FairCache: Introducing Fairness to ICN Caching

Liang Wang[1] Gareth Tyson[2] Jussi Kangasharju[3] Jon Crowcroft[1]
[1]University of Cambridge, UK [2]Queen Mary University London, UK [3]University of Helsinki, Finland

***Abstract*—Caching is a core principle of information-centric networking (ICN). Many novel algorithms have been proposed for enabling ICN caching, many of which rely on collaborative principles, *i.e.* multiple caches interacting to decide what to store. Past work has assumed entirely altruistic nodes that will sacrifice their own performance for the global optimum. In this paper, we argue that this assumption is flawed. We address this problem by modelling the in-network caching problem as a Nash bargaining game. We develop optimal and heuristic caching solutions that explicitly consider both performance and *fairness*. We argue that only algorithms that are fair to all parties 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.**

I. Introduction

Information-Centric Networking (ICN) [2], [3] has been proposed to ameliorate the pressure on current network infrastructures. It replaces the existing end-to-end Internet model with a content request/response model. Many benefits have been suggested, including superior performance, better support for mobility and reduced overheads. A fundamental enabler of these attributes is in-network caching. Whereas initial ICN caching studies used traditional algorithms (*e.g.* Least Recently Used), there has been a flurry 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 [5], [6], [26].

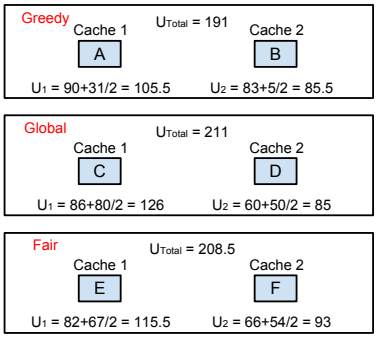
A key outcome of this work has been the observation that collaborative caching usually outperforms locally optimised algorithms [4], [6]–[8]. This is primarily caused by the nature of ubiquitous ICN caching, where nearby caches will often wastefully store the same objects [7]. To remedy this, a simple collaborative algorithm between two nearby nodes may involve strategically caching distinct objects [5], [9]. Any future ICN deployments are therefore likely to involve some form of cache collaboration. In tandem, we are witnessing a fragmentation of cache ownership in the live Internet, with large content providers deploying separate infrastructures (*e.g.* Google, Facebook, Netflix). A more extreme example of this fragmentation is within the expanding number of wireless community mesh networks, *e.g.* Guifi [19]; these are constructed by groups of individuals who each contribute wireless routers, usually mounted on their property on a city-wide scale. In a community network, *every* router/cache is operated by a separate individual. Hence, we predict that future ICNs will use caches that are provisioned not just by network operators, but also various distinct stakeholders at strategic in-network 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, nearly every ICN collaborative algorithm proposed assumes altruistic nodes that simply strive to reach a global optimum [5]–[10].

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 individual who sees his/her own performance consistently detrimented by 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 delineating optimal algorithms for allocating objects to caches (§IV). We then propose FairCache, a heuristic collaborative caching algorithm (§V) with fairness at its heart. We show that FairCache achieves in excess of 90% accuracy compared to the optimal solution, at a fraction of the overheads (§VI). 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 (§IX).

II. Motivation and System Model

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 and, consequently, it is desirable that they collaborate to decide



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.

which objects should be cached (*e.g.* to avoid storing the same object twice). To decide which object to store, the caches inspect the request rates they receive, also 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. 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 [21]. 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 assume a network whose topology can be represented 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 organisataions. We follow an NDN [2] 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 that could be stored in the cache. 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 uniquely 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.

Thus, a collaborative caching algorithm can be decomposed into "caching decisions" and "retrieval decisions". These two parts solve "what to cache" and "where to fetch", respectively. The latter is necessary to allow nodes to redirect requests to other caches that have agreed to store objects on their behalf (opposed to forwarding it along the default route to the original source). 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 Ψ, 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 players 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 [11], [12]: (*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 (1)-(4), termed the Nash Bargaining Solution (NBS) [11]. The NBS can be extended to multiple players. On the other hand, the Kalai-Smorodinsky Solution (KS) [12] satisfies axiom (1)-(3) and (5). 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* $f : \mathbf{\Omega^e} \rightarrow \Psi$ *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 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 [13]–[15], *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 focus on *PF*, defined as:

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. In our caching games, *PF* is guaranteed by NBS; the proof is trivial and is available in [1].

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*. Due to space limitations, we cannot present all the derivation and algorithm details but rather focus on the key mechanisms. Please refer to our online technical report [1] for further details such as the fairness proof, convergence proof, complexity analysis etc.

A. Defining a Utility Function

In this paper, 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, uility 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. We leave experimentation with these other metrics to future work.

[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.

B. Centralised Solution

We begin by outlining the optimal solution, which can be computed centrally (*e.g.* on a software 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 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. This is clearly a subset of the full range of practical constraints that a cache infrastructure may embody; for example, a cache server may suffer from limitations regarding availability of processor cycles, network capacity or bus speed. We leave these to future work, as currently content availability is the most prominent bottleneck.

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 $y_{i,j,k} \in [0,1]$, so the original problem can be transformed into a linear programming problem. 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, its composite with logarithmic function preserves concavity. Thus, the problem (4) is a convex optimisation problem. The centralised solution can be derived by applying standard convex optimisation techniques (see [1]). The solver needs the demand matrix of each cache, cache size, content object set, whole network topology and *etc.* as inputs. The whole equation system has $3|O|\cdot|V|^2+2|O|\cdot|V|+|V|$ variables and the same number of equations. Thus, its computation 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* [16]. 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:

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

$\boldsymbol{\lambda} \succeq 0$ 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 [16] 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

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

(13) to update the dual variable $\boldsymbol{\lambda}$. t represents the t^{th} iteration. ξ_t is the step-size in the t^{th} iteration which can be determined by several standard methods [16]. 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 $\boldsymbol{\lambda}^{(t)} \to \boldsymbol{\lambda}^*$ when $t \to \infty$. 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* $\{\boldsymbol{\lambda}^{(1)}, \boldsymbol{\lambda}^{(2)} \ldots \boldsymbol{\lambda}^{(t)}\}$ *converges, if a diminishing step size is used such that* $\lim_{t\to\infty} \xi_t = 0$ *and* $\sum_{t=1}^{\infty} \xi_t = \infty$.

The above theorem guarantees convergence [1]. $\lambda_{i,j,k}$ 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. It is worth emphasising that although the optimal algorithm distributes the calculations over nodes, the overall computations are not reduced. At the same time, the communication cost increases significantly due to exchanging "shadow price" information.

V. FairCache: A Low-Complexity Heuristic Design

The computational cost of the distributed optimal algorithm comes at the price of high communication overheads. To address this, we propose FairCache, a heuristic algorithm which does not require global knowledge of content and network. 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.

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 periodicly 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.

Algorithm 1 Fair in-network caching (FairCache) on v_i

Input:
  Demand matrix $\mathbf{w}$
  Dual variables $\boldsymbol{\lambda}$
  Search radius $r = 0$
  Improvement threshold θ, θ' $(\theta = 10^{-2}; \theta < \theta')$
Output:
  Caching decision $\mathbf{x}_i$
  Collaboration decision $\mathbf{y}_i$
while $\theta' \geq \theta$ **and** $r <$ network diameter **do**
  $r = r + 1$; $t = 0$;
  while $t < t_{stop}$ **do**
    $\mathbf{x}_i$, $\mathbf{y}_i = \arg\min_{\mathbf{x},\mathbf{y}} \mathcal{L}_i(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda})$
    $\mathbf{h} = \mathbf{y}_i - \mathbf{x}_i$; trim $\mathbf{h}$
    $\mathbf{h} = \mathbf{h} + \sum_{\forall v_j \in N_i} \mathbf{h}_j$
    $\boldsymbol{\lambda} = (\boldsymbol{\lambda} + \xi \mathbf{h})_+$
    $t = t + 1$
  end while
  Update θ' with current improvement
end while

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 [2]).

***(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 [22]; 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 gradually shrinks from the network diameter (r) 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. θ' is used for recording the utility improvement by increasing the radius from r to $r+1$, while θ 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.

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 optimising 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. Obviously, these approximations render incomplete information (due to removing unpopular content and distant nodes). To handle 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 optimisation algorithm should neither exchange unpopular content nor exchange content with distant nodes.

Looking more closely, FairCache consists of two loops. The outer loop (line 9–19) increases the search radius r by one hop in each iteration. The outer loop stops when the current improvement, θ', drops below the threshold θ (*i.e.* $\theta' < \theta$) due to enlarging the neighbourhood. The inner loop (line 11–17) finishes the local optimisation which is the calculations 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 Evaluation

A. Methodology

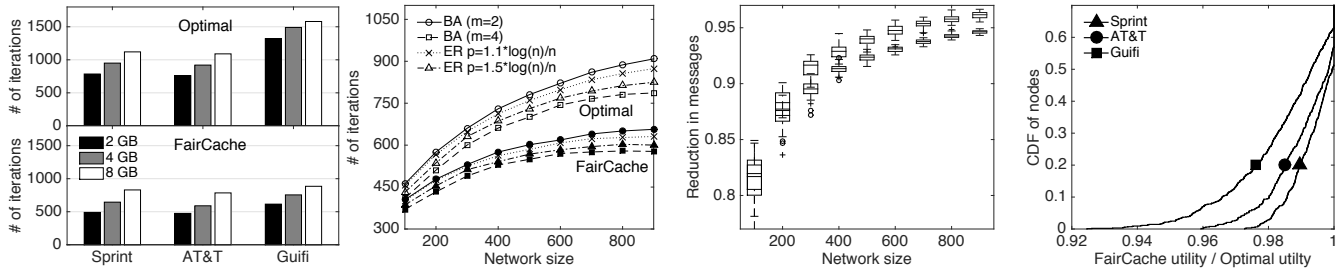
To evaluate FairCache, we perform extensive simulations using the publicly available LiteLab platform [17]. We use several topologies. First, we use real topologies collected by the Rocketfuel project [18]; namely, two ISP router-level topologies: Sprint (604 nodes, 2,279 edges) and AT&T (631 nodes, 2,078 edges). Second, we use traces from Guifi [19]. 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, 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 $\{ER_2 : p = 1.5 \cdot \log(n)/n\}$. For each topology we attach a single client to each edge router (*i.e.* with degree of 1). 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 [20]. 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*(α) distribution ($\alpha = 0.9537$) to model the overall content popularity. To explore the impact of different request patterns, we also perform sensitivity analysis on α. 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 real ISP and Guifi 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 sublinearly, stabilising at networks of size 1k; we experimented with topologies of upto 9k nodes to find consistent results.

We also measure FairCache's scalability by its traffic overheads. Clearly, it is undesirable to require large amounts of control message volume 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) Convergence on ISP networks. (b) Convergence on synthetic networks. (c) Reduction of traffic overhead. (d) The accuracy of FairCache.

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 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 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 consume just 1.4% of traffic. 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, these updates constitute under 10 KB/s.

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). 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. We also measure the aggregated accuracy using $\frac{\sum_i U_i'}{\sum_i U_i}$. The aggregated accuracy is always above 95% and may slightly degrade as the network size increases (*i.e.* 3% drop from Sprint to Guifi). These findings are consistent across the other topologies. To validate that these benefits continue to be enjoyed by large topologies, we repeated the experiments presented in Figure 2b on topologies ranging from 1k–9k nodes. Again, we find very high levels of accuracy, stabilising at 95%.

The results confirm the rationale behind the heuristics used by FairCache. 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 requiring to share information about all objects (unlike the optimal). Further, by localising interactions to neighbouring nodes, FairCache can scale-up easily, without be overly effected by increasing network sizes.

D. Price of Fairness

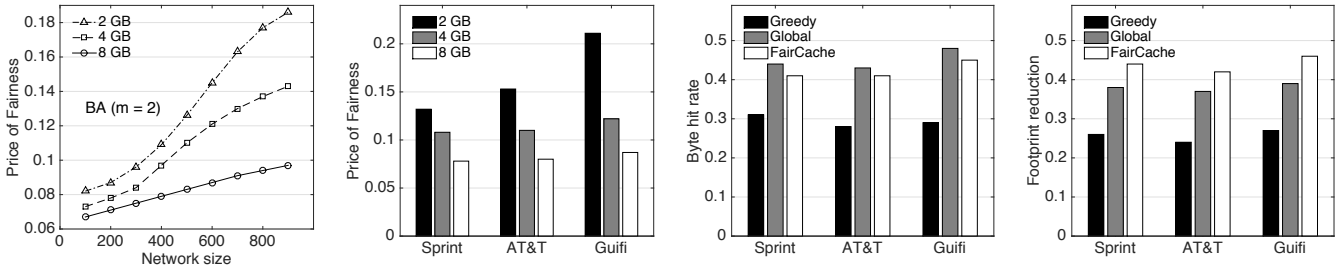
FairCache aims to realise fair collaboration amongst nodes, which could cause a degradation in aggregated global utility. We use the *Price of Fairness* (PoF) to measure the loss in utility. The PoF is calculated as the ratio between the aggregated utilities of all nodes using FairCache and the global optimal that does not consider fairness [21]. 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 synethic networks of up to 9k nodes, to find that the PoF stabilises after reaching a size of ≈3k nodes, with a maximum of PoF of 24% (and a maximum of 20% in the real topologies, *i.e.* Guifi). This is not negligible, but is likely not significant enough to dissuade caches who 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. Despite of the promising results, we acknowledge that further investigations are needed to back up our observations.

Overall, we believe that an average PoF of $< 8\%$ is a cost worth paying for those concerned by a need for fairness.

E. Caching Performance

The previous section has shown that utility is reduced by considering fairness. Next, we explore performance from the



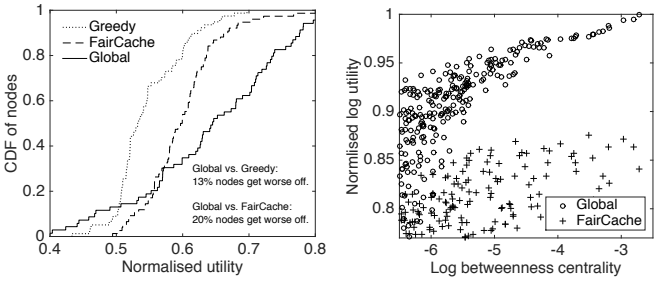
(a) PoF on synthetic networks. (b) PoF on ISP networks. (c) Byte hit rate, 4GB cache. (d) Footprint reduction, 4GB cache.

Fig. 3: FairCache achieves fairness by trading off some efficiency. However, 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.

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 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.



(a) Cumulative distribution of utilities. (b) Betw. centrality vs. utility.

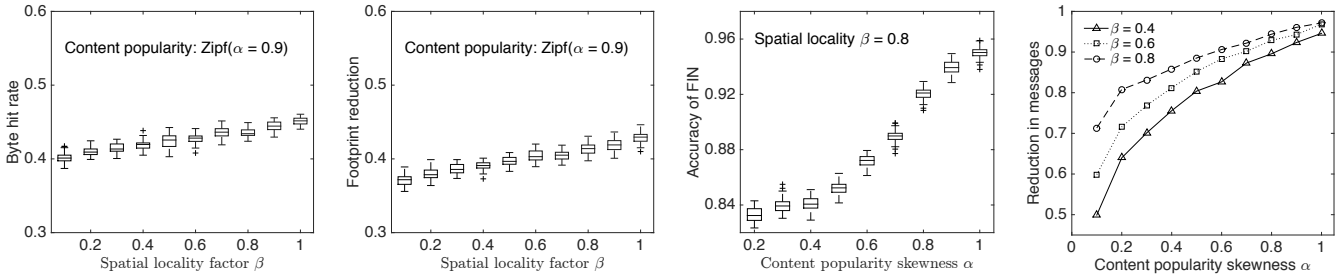
Fig. 4: Comparison of strategies on AT&T, 4 GB cache size.

Figure 4b shows the log-log plot of nodes' normalised utility as a function of betweenness centrality [7]. Nodes with a high betweeness are core routers, whilst those with low betweeness 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 analysis of spatial locality. We have confirmed that the results are representative of the other topologies.

We use a Hawkes process-based algorithm [22] to generate a user request trace. The algorithm is controlled by two



(a) Byte hit rate vs. spatial locality (b) FP reduction vs. spatial locality (c) Accuracy vs. popularity skew (d) Msg reduction vs. popularity skew

Fig. 5: Experiments on the Guifi network, 4 GB cache size. We vary both content popularity skewness α and spatial locality factor β 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.

parameters: a *content popularity skewness* α and a *spatial locality factor* β. α controls the overall content popularity which follows *Zipf*(α). 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 α and β in $(0, 1]$. From Figure 5a and 5b, we observe a shallow improvement on byte hit rate and footprint reduction as β increases. Specifically, they increase by only 6% and 8% respectively when increasing β from 0.1 to 1. This suggests that spatial locality is not a critical requirement for FairCache.

On the other hand, the popularity skew, α, 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 α from 0.2 to 1. The speed of degradation of accuracy by decreasing α 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 α). 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 α and β have a notable impact. Higher α and β both result in lower overheads (*i.e.* higher reductions). The reason is that a smaller α 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 β values have almost the same effect on demand matrices as that of α. However, we also notice that β has more significant impacts when α is small.

VII. Discussion and Limitations

Deployment of FairCache raises a number of questions. To ensure tractability, we have designed FairCache with a number of assumptions that must be discussed. So far, we have focussed on storage as the key bottleneck (*i.e.* whether or not an object is locally stored). However, there are also a number of other constraints that we have not considered yet. For instance, hardware bottlenecks can render servers useless even whilst in possession of content (*e.g.* CPU, I/O bus, congestion collapse). As FairCache is aimed at situations where caches are owned by *separate* organisations, it is likely that these concerns will be of potential interest. Thus, deployment will probably involve the integration 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 in most scenarios, although it is something future work should explore. Another point worth highlighting is that we base caching decisions on per-object popularity counts, therefore introducing greater memory overheads. We emphasise, however, that our heuristic removes all unpopular content, making such counts highly feasible (see our complexity analysis [1]).

Finally, a critical practical concern is the potential for stakeholders to try to maliciously undermine FairCache. This is because we require nodes to accurately report shadow prices. FairCache was 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 (*e.g.* RPKI [30]). Equally, we envisage FairCache could rely on similar principles, in which legally formed (potentially transitative) collaboration agreements are underpinned by public key cryptography. However, we delay such exploration to future work.

VIII. 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 [4]–[10], [23], even

though edge caching has also been shown to be effective [24]. Previously proposed solutions are either limited by a centralised solver [4], [25] which makes scalability difficult, or limited by distributed heuristics [5]–[10], [23], 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.

The current solution used for Internet-scale content delivery are CDNs. They hold many similarities to ICNs [24], 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 [31], 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.

Recent work [25]–[29] 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, these work take 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 [25], [26], [28], 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 though; we introduce this as an integral requirement of cooperation.

IX. Conclusion

To date, studies of collaborative ICN caching have focussed on traditional 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 a number of avenues of future work. The most prominent challenge is implementing a FairCache protocol that is robust against cheating. However, there are several other real-world concerns that we wish to integrate into FairCache, *e.g.* considering link dynamics and bandwidth constraints.

References

[1] L. Wang et al., "Fair Collaborative In-Network Caching Game and Its Cost Analysis on General Network Topologies", https://www.dropbox.com/s/woize3vkki1z5bt/report.pdf?dl=0

[2] V. Jacobson, et al., "Networking named content," in *CoNext, ACM*, 2009.

[3] T. Koponen, et al., "A data-oriented (and beyond) network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 181–192, 2007.

[4] M. D. Dahlin, et al., "Cooperative caching: Using remote client memory to improve file system performance," in *USENIX OSDI*, 1994.

[5] E. Rosensweig, et al., "Breadcrumbs: Efficient, best-effort content location in cache networks," in *INFOCOM, IEEE*, April 2009.

[6] J. Dai, et al., "Collaborative hierarchical caching with dynamic request routing for massive content distribution," in *INFOCOM, IEEE*, 2012.

[7] W. K. Chai, et al., "Cache "less for more" in information-centric networks," in *IFIP'12 Networking*, 2012.

[8] S. Saha, et al., "Cooperative caching through routing control in information-centric networks," in *INFOCOM, IEEE*, April 2013.

[9] I. Psaras, et al., "Probabilistic in-network caching for information-centric networks," in *ICN, ACM*, 2012.

[10] S. Borst, et al., "Distributed caching algorithms for content distribution networks," in *INFOCOM, IEEE*, pp. 1–9, 2010.

[11] J. Nash, et al., "The bargaining problem," *Econometrica*, vol. 18, 1950.

[12] E. Kalai, et al., "Other solutions to nash's bargaining problem," in *Econometrica*, vol. 43, 1975.

[13] F. P. Kelly, et al., "Rate control for communication networks: shadow prices, proportional fairness and stability," in *JORS*, 1998.

[14] H. Boche, et al., "Nash bargaining and proportional fairness for wireless systems," *Transactions on Networking, IEEE/ACM*, 2009.

[15] A. Muthoo, *Bargaining theory with appl.*, Cambridge Univ Press, 1999.

[16] S. Boyd, et al., *Convex optimization*, Cambridge Univ Press, 2004.

[17] L. Wang, et al., "LiteLab: Efficient Large-scale Network Experiments," in *CCNC, IEEE*, 2016.

[18] N. Spring, et al., "Measuring ISP topologies with rocketfuel," in *SIGCOMM, ACM*, 2002.

[19] D. Vega, et al., "Topology patterns of a community network: Guifi.net," in *WiMob, IEEE*, pp. 60-66, 2012.

[20] M. Cha, et al., "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system," in *IMC*, 2007.

[21] D. Bertsimas, et al., "The price of fairness," *Operations research*, 2011.

[22] L. Wang, et al., "Pro-Diluvian: Understanding Scoped-Flooding for Content Discovery in Information-Centric Networking," in *ICN, ACM*, 2015.

[23] L. Fan, et al., "Summary cache: A scalable wide-area web cache sharing protocol," *Transactions on Networking, IEEE/ACM*, 2000.

[24] S. K. Fayazbakhsh, et al., "Less pain, most of the gain: Incrementally deployable icn," in *SIGCOMM, ACM*, 2013.

[25] B.-G. Chun, et al., "Selfish caching in distributed systems: a game-theoretic analysis," in *PODC, ACM*, 2004.

[26] V. Pacifici, et al., "Selfish content replication on graphs," in *Proceedings of the 23rd International Teletraffic Congress*, 2011.

[27] V. Pacifici, et al., "Content-peering dynamics of autonomous caches in a content-centric network," in *INFOCOM, IEEE*, 2013.

[28] N. Laoutaris, et al., "Distributed selfish replication," *TPDS*, 2006.

[29] G. Dan, "Cache-to-cache: Could isps cooperate to decrease peer-to-peer content distribution costs?" *TPDS*, 2011.

[30] M. Wahlisch, et al. "RiPKI: The Tragic Story of RPKI Deployment in the Web Ecosystem", in *HotNets*, 2015.

[31] L. Peterson, et al. "Framework for Content Distribution Network Interconnection (CDNI)", *RFC 7336*