

[11] D. Vega, L. Cerdà-Alabern, L. Navarro, and R. Meseguer, "Topology patterns of a community network: Guifi.net," in *Proc. IEEE 8th Int. Conf. Wireless Mobile Comput. Netw. Commun. (WiMob)*, Oct. 2012, pp. 612–619.

[12] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. IEEE INFOCOM*, Apr. 2010, pp. 1–9.

[13] L. Wang, S. Bayhan, and J. Kangasharju, "Effects of cooperation policy and network topology on performance of in-Network caching," *IEEE Commun. Lett.*, vol. 18, no. 4, pp. 680–683, Apr. 2014.

[14] L. Wang, S. Bayhan, and J. Kangasharju, "Optimal chunking and partial caching in information-centric networks," *Comput. Commun.*, vol. 61, pp. 48–57, May 2015.

[15] D. Bertsimas, V. F. Farias, and N. Trichakis, "The price of fairness," *Oper. Res.*, vol. 59, no. 1, pp. 17–31, 2011.

[16] V. Jacobson *et al.*, "Networking named content," in *Proc. 5th ACM Conext*, 2009.

[17] J. Nash and F. John, "The bargaining problem," *Econometrica*, vol. 18, no. 2, pp. 155–162, 1950. [Online]. Available: http://www.jstor.org/stable/1907266

[18] E. Kalai and M. Smorodinsky, "Other solutions to Nash's bargaining problem," *Econometrica*, vol. 43, no. 3, pp. 513–518, 1975. [Online]. Available: http://EconPapers.repec.org/RePEc:ecm:emetrp:v:43:y:1975:i:3:p:513-18

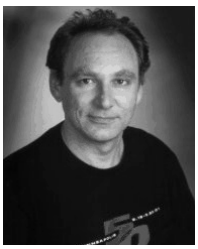
[19] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: Shadow prices, proportional fairness and stability," *J. Oper. Res. Soc.*, vol. 49, no. 3, pp. 237–252, Mar. 1998.

[20] H. Boche and M. Schubert, "Nash bargaining and proportional fairness for wireless systems," *IEEE/ACM Trans. Netw.*, vol. 17, no. 5, pp. 1453–1466, Oct. 2009.
[21] A. Muthoo, *Bargaining Theory With Applications*. Cambridge, U.K.: Cambridge Univ. Press, 1999.
[22] L. Wang, G. Tyson, J. Kangasharju, and J. Crowcroft. "FairCache: Introducing fairness to ICN caching-technical report," [Online]. Available: https://arxiv.org/abs/1412.0041
[23] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[24] L. Wang *et al.*, "Pro-diluvian: Understanding scoped-flooding for content discovery in information-centric networking," in *Proc. ICN*, New York, NY, USA, 2015, pp. 9–18. [Online]. Available: http://doi.acm.org/10.1145/2810156.2810162
[25] A. Dabirmoghaddam, M. M. Barijough, and J. Garcia-Luna-Aceves, "Understanding optimal caching and opportunistic caching at 'the edge' of information-centric networks," in *Proc. ICN*. New York, NY, USA, 2014, pp. 47–56. [Online]. Available: http://doi.acm.org/10.1145/2660129.2660143
[26] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I tube, You Tube, everybody tubes: Analyzing the world's largest user generated content video system," in *Proc. 7th ACM SIGCOMM Conf. Internet Meas. (IMC)*, New York, NY, USA, 2007, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/1298306.1298309
[27] X. Cheng, C. Dale, and J. Liu, "Statistics and social network of YouTube videos," in *Proc. 16th Int. Workshop Quality Service (IWQoS)*, Jun. 2008, pp. 229–238.
[28] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," in *Proc. SIGCOMM*, New York, NY, USA, 2002, pp. 133–145. [Online]. Available: http://doi.acm.org/10.1145/633025.633039
[29] L. Wang, A. Sathiaseelan, J. Crowcroft, and J. Kangasharju, "LiteLab: Efficient large-scale network experiments," in *Proc. 13th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2016, pp. 60–66.
[30] B. Huffaker, M. Fomenkov, and K. Claffy, "Internet topology data comparison," Cooperat. Assoc. Internet Data Anal. (CAIDA), La Jolla, CA, USA, Tech. Rep., May 2012.
[31] S. Bornholdt and H. G. Schuster, *Handbook of Graphs and Networks*, vol. 2. Hoboken, NJ, USA: Wiley, 2003.
[32] M. Wählisch *et al.*, "RiPKI: The tragic story of RPKI deployment in the Web ecosystem," in *Proc. 14th ACM Workshop Hot Topics Netw. (HotNets-XIV)*, New York, NY, USA, 2015, pp. 11-1–11-7. [Online]. Available: http://doi.acm.org/10.1145/2834050.2834102
[33] D. Perino and M. Varvello, "A reality check for content centric networking," in *Proc. ACM SIGCOMM Workshop Inf.-Centric Netw. (ICN)*, New York, NY, USA, 2011, pp. 44–49. [Online]. Available: http://doi.acm.org/10.1145/2018584.2018596
[34] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area Web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, Jun. 2000.
[35] S. K. Fayazbakhsh *et al.*, "Less pain, most of the gain: Incrementally deployable ICN," in *Proc. SIGCOMM*, New York, NY, USA, 2013, pp. 147–158.
[36] B.-G. Chun *et al.*, "Selfish caching in distributed systems: A game-theoretic analysis," in *Proc. 23rd Annu. ACM Symp. Principles Distrib. Comput. (PODC)*, New York, NY, USA, 2004, pp. 21–30. [Online]. Available: http://doi.acm.org/10.1145/101>1767.1011771
[37] Q. Pu, H. Li, M. Zaharia, A. Ghodsi, and I. Stoica, "FairRide: Near-optimal, fair cache sharing," in *Proc. 13th Usenix Conf. Netw. Syst. Design Implement. (NSDI)*, Berkeley, CA, USA, 2016, pp. 393–406. [Online]. Available: http://dl.acm.org/citation.cfm?id=2930611.2930637
[38] L. Peterson, B. Davie, and R. van Brandenburg, "Framework for content distribution network interconnection (CDNI)," Internet Eng. Task Force (IETF), Fremont, CA, USA, Tech. Rep. rfc7336, Aug. 2014. [Online]. Available: https://tools.ietf.org/html/rfc7336
[39] V. Pacifici and G. Dán, "Distributed algorithms for content allocation in interconnected content distribution networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 2362–2370.
[40] P. Key, L. Massoulié, and D. Towsley, "Path selection and multipath congestion control," *Commun. ACM*, vol. 54, no. 1, pp. 109–116, Jan. 2011. [Online]. Available: http://doi.acm.org/10.1145/1866739.1866762
[41] V. Pacifici and G. Dan, "Content-peering dynamics of autonomous caches in a content-centric network," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 1079–1087.
[42] N. Laoutaris, O. Telelis, V. Zissimopoulos, and I. Stavrakakis, "Distributed selfish replication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 12, pp. 1401–1413, Dec. 2006.
[43] G. Dán, "Cache-to-cache: Could ISPs cooperate to decrease peer-to-peer content distribution costs?" *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 9, pp. 1469–1482, Sep. 2011.
[44] M. Dehghan, L. Massoulie, D. Towsley, D. Menasche, and Y. C. Tay, "A utility optimization approach to network cache design," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.

**Liang Wang** received the B.Eng. degree in computer science and mathematics from Tongji University, Shanghai, China, in 2003, the M.Sc. and Ph.D. degrees in computer science from the University of Helsinki, Finland, in 2011 and 2015, respectively. He is currently a Research Associate with the Computer Laboratory, University of Cambridge, U.K. His research interests include system and network optimization, modeling and analysis of complex networks, information-centric networks, and distributed data processing.

**Gareth Tyson** received the Ph.D. degree from Lancaster University, U.K. in 2010. He is currently a Lecturer with the Queen Mary University of London. His research centers on system measurements and design, looking at topics ranging from network operations to social media. He received the Outstanding Reviewer Award at ICWSM'16. He serves as a Reviewer and Program Committee Member for a number of prominent journals, such as the IEEE/ACM TRANSACTIONS ON NETWORKING, the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, *ACM*, TRANSACTIONS ON MULTIMEDIA, and the IEEE TRANSACTIONS ON COMPUTERS.

**Jussi Kangasharju** received the M.Sc. degree from the Helsinki University of Technology in 1998, Diplome d'Etudes Approfondies from the Ecole Superieure des Sciences Informatiques, Sophia Antipolis, and the Ph.D. degree from the University of Nice Sophia Antipolis/Institut Eurecom, in 2002. In 2002, he joined the Darmstadt University of Technology, as Post-Doctoral Researcher, where he has been as an Assistant Professor, since 2004. Since 2007, he has been a Professor with the University of Helsinki.



**Jon Crowcroft** (SM'95–F'04) received the B.S. degree in physics from the University of Cambridge in 1979, and the M.Sc. degree in computing and Ph.D. degree from University College London in 1981 and 1993, respectively. He has been a Professor with the University of Cambridge since 2001, where has been involved in Internet support for multimedia communications. He is a Fellow of the Royal Society, the Association for Computing Machinery, the British Computer Society, the IET, and the Royal Academy of Engineering