Fair Distribution of Digital Payments: Balancing Transaction Flows for Regulatory Compliance

Ashlesha Hota*
IIT Kharagpur
India

Shashwat Kumar*
IIT Kharagpur
India

Daman Deep Singh
IIT Delhi
India

Abolfazl Asudeh
University of Illinois Chicago
United States

Palash Dey
IIT Kharagpur
India

Abhijnan Chakraborty
IIT Kharagpur
India

Abstract

The Unified Payments Interface (UPI) is the world's largest real-time payment system, processing more than 21 billion transactions per month, and has emerged as India's primary mode of digital payments. However, the concentration of UPI transactions within just two apps - PhonePe and Google Pay - has raised concerns of duopoly in India's digital financial ecosystem. To address this, the National Payments Corporation of India (NPCI) has mandated that no single UPI app should exceed 30% of total transaction volume. Enforcing this cap, however, presents a significant computational and operational challenge: *how to redistribute user transactions across apps while minimizing user inconvenience and maintaining capacity limits?*

In this paper, we formalize this challenge as the Cap-Constrained UPI problem and model it as a Minimum Edge Activation Flow problem on a bipartite user-application network, where activating an edge represents an additional app installation. Our goal is to determine the smallest set of new installations that allows the system to route all transactions without violating app capacity constraints. We show that Minimum Edge Activation Flow is NP-Complete through a reduction from 3-Partition problem, motivating the need for scalable approximation strategies. To tackle this challenge, we propose Decoupled Two-Stage Allocation Strategy (DTAS), a scalable allocation framework with both offline and adaptive online variants. We further extend it with a fairness-aware mechanism that balances transaction load across applications while keeping additional installations low. Our experimental results on a dataset of 100 million transactions show that DTAS comes within 1–2 additional installations of the ILP optimum at every tested scale, while running several orders of magnitude faster. We further demonstrate a concrete trade-off curve between fairness and installation cost that can directly inform NPCI's forthcoming regulatory enforcement deadline. To our knowledge, this is the first work addressing the problem of balancing financial transaction flows for regulatory compliance and likely have significant real-world impact.

Keywords

UPI Transaction Caps, Flow Optimization, Fair Distribution

1 Introduction

Digital payment platforms have transformed how billions of people transact. In India, this shift is driven by the Unified Payments Interface (UPI), a government-backed system that enables instant bank-to-bank transfers across more than 77 apps such as PhonePe, Google Pay, and Paytm [16]. Unlike proprietary systems, UPI is shared public infrastructure with government-subsidized zero-fee transactions, and currently accounting for over 85% of India's digital payments [17].

In terms of absolute numbers, as of mid-2025, UPI served over 491 million users and 65 million merchants, processing over 640 million transactions daily, surpassing Visa's daily volume [4]. UPI now accounts for nearly 50% of all global real-time payment transactions, making it the world's most consequential retail payment rail [10]. Beyond India, UPI has been deployed in nine countries: Bhutan, Nepal, Singapore, Sri Lanka, Mauritius, UAE, France, Cyprus, and Qatar, and has been integrated into the PayPal World platform for international transactions [4].

Despite the open architecture, usage is highly concentrated. PhonePe and Google Pay together process over 80% of transactions [15]. This concentration creates systemic risk, since failures in a single app can disrupt a large fraction of payments, and leads to an uneven use of public subsidies, with most benefits going to a small number of platforms. To address this, India's payments regulator, the National Payments Corporation of India (NPCI), introduced a policy mandating that no single UPI app may process more than 30% of total transactions [11]. The goal is not to penalize dominant apps, but to ensure the ecosystem remains competitive, resilient, and equitable so that public infrastructure serves all participants, not just the most popular ones. However, enforcing this cap is far from straightforward as users naturally gravitate toward familiar apps. NPCI can not ask millions of users to change their habits overnight.

A naive enforcement mechanism is a tail-drop policy, blocking further transactions once an app exceeds its quota. Although effective, it leads to abrupt failures and poor user experience. A more practical solution is an alert-based strategy, where users are notified when an app nears its limit and encouraged to switch to another app. This idea parallels Random Early Detection (RED) in network congestion control [13], where early warnings prevent sharp throughput drops.

However, implementing such alert-based strategies is limited in practice, since users cannot be expected to install and maintain multiple UPI applications. As a result, routing decisions should consider user preferences and usage frequency rather than assuming constant switching across apps. In practice, achieving compliance with the cap requires that some users install additional applications in advance, enabling the system to redistribute load away from heavily used apps. Here, our notion of fairness is not based on equilibrium considerations, but rather on preventing excessive concentration

*Both authors contributed equally to this research.

of transactions on a small subset of applications, thereby ensuring balanced utilization of the shared infrastructure.

A natural approach is to incentivize such installations. Since the government already subsidizes UPI transactions to maintain zero fees [17], a portion of this budget could be repurposed to encourage users to adopt additional, underutilized applications – for example, via time-bound or usage-linked rewards. Given that such incentives are inherently limited, the central question we address is: *what is the minimum number of additional app installations required to ensure that all transactions can be routed within the regulatory cap, without any failures?*

The problem arises at two natural operational timescales in digital payment systems. In the offline setting, we assume access to historical transaction logs, which reveal user activity patterns such as frequent users and their transaction volumes. This allows the system to proactively recommend additional app installations for a carefully chosen subset of users, so that future transaction demand can be feasibly routed without disruption. Operationally, this corresponds to periodically running our augmentation algorithm on aggregated data and deploying targeted interventions (e.g., incentives) ahead of time. In contrast, the online setting captures the real-time nature of payment systems, where transactions arrive sequentially and future demand is unknown. Upon the arrival of each transaction, the system must immediately decide how to route it and, if necessary, whether to activate a new user-application connection without knowledge of future requests.

We formalize this as the Minimum Edge Activation Flow problem on a bipartite flow network $G = (V = \{s\} \cup U \cup A \cup \{t\}, E = E_{\text{solid}} \cup E_{\text{dashed}})$. Here, users U and apps A form two partitions: solid edges represent existing app installations, dashed edges denote potential ones, s–U edges have capacities t_u (user transaction volumes), and A–t edges have capacities c_a (app limits). The objective is to activate the minimum number of dashed edges to ensure a feasible integral flow from s to t without exceeding any app's capacity.

Despite its intuitive formulation, the problem is computationally intractable: we prove that determining the smallest feasible activation set or equivalently, the minimal number of additional app installations is NP-complete, even when restricted to only three apps. This rules out any polynomial-time exact solution unless P = NP. Consequently, we propose efficient greedy heuristics that approximate the optimal activation pattern while being scalable for large transaction networks.

Our Contributions. Our main contributions are as follows:

- ▷ We introduce the Cap-Constrained UPI, Online Cap-Constrained UPI, and Fair Cap-Constrained UPI problems for capacity-aware transaction routing, and show that the corresponding decision problem is NP-complete via a reduction from 3-Partition;
- ▷ We design two ILP formulations for the problem: an installation-minimization formulation and a fairness-aware extension for balancing transaction loads across applications;
- ▷ We propose Decoupled Two-Stage Allocation Strategy, a scalable offline layered allocation strategy that exploits workload skewness and application reuse patterns to minimize unnecessary app installations while remaining close to the ILP optimum;
- ▷ Thereafter we develop Online_DTAS, an adaptive online algorithm employing sketch-based heavy-user detection, delayed scheduling, and capacity reservation mechanisms, and further introduce Fair_DTAS, a fairness-aware extension based on composite allocation scoring; and
- ▷ We present an extensive experimental evaluations on synthetic and transaction-driven datasets against multiple baselines, demonstrating improvements in installation efficiency, fairness, and scalability.

2 Background and Related Work

The Unified Payments Interface (UPI), developed by the National Payments Corporation of India (NPCI) under the guidance of the Reserve Bank of India (RBI), has become a major part of India's digital economy. It provides an interoperable payment infrastructure that enables secure, real-time fund transfers and supports innovation through its open design [18]. As of 2024, UPI serves over 350 million users, connects more than 550 banks, and supports 77 payment apps including Google Pay, PhonePe, BHIM, and WhatsApp Pay [2]. In 2023, it processed 117 billion transactions worth USD 2.19 trillion [2]. However, rising transaction volumes have simultaneously increased the risk of fraud and anomaly. Several scholarly works, such as [6, 9, 12], have emphasized AI-driven approaches to strengthen fraud detection mechanisms in digital payment systems.

Beyond fraud detection, another critical concern in the financial domain is fairness in decision-making. Unfair decision-making in financial services occurs when certain groups or individuals face biased treatment in areas such as loan approvals, credit scoring, or mortgage access. Such bias can arise from historical discrimination or from machine learning models that inadvertently learn unfair patterns from data. Consequently, minority groups or equally qualified applicants may be denied fair financial opportunities, reinforcing existing social and economic inequalities. Song et al. [20] address this issue by proposing a Temporal Fair Graph Neural Network (TF-GNN) framework that models financial transactions as dynamic networks and enforces individual fairness over time. They introduce two new fairness notions specific to temporal graphs, provide a theoretical analysis of their fairness regret, and demonstrate through real-world experiments that their approach improves both prediction accuracy and fairness compared to prior methods.

There is another growing problem: UPI usage is highly concentrated. In September 2025, two third-party apps handled over 80% of all UPI transactions [15]. Such concentration creates systemic risks and limits competition. Similar concentration patterns have also been observed in other large-scale digital payment ecosystems. In China, Alipay and WeChat Pay have become the dominant digital payment platforms and largely displaced traditional card-based payments through strong network effects and ecosystem integration [21]. Studies and industry analyses report that the Chinese mobile payment market has evolved into a near-duopoly, raising concerns regarding market concentration and platform dependence [5]. These observations suggest that payment concentration is not unique to UPI but represents a broader challenge in digital payment systems.

To address this issue, NPCI introduced a 30% transaction cap for third-party applications. Existing work mainly focuses on policy and

market effects, with little attention to algorithmic approaches for enforcing such caps efficiently.

A natural perspective for our problem comes from network flow and augmentation problems. In particular, the Pure Fixed Charge Transportation problem [1, 3, 7, 22] is closely related and provides important technical foundations. However, our setting differs fundamentally because activating additional edges corresponds to installing new user–application connections. This introduces behavioral and infrastructural constraints that traditional augmentation models do not capture.

Finally, our motivation for early alerts draws inspiration from congestion-control mechanisms such as Random Early Detection (RED) [13], where systems proactively react before hard capacity limits are reached. Rather than dropping requests after exceeding thresholds, RED uses early interventions to prevent congestion collapse. We adopt a similar philosophy: proactively recommending app installations before capacities become binding.

To the best of our knowledge, this is the first work that formally studies and models this concentration issue, proposing mechanisms to enhance resilience and promote a more balanced and inclusive UPI ecosystem.

3 Problem Definition

We model the problem of fairly balancing UPI transactions as a bipartite flow network. Let U denote the set of users and A denote the set of UPI apps. Each user $u \in U$ generates t_u transactions, while each app $a \in A$ can process at most c_a transactions, typically defined as a fraction of the total transaction volume. Users may already have certain apps installed, represented by solid edges $E_{\text{solid}} \subseteq U \times A$, while potential additional installations are represented by dashed edges $E_{\text{dashed}} \subseteq U \times A$. A source node s is connected to all users with edges of capacity t_u, and a sink node t is connected to all apps with edges of capacity c_a. The goal is to identify the minimal subset of dashed edges $E' \subseteq E_{\text{dashed}}$ that need to be activated such that all transactions can be routed from s to t without exceeding app capacities or dropping transactions. Figure 1 illustrates our construction.

We now formally state our problems.

Definition 3.1 (Cap-Constrained UPI). Given a bipartite graph $G = (U, A, E_{\text{solid}} \cup E_{\text{dashed}})$, where an edge $(u, a) \in E_{\text{solid}}$ indicates that user u has application a installed, and $(u, a) \in E_{\text{dashed}}$ is a potential installation. Each user $u \in U$ has t_u transactions to route, and each application $a \in A$ has a capacity $c_a \in \mathbb{N}$. The goal is to find a minimum subset $E' \subseteq E_{\text{dashed}}$ such that all transactions can be feasibly routed via $E_{\text{solid}} \cup E'$, with the load on every application $a \in A$ not exceeding c_a.

Definition 3.2 (Online Cap-Constrained UPI). Given a bipartite graph $G = (U, A, E_{\text{solid}} \cup E_{\text{dashed}})$, where an edge $(u, a) \in E_{\text{solid}}$ indicates that user u has application a installed, and $(u, a) \in E_{\text{dashed}}$ is a potential installation. Each application $a \in A$ has a capacity $c_a \in \mathbb{N}$. Transactions arrive as an online sequence $\sigma = (\tau_1, \tau_2, \ldots, \tau_n)$, where each τ_i belongs to some user $u_i \in U$ and must be routed immediately and irrevocably. The goal is to find a minimum subset $E' \subseteq E_{\text{dashed}}$ such that all transactions can be feasibly routed via $E_{\text{solid}} \cup E'$, with the load on every application $a \in A$ not exceeding c_a.

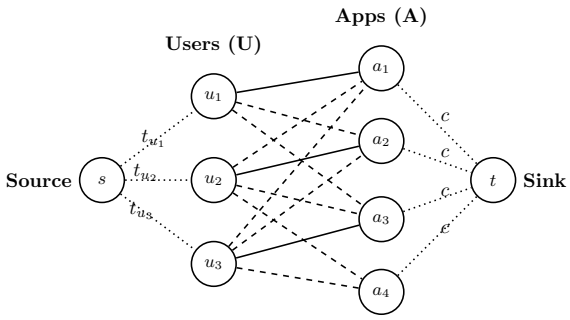


Figure 1: Illustration of the bipartite UPI transaction flow network. Solid edges represent pre-installed apps (E_{solid}), dashed edges denote potential installations (E_{dashed}), and dotted edges indicate user transactions (t_{u_i}) and app capacities (c).

Definition 3.3 (Fair Cap-Constrained UPI). Given a bipartite graph $G = (U, A, E_{\text{solid}} \cup E_{\text{dashed}})$, where an edge $(u, a) \in E_{\text{solid}}$ indicates that user u has application a installed, and $(u, a) \in E_{\text{dashed}}$ is a potential installation. Each user $u \in U$ has t_u transactions to route, and each application $a \in A$ has a capacity $c_a \in \mathbb{N}$. The goal is to find a minimum subset $E' \subseteq E_{\text{dashed}}$ such that all transactions can be feasibly routed via $E_{\text{solid}} \cup E'$, with the load on every application $a \in A$ not exceeding c_a, and the load vector $L = (\ell_1, \ldots, \ell_{|A|})$ is balanced across applications.

Let $L = (\ell_1, \ell_2, \ldots, \ell_{|A|})$ denote the vector of application loads. Fairness of a routing is evaluated using:

- ▷ **Max-Min Gap:**

$$\max_{a \in A} \ell_a - \min_{a \in A} \ell_a.$$

  Smaller gap indicates fairer load balancing.

- ▷ **Gini Coefficient:**

$$G(L) = \frac{\sum_{i=1}^{|A|} \sum_{j=1}^{|A|} |\ell_i - \ell_j|}{2|A| \sum_{i=1}^{|A|} \ell_i}.$$

  Lower Gini coefficient implies a more equal distribution of transactions across applications.

The Fair Cap-Constrained UPI problem asks for a feasible routing that minimizes the number of activated edges $|E'|$ while simultaneously maintaining balanced application loads according to the above fairness measures.

4 Complexity Status

We formulate the problem theoretically as the Minimum Edge Activation Flow (MEAF) problem.

Minimum Edge Activation Flow (MEAF)
Input. Given a bipartite graph $G = (U \cup A, E_{\text{solid}} \cup E_{\text{dashed}})$, a source node s connects to each $u \in U$ by an edge of capacity t_u, and a sink node t connects to each $a \in A$ by an edge of capacity c. Edges in E_{solid} represent existing connections, while E_{dashed} denote optional edges that can be activated.
Output: Find the smallest subset $E' \subseteq E_{\text{dashed}}$ such that all demands t_u can be routed from s to t through $E_{\text{solid}} \cup E'$ without violating any capacity c.

Formally, let $f(u,a)$ denote the number of transactions routed from user u to app a. The flow must satisfy the following conditions: every user's transactions are fully routed, i.e., $\sum_{a\in A} f(u,a) = t_u$, and the total flow into any app respects its capacity, $\sum_{u\in U} f(u,a) \leqslant c$. Flow is allowed only on solid edges or activated dashed edges, and transaction units are indivisible, so $f(u,a) \in \mathbb{Z}_{\geqslant 0}$. Finally, the total flow in the network equals the total number of transactions, ensuring no transaction is dropped. The objective is to minimize $|E'|$, the number of additional app installations required to achieve a feasible integral flow. We now formally show that Minimum Edge Activation Flow is NP-complete.

Theorem 4.1. Minimum Edge Activation Flow *is* NP*-complete.*

Proof: It is easy to see that the problem is in NP: given a set of activated dashed edges and an integral flow assignment, we can verify in polynomial time that (i) flow conservation holds, (ii) capacities on (s,u) and (a,t) are respected, (iii) flow is sent only on activated dashed edges, and (iv) the number of activated dashed edges is at most k.

We prove NP-hardness by a polynomial-time reduction from 3-Partition. Let an instance of 3-Partition be given by $S = \{s_1,\dots,s_{3m}\}$ and B with $\sum_i s_i = mB$ and $B/4 < s_i < B/2$ for all i. We construct an instance of UPI flow problem as follows.

Construction. Create $3m$ users $U = \{u_1,\dots,u_{3m}\}$ with $t_{u_i} = s_i$, $\forall i \in [3m]$. Create m apps $A = \{a_1,\dots,a_m\}$ with capacities $c_{a_j} = B$ for all j. Add source s and sink t with edges (s,u_i) of capacity t_{u_i} and edges (a_j,t) of capacity B. Let $E_{\text{solid}} = \emptyset$ and $E_{\text{dashed}} = U \times A$ (i.e., every user can connect to every app via a dashed edge). All (u,a) edges have sufficiently large capacity (e.g., capacity t_u) so that the app capacities are the binding constraints. Set the activation budget $k := 3m$.

This construction is clearly polynomial in the input size.

($\Rightarrow$) If the 3-Partition instance is a YES-instance, then the constructed flow instance admits a feasible integral flow using at most $k = 3m$ activated dashed edges. Assume s can be partitioned into m disjoint triples $T_1,\dots,T_m$ with $\sum_{s\in T_j} s = B$ for each j. For each triple T_j, choose an app a_j and for each user u_i with $s_i \in T_j$ activate exactly the dashed edge (u_i,a_j) and send the full amount $t_{u_i} = s_i$ on that edge. For each app a_j, the incoming flow is $\sum_{u_i\in T_j} t_{u_i} = B$, so the capacity (a_j,t) is respected. Each user uses exactly one dashed edge, so the number of activated dashed edges is exactly $3m = k$. All source capacities (s,u_i) are saturated, hence the total flow value is $\sum_i s_i = mB$. Thus there exists a feasible integral flow using at most k activations.

($\Leftarrow$) If the constructed flow instance admits a feasible integral flow using at most $k = 3m$ activated dashed edges, then the 3-Partition instance is a YES-instance. Suppose there is a feasible integral s–t flow of value $\sum_{i=1}^{3m} s_i = mB$ using at most $3m$ activated dashed edges. Since $E_{\text{solid}} = \varnothing$ and every user u_i has positive demand $t_{u_i} = s_i > 0$, each user must have at least one activated outgoing dashed edge in order to send any flow. Therefore any feasible solution uses at least $3m$ activated dashed edges. By the budget bound, the solution uses exactly $3m$ activations, hence each user activates *exactly one* dashed edge and sends all of its (integral) demand through that edge.

Let $S_j \subseteq U$ be the set of users assigned to app a_j (i.e., those for which (u,a_j) is activated and carries the full t_u). Flow feasibility and capacity imply for every j that

$$\sum_{u\in S_j} t_u \leqslant c_{a_j} = B.$$

Summing over all apps and using that the total flow equals mB gives

$$\sum_{j=1}^{m}\sum_{u\in S_j} t_u = \sum_{u\in U} t_u = mB = \sum_{j=1}^{m} B,$$

which forces equality in each app separately: $\sum_{u\in S_j} t_u = B$ for all j. Finally, by the 3-Partition bounds $B/4 < t_u < B/2$, no app can receive 1 item (any $t_u < B/2$) or $\geqslant 4$ items (each $t_u > B/4$ would exceed B). Therefore each S_j has exactly three users and their demands sum to B. The family $\{S_1,\dots,S_m\}$ thus yields a partition of s into m triples each summing to B, i.e., a YES-solution for 3-Partition. □

It is easy to see that Minimum Edge Activation Flow problem generalizes the Set Cover problem and hence inherits its hardness of approximation [8]. We now state the inapproximability result.

Remark 4.2. Unless P = NP, Minimum Edge Activation Flow admits no polynomial-time approximation algorithm with an appoximation factor better than $(1 - o(1))\log n$.

5 Proposed Methodology

As shown in the previous section, our problems are NP-complete. Therefore, we seek heuristics that perform well in practice. This section presents two ILP formulations for the transaction-routing problem. The first formulation minimizes additional app installations under application-capacity constraints, while the second extends the model with a max-min fairness objective over application loads.

We then introduce Decoupled Two-Stage Allocation Strategy, an offline greedy allocation framework that provides key empirical insights into the trade-off between installation efficiency and fairness. Building on these observations, we finally present a family of online algorithms, including our proposed adaptive framework Online_DTAS.

5.1 ILP-Based Routing Framework

We begin by presenting Integer Linear Programming (ILP) formulations for the transaction-routing problem. The formulations model the two central objectives in our setting: minimizing additional app installations and achieving fairness across applications.

Install-Minimization ILP: This problem can be formulated as an Integer Linear Programming (ILP) problem. We introduce variables $f(u,a) \in \mathbb{Z}_{\geqslant 0}$ for the number of transactions routed from user u to app a, and binary variables $x(u,a)$ for $(u,a) \in E_{\text{dashed}}$ indicating whether a potential edge is activated.

The objective is to minimize the number of additional edges activated:

$$\min \sum_{(u,a)\in E_{\text{dashed}}} x(u,a). \tag{1}$$

Each user's transactions must be fully routed, captured by

$$\sum_{a\in A} f(u,a) = t_u, \quad \forall u \in U, \tag{2}$$

while the capacities of apps must not be exceeded:

$$\sum_{u \in U} f(u, a) \leqslant c_a, \quad \forall a \in A. \tag{3}$$

Flow through a dashed edge is permitted only if the edge is activated:

$$f(u, a) \leqslant t_u \cdot x(u, a), \quad \forall (u, a) \in E_{\text{dashed}}. \tag{4}$$

Flow on solid edges is unconstrained by activation, i.e., $f(u, a) \geqslant 0$ for $(u, a) \in E_{\text{solid}}$. All flows are integral:

$$f(u, a) \in \mathbb{Z}_{\geqslant 0}, \quad x(u, a) \in \{0, 1\}. \tag{5}$$

Max-Min Fairness ILP: For the fair variant, we keep the same routing variables and add y_a for additional installations on app a and T for the minimum installation level.

The objective becomes:

$$\max T. \tag{6}$$

The extra constraints are:

$$y_a = \sum_{u \in U} x(u, a), \quad \forall a \in A, \tag{7}$$

$$y_a \geqslant T, \quad \forall a \in A, \tag{8}$$

$$\sum_{a \in A} y_a \leqslant B. \tag{9}$$

Here, B is not a free parameter: it is set to the total number of additional app installations returned by the Install-Minimization ILP. Therefore, the second ILP does not increase the installation volume; instead, it redistributes the same installation budget across apps to maximize the minimum installation level T.

Together, these formulations capture both efficiency and fairness in the routing model. The first ILP determines how many new installations are minimally necessary to serve all demand, and the second uses exactly that amount as a cap and decides how to distribute installations so that the least-installed app is as high as possible.

5.2 Offline Strategy

Although the ILP formulations provide optimal allocations, they become computationally expensive at large scales. To improve scalability, we develop an offline greedy heuristic called Decoupled Two-Stage Allocation Strategy (Algorithm 1).

In the offline setting, all transaction requests are available in advance, allowing users to be reordered prior to allocation. As shown in Algorithm 1, transactions for each user are allocated using a three-layer strategy:

(1) preinstalled applications,
(2) previously activated extra applications,
(3) new app installations only when necessary.

By prioritizing reuse before introducing new installations, Decoupled Two-Stage Allocation Strategy reduces redundant app activations while improving capacity utilization.

A natural strategy is to process users in descending order of their transaction-to-app ratio, prioritizing heavy users first. Intuitively, allocating demanding users early appears beneficial because they have larger transaction loads and potentially fewer feasible allocation choices. However, empirical analysis on real transaction traces revealed a highly skewed workload distribution, where a small fraction of users generated the majority of transactions (see Figure 2). Prioritizing these heavy users rapidly saturated frequently installed high-capacity applications, leaving lightweight users with fewer reusable options and forcing additional application installations despite sufficient aggregate capacity being available.

To address this issue, Algorithm 1 instead processes users in ascending order of their transaction-to-app ratio, prioritizing lightweight users first. Since lightweight users constitute the majority, serving them early preserves allocation flexibility and significantly reduces unnecessary installations while maintaining scalability.

Insights from DTAS**.** The behavior of DTAS revealed that the timing of heavy-user allocations has a substantial impact on system efficiency. Immediately serving heavy users reduces future reuse opportunities by consuming a large fraction of the available application capacity early in the allocation process. In contrast, postponing heavy-user requests allows lightweight users to exploit existing installations first, preserving flexibility and reducing redundant application activations.

This observation directly motivates the delay mechanism in Online_DTAS, where dynamically identified heavy users are temporarily stalled and scheduled later to improve long-term allocation efficiency.

5.3 Online Strategies

In practice, transaction-routing decisions arise in a dynamic environment where requests arrive sequentially and decisions must be made in real time. At each step, the system faces a choice: continue routing through currently available applications or activate additional applications when necessary. Since future requests are unknown, decisions must be made online using only the information available at the current time.

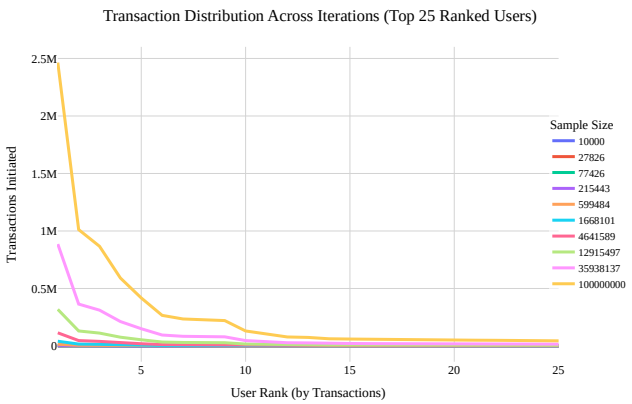


Figure 2: Distribution of user transaction volumes showing a highly skewed workload, where a small fraction of users contributes a disproportionately large number of transactions.

Motivated by this setting, we propose two online strategies: Online_DTAS and Fair_DTAS.

▷ **Online_DTAS**. To handle streaming transaction arrivals, we further extend DTAS into an online framework (Algorithm 2). Unlike the offline setting, complete user information is not available in advance, making it impossible to identify heavy

Algorithm 1: DTAS: Decoupled Two-Stage Allocation Strategy

Input : Users U, transaction demand T_u for each user u, preinstalled applications A_u, application capacity c
Output : Updated application assignments and transaction allocation

Initialize remaining capacity $cap[a] \leftarrow c$ for all applications a;
Initialize shared application pool:

$$P \leftarrow \bigcup_{u \in U} A_u$$

Sort users by increasing demand and number of installed apps;
Phase 1: Allocate using preinstalled applications
foreach $u \in U$ **do**
    $remaining \leftarrow |T_u|$;
    Sort A_u by decreasing $cap[a]$;
    foreach $a \in A_u$ **do**
        allocate as many transactions as possible to a;
        update $cap[a]$;
        if $remaining = 0$ **then**
            break
    store remaining demand of user u;
Phase 2: Satisfy unmet demand
foreach $u \in U$ *with remaining demand* **do**
    `// Step 2a: borrow from existing shared apps`
    candidateApps $\leftarrow P \setminus A_u$;
    Sort candidateApps by decreasing $cap[a]$;
    allocate transactions while capacity exists;
    `// Step 2b: install fresh applications if needed`
    if *demand still remains* **then**
        freshApps $\leftarrow$ apps not in A_u and not in P;
        Sort freshApps by decreasing $cap[a]$;
        allocate transactions;
        add newly installed apps to P;
Return allocation and updated application assignments;

Algorithm 2: Online_DTAS: Online DTAS with Heavy-User Delaying

Input : Transaction stream S, preinstalled apps A_u, application capacity c
Output : Updated application assignments and transaction allocation

Initialize remaining capacities $cap[a] \leftarrow c$;
Reserve a fraction αc of each application's capacity;
Initialize shared application pool:

$$P \leftarrow \bigcup_{u \in U} A_u$$

Initialize Space-Saving sketch with at most k counters;
Initialize heavy-user set $H \leftarrow \emptyset$;
Initialize bounded delay queue Q;
Initialize light transaction counter;
foreach *transaction* $(u, t) \in S$ **do**
    Update frequency estimate of user u in the Space-Saving sketch;
    if *reclassification interval reached* **then**
        Age sketch counters using decay factor ρ;
        Identify heavy users as top-percentile sketch entries;
        Update heavy-user set H;
    if $u \in H$ **then**
        Insert transaction into delay queue Q;
        if $|Q|$ *exceeds threshold* **then**
            Drain least-heavy delayed request;
    else
        Allocate transaction immediately using Algorithm 3;
        Increment light transaction counter;
        if *drain interval reached* **then**
            Determine drain count based on queue size;
            Drain delayed transactions from Q;
Drain all remaining delayed transactions;
Return allocation;

and light users beforehand. Therefore, Online_DTAS employs an adaptive frequency-tracking mechanism based on the *Space-Saving* algorithm [14] to dynamically identify heavy users from streaming transaction data.

As illustrated in Algorithm 2, the scheduler processes a unified transaction stream and continuously updates user frequencies through a compact Space-Saving sketch. Rather than maintaining explicit counters for every user, the sketch stores only a bounded number of entries and incrementally tracks approximate transaction frequencies using limited memory. For each incoming transaction, the corresponding user frequency estimate is updated within the sketch. Periodically, sketch counters are aged using multiplicative decay and users are reclassified according to the highest estimated frequency values. Specifically, users whose sketch scores fall within the top percentile are identified as heavy users, while the remaining users are treated as light users. This mechanism enables the algorithm to adapt to evolving workload patterns while preventing stale historical activity from dominating future classifications.

Heavy-user transactions are temporarily delayed and inserted into a bounded priority queue ordered by estimated transaction frequency, ensuring that the *least-heavy* delayed users are released first. In contrast, light-user requests are immediately allocated using the three-tier application selection strategy shown in Algorithm 3, inherited from offline DTAS:

▷ **Fair_DTAS**: Fair_DTAS extends the base online DTAS framework by incorporating an explicit fairness objective into the application selection process (Algorithm 4). The overall scheduling pipeline—including the streaming transaction model, heavy-user classification, delayed queue mechanism, adaptive draining policy, and capacity reservation strategy—remains identical to the base online framework described in Algorithm 2. The primary modification lies in replacing the application selection strategy with a fairness-aware allocation rule.

As shown in Algorithm 4, the key idea behind Fair_DTAS is to balance transaction load across applications while minimizing unnecessary installations. Instead of selecting applications solely based on availability and reuse, the algorithm evaluates

Algorithm 3: Application Selection Procedure

Input : User u, installed apps A_u, shared pool P, remaining capacities
Output : Selected application

Tier 1: Search installed applications;
if *feasible app exists* **then**
  Return highest affinity, highest capacity app;
Tier 2: Search shared-pool applications;
if *feasible app exists* **then**
  Return highest affinity, highest capacity app;
Tier 3: Search fresh applications;
if *feasible app exists* **then**
  Return highest affinity, highest capacity app;
Return NULL;

feasible applications using a fairness-aware composite scoring function:

$$Score(a) = \alpha \cdot InstallPenalty(a) + \beta \cdot \Delta Fair(a), \qquad (10)$$

where *InstallPenalty(a)* is zero for already-installed applications and positive otherwise, scaled by user–app affinity to encourage reuse. The fairness term $\Delta Fair(a)$ estimates the incremental increase in global load imbalance if the incoming transaction is assigned to application a. Intuitively, applications that are already heavily loaded incur larger fairness penalties, whereas lightly utilized applications receive lower penalties. In our implementation, this quantity is computed as the increase in load variance across applications after hypothetically assigning the transaction to app a. Consequently, assignments that disproportionately increase concentration on popular applications become less attractive.

The parameters α and β determine the trade-off between minimizing installation overhead and improving fairness. Larger values of α favor application reuse and lower installation cost, whereas larger values of β emphasize balanced load distribution.

The selected application is the one with minimum score; ties are broken using *(i) higher user–app affinity* and *(ii) larger remaining capacity*. By penalizing allocations that increase load imbalance, Algorithm 4 avoids excessive concentration on a small set of applications and promotes more balanced utilization.

By explicitly incorporating fairness into allocation decisions, Fair_DTAS achieves improved load balancing across applications while preserving the low-installation characteristics of the original DTAS framework, albeit with a modest increase in installation overhead.

Algorithm 4: Fair_DTAS: Fairness-Aware Online DTAS

Input : Transaction stream S, preinstalled apps A_u, application capacity c, fairness weights α, β
Output : Updated application assignments and transaction allocation

Execute the online scheduling procedure from Algorithm 2;
Replace the application-selection strategy with the following fairness-aware procedure:;
foreach *incoming transaction of user* u **do**
  Determine feasible applications:

$$F \leftarrow \{a \mid cap[a] > 0\}$$

  foreach $a \in F$ **do**
    Compute installation cost:

$$InstallCost(a) = \begin{cases} 0, & a \in A_u \\ \frac{1}{1+Affinity(u,a)}, & \text{otherwise} \end{cases}$$

    Estimate fairness impact:

$$\Delta Fairness(a)$$

    Compute combined score:

$$Score(a) = \alpha \cdot InstallCost(a) + \beta \cdot \Delta Fairness(a)$$

  Select application with minimum score;
  Break ties using:
  (1) higher user–app affinity
  (2) larger remaining capacity
  Assign transaction and update app load;
Return allocation;

6 Experimental Evaluations

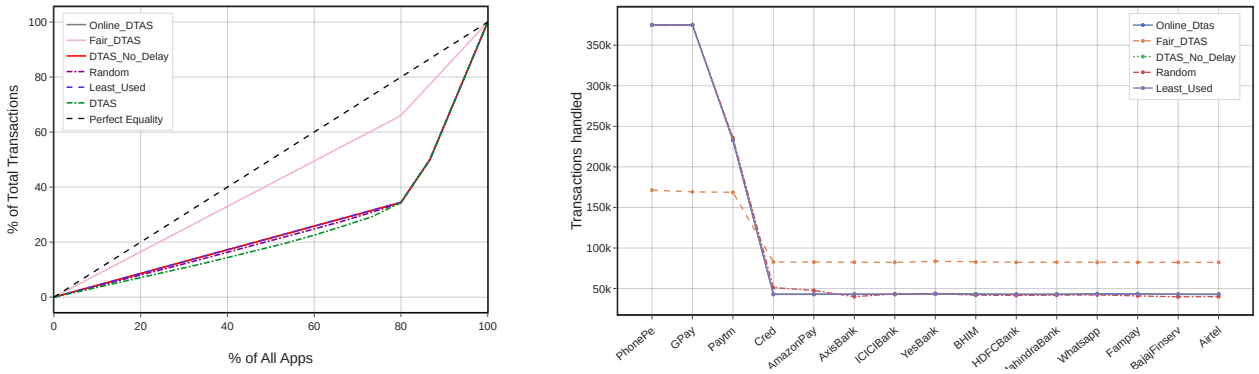
In this section, we present the experimental analysis on semi- synthetic data generated from Rabobank Transactinos data.

Computational Environment. We conduct experiments on a Linux-based system with an Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60 GHz and 125 GB RAM running Ubuntu 22.04 LTS. All algorithms are implemented in Python 3.12.3, and the Gurobi Optimizer (version 12.0.3) is used for solving the ILP formulations. Each experimental configuration represents a distinct combination of users, transaction workloads, and UPI applications under varying capacity constraints.

Dataset Description. Since real-world UPI transaction data is proprietary and not publicly accessible due to privacy and regulatory constraints, we use the Rabobank Transaction Dataset [19] as the underlying transaction corpus. The dataset contains anonymized records of banking activity within Rabobank collected over an 11-year period (2010–2020), covering 1,624,030 bank accounts and 4,127,043 transaction relationships. The original dataset includes customer-to-customer (C2C), customer-to-business (C2B), and business-to-business (B2B) interactions.

To approximate a UPI ecosystem, we assign each user a set of preinstalled payment applications according to NPCI-reported monthly transaction shares. Consequently, users are more likely to adopt high-volume applications such as PhonePe and GPay while maintaining representation of lower-volume applications. This augmentation captures realistic application adoption patterns while preserving diversity in user preferences.

Using this enriched dataset, we generate experimental instances across varying scales ranging from 10,000 to 100 million transactions and up to 1.2 million users under different transaction-to-user ratios and uniform application-capacity settings.



(a) Lorenz curve for various algorithms. **(b) Load Distribution across apps.**

Figure 3: Lorenz curve comparison across allocation approaches.

6.1 Baselines

To benchmark the proposed DTAS, Online_DTAS, and Fair_DTAS algorithms, we compare them against the following baseline strategies:

- ▷ **DTAS_No_Delay**. An ablation of Online-DTAS where the stalling mechanism is removed. Transactions are allocated immediately using the same three-tier strategy, i.e. *preinstalled applications, shared-pool applications,* and *new applications* with affinity-based tie-breaking. This baseline isolates the contribution of delayed scheduling and adaptive heavy-user handling.
- ▷ **Random**. A simple allocation strategy where each transaction is assigned uniformly at random among feasible applications. Preinstalled applications are preferred whenever available; otherwise, any application with remaining capacity is selected randomly. This serves as a lower-bound baseline with no optimization or workload awareness.
- ▷ **Least Used**. A greedy load-balancing baseline that assigns each transaction to the feasible application with the smallest current load (number of handled transactions), while following the same tiered allocation structure. This baseline explicitly prioritizes load balancing without considering installation overhead.

6.2 Evaluation Metrics

We evaluate all algorithms using the following metrics:

- ▷ **Per-App Load Distribution.** Let ℓ_a denote the number of transactions routed to application a. Since the primary fairness objective is to avoid concentrating transactions on a small subset of applications, we analyze the resulting load distribution across apps. More balanced distributions indicate better utilization of system capacity.
- ▷ **Extra App Installations.** We measure the total number of activated edges $|E'|$, corresponding to additional application installations beyond users' preinstalled apps. Lower values indicate greater installation efficiency and reduced user friction.
- ▷ **Fairness Comparison (Jain's Index).** To quantify load balancing across applications, we compute Jain's fairness index over application loads:

$$J(L) = \frac{(\sum_{a \in A} \ell_a)^2}{|A| \sum_{a \in A} \ell_a^2}.$$

Jain's index ranges from 0 to 1, where values closer to 1 indicate a more uniform distribution of transaction loads across applications.

6.3 Quantitative Results

In this section, we analyze the proposed allocation strategies from the perspectives of fairness, installation overhead, and load distribution across applications. We examine trade-offs between balancing application utilization and minimizing additional installations, while studying how algorithmic choices influence these objectives.

Figures 3a and 3b present Lorenz curves and per-application load distributions across allocation strategies. The Lorenz curves provide a visual interpretation of fairness, where curves closer to the equality line indicate more balanced transaction distributions.

As expected, Fair_DTAS remains closest to the equality line, indicating a more uniform allocation than other methods. In contrast, Online_DTAS, DTAS _No_Delay, and baseline approaches exhibit greater concentration, consistent with lower fairness.

Table 1a reports experiments on a workload of 1.5 million transactions and highlights a clear trade-off between fairness and installation overhead. Fair_DTAS achieves the strongest fairness among all methods, but this comes with increased installation cost. Conversely, Online_DTAS prioritizes minimizing additional installations, resulting in lower fairness but reduced overhead.

The lower fairness observed in Online_DTAS is partly driven by skew in application availability across users. Since PhonePe and GPay are preinstalled for many users, and Online_DTAS prioritizes already-installed applications before recommending new installations, these applications naturally receive a larger transaction share

Algorithm	Extra Inst.	Jain's Index	Time (s)
Online_DTAS	152,344	0.4190	17.04
Fair_DTAS	183,907	0.8916	130.30
DTAS _No_Delay	155,671	0.4191	8.71
Random	362,267	0.4178	7.16
Least_Used	373,117	0.4194	7.71

(a) Performance comparison of allocation strategies

App Cap.	Users w/ Remaining Demand (%)	Unallocated Transactions
10	2.25%	65,982,913
15	1.00%	50,982,913
20	0.54%	40,418,301
25	0.29%	30,418,301
30	0.13%	20,418,301
35	0.02%	10,418,301
40	$\approx$ 0%	1,525,685

(b) Impact of app capacity on remaining demand

Table 1: Experimental results across allocation performance and capacity sensitivity analyses.

and higher capacity utilization. Thus, part of the observed concentration arises from user-installation distributions rather than purely algorithmic decisions.

The effect of delaying heavy users can be observed by comparing Online_DTAS and DTAS _No_Delay. While both methods exhibit nearly identical fairness behavior, removing the delay mechanism increases installation overhead without meaningful fairness gains. This suggests that selectively postponing heavy users improves allocation efficiency by reducing unnecessary installations.

Baseline methods such as Random and Least_Used incur higher installation overhead while exhibiting fairness characteristics similar to Online_DTAS. In particular, Online_DTAS consistently outperforms Least_Used, indicating that allocation decisions cannot rely solely on selecting applications with maximum available capacity. Instead, applications already installed by a user play a critical role in efficient allocations. This highlights the importance of reuse-aware routing, where prioritizing existing installations significantly reduces overhead without sacrificing allocation quality.

Overall, the results demonstrate a trade-off between fairness and installation efficiency while highlighting adaptive scheduling mechanisms.

For the experiments in Table 1b and Table 2, we consider the complete dataset consisting of 1.21 million users and 100 million transactions to evaluate varying application capacity limits.

Table 2 compares additional application installations across transaction scales. DTAS closely matches ILP performance while remaining scalable. Online variants incur higher installation overhead due to lack of future information. Fair_DTAS improves fairness at increased cost, while increased installations in DTAS _No_Delay highlight the benefit of delaying heavy-user requests.

As shown in Table 1b, increasing app capacity sharply reduces users requiring new installations from approximately 2.25% at a 10% capacity limit to nearly zero at 40%. Most users can therefore be fully served under moderate settings, while less than 2.3% experience unmet demand.

However, increased capacity also accentuates duopoly tendencies, where a few dominant applications absorb a disproportionate share of total transaction load. Such concentration may negatively affect competition and resilience. Conversely, lowering app capacity reduces dominance but results in over 65 million unallocated transactions at a 10% limit, indicating potential service degradation under strict caps. If regulatory constraints such as NPCI thresholds were enforced, remaining transactions would directly represent unserved demand, potentially causing user friction and reduced system efficiency.

6.4 Practicalities and Discussion

Given increasing concerns regarding market concentration and the emerging UPI duopoly, transaction-capacity regulations may eventually be enforced by NPCI. Our work provides an initial framework toward implementing such regulations while minimizing disruption to user behavior and market dynamics. Beyond demonstrating feasibility, our results provide a quantified path toward compliance. Across transaction scales, our ILP and DTAS formulations identify the minimum number of incremental user–application connections required to satisfy capacity constraints. At 100 million transactions, DTAS achieves compliance using only 6,599 additional installations while remaining computationally tractable, providing policymakers with practical estimates for sizing incentive budgets such as cashback programs or merchant rewards.

Our sensitivity analysis shows that selecting an appropriate capacity threshold is critical. At a 10% per-app cap, more than 65 million transactions remain unallocatable and approximately 2.25% of users require additional installations, whereas both quantities become nearly negligible at 40%. The proposed 30% threshold lies in a steep feasibility region where small relaxations substantially reduce unmet demand without significantly altering market structure. This provides NPCI with a principled basis for calibrating thresholds rather than treating a specific percentage as arbitrary.

A practical deployment strategy could begin with a trial phase in which users are not forced to switch applications but instead receive recommendations or incentives encouraging diversification of installed applications. Data collected during this phase can provide insights into user transaction frequencies and preferences required by our framework. Our findings also suggest that intervention strategies should be carefully designed. A counterintuitive but robust observation is that redirecting lightweight users before heavy users substantially reduces installation overhead. Heavy users rapidly consume available capacity, often forcing lightweight users to install new applications despite remaining system-wide capacity. Redirecting lightweight users therefore preserves flexibility and lowers overall system cost.

Interestingly, increasing app capacity improves allocation feasibility while simultaneously worsening concentration. With larger capacity limits, dominant applications absorb a disproportionately larger share of total transaction load, potentially strengthening the same duopoly regulations seek to address. Thus, capacity limits should not merely serve as hard ceilings but as routing targets directing traffic toward underutilized applications.

For regulators prioritizing ecosystem health, Fair_DTAS provides a useful blueprint. Although it incurs approximately 21% more

Transactions	ILP	LP	DTAS	Online_DTAS	Fair_DTAS	DTAS _No_Delay
10,000	5	3.78	5	26	104	37
63,095	25	24.22	25	178	698	533
398,107	90	88.99	93	690	3,979	2,782
1,000,000	198	196.88	198	2,031	7,771	7,799
6,309,573	1,314	1311.82	1,314	40,190	59,145	53,005
39,810,717	7,676	7668.81	7,676	320,510	406,657	353,934
100,000,000	***	***	6,599	738,726	904,179	797,000

Table 2: Comparison of Installations Across Different Algorithms

Note: LP values correspond to lower bounds obtained from the linear relaxation of the optimization problem. '***' indicates that ILP and LP could not be solved within practical runtime limits. **DTAS** assumes complete future knowledge and serves as an offline baseline. Online_**DTAS** performs transaction allocation without future information. Fair_**DTAS** incorporates fairness objectives during allocation. **DTAS** _No_Delay removes the stalling mechanism and allocates transactions immediately.

installations, it achieves substantially improved fairness and demonstrates that modest deployment costs can produce a more balanced application ecosystem.

Although access to real UPI transaction data remains restricted, our evaluation using a realistic financial transaction dataset suggests that the proposed framework scales effectively and can serve as a practical decision-support mechanism. The framework is also relevant beyond India, particularly as UPI expands internationally and interoperates with payment ecosystems experiencing similar concentration dynamics.

7 Conclusion

The current UPI ecosystem is heavily dominated by a small number of applications, raising concerns regarding market concentration and motivating interventions such as NPCI's proposed 30% market-share cap. Enforcing such constraints at scale is computationally challenging, particularly when user installation preferences and application capacities must be respected. In this work, we take a first step toward addressing this problem and propose DTAS, a scalable allocation framework that preserves users' existing application choices while minimizing additional installations required for compliance.

Our empirical evaluation reveals several important insights. First, reuse-aware allocation substantially reduces installation overhead compared to naive load-spreading strategies. Second, application availability and existing user installations play a critical role in allocation quality, highlighting the importance of incorporating user-level constraints into routing decisions. Third, heavy-user behavior strongly influences allocation efficiency: prioritizing or immediately serving heavy users rapidly saturates popular applications and reduces future reuse opportunities. This observation motivates the delay mechanism in Online_DTAS, where postponing dynamically identified heavy users improves long-term allocation efficiency without significantly affecting fairness. Finally, fairness and installation efficiency exhibit an inherent trade-off, where stronger fairness objectives improve load balancing at the cost of additional installations.

Overall, DTAS and its online variants achieve performance close to optimization-based approaches while remaining computationally efficient at large scales, even when optimization becomes impractical. These findings suggest that scalable reuse-aware allocation strategies can serve as practical decision-support tools for regulators seeking to improve ecosystem balance while preserving user convenience.

Our current formulation focuses on balancing transaction counts across applications. An important future direction is incorporating transaction values into the allocation process. High-value transactions may introduce different capacity requirements, user preferences, and financial risks, potentially revealing richer trade-offs between fairness, efficiency, and ecosystem resilience.

Acknowledgements

We thank Sreemanti Dey for her valuable help with an earlier version of this work. We are also grateful to Akrati Saxena for sharing the Rabobank transaction dataset, which significantly facilitated our empirical evaluation.

References

[1] Veena Adlakha and Krzysztof Kowalski. 1999. On the fixed-charge transportation problem. *Omega* 27, 3 (1999), 381–388.
[2] European Payments Council AISBL. 2024. UPI: revolutionising real-time digital payments in India. https://www.europeanpaymentscouncil.eu/news-insights/insight/upi-revolutionising-real-time-digital-payments-india. Accessed: 2024-06-26.
[3] John W Billheimer and Paul Gray. 1973. Network design with fixed and variable cost elements. *Transportation Science* 7, 1 (1973), 49–74.
[4] CNBC. 2025. UPI's global push: Exporting indigenous tech, and furthering economic strategy. https://www.cnbc.com/2025/10/16/cnbcs-inside-india-newsletter-upis-global-push-seems-an-economic-strategy.html
[5] Wikipedia contributors. 2025. Mobile payments in China. Market characterized by a WeChat Pay–Alipay duopoly.
[6] Deepa Devassy, Divya K S, Gripsy Paul Mannickathan, Alen Biju, Aswathi T, and Devarsh R. 2025. FedUPI: Federated Learning Empowered Detection of Fraudulent UPI Transactions. In *2025 5th International Conference on Soft Computing for Security Applications (ICSCSA)*. 1007–1015. doi:10.1109/ICSCSA66339.2025.11170880
[7] Teresa Estañ, Natividad Llorca, Ricardo Martínez, and Joaquín Sánchez-Soriano. 2021. On how to allocate the fixed cost of transport systems. *Annals of Operations Research* 301, 1 (2021), 81–105.
[8] Uriel Feige. 1998. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)* 45, 4 (1998), 634–652.
[9] Muhammad Liman Gambo, Anazida Zainal, and Mohamad Nizam Kassim. 2022. A convolutional neural network model for credit card fraud detection. In *2022 International Conference on Data Science and Its Applications (ICoDSA)*. IEEE, 198–202.
[10] International Monetary Fund. 2025. Growing Retail Digital Payments: The Value of Interoperability. https://www.newsonair.gov.in/india-making-fastest-payments-globally-due-to-massive-upi-adoption-imf/
[11] Jaspreet Kalra. 2024. India delays UPI market share cap in relief for Walmart backed PhonePe, Google Pay. https://www.reuters.com/technology/india-delays-upi-payments-market-share-cap-relief-walmart-backed-phonepe-google-2024-12-31/.
[12] N Lingareddy, Devadutta Indoria, S Deepika, Geogen George, M Keerthi Priya, et al. 2025. Enhancing Digital Payment Security: UPI Fraud Detection with Advanced Machine Learning Algorithms. In *2025 Global Conference in Emerging Technology (GINOTECH)*. IEEE, 1–7.

[13] Amar A Mahawish and Hassan J Hassan. 2022. Improving RED algorithm congestion control by using the Markov decision process. *Scientific Reports* 12, 1 (2022).

[14] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient Computation of Frequent and Top-k Elements in Data Streams. In *Proceedings of the International Conference on Database Theory (ICDT)*. Springer, 398–412.

[15] Meghna Mittal. 2025. 80% of UPI volumes concentrated with two app providers, fintech body writes to FinMin. https://shorturl.at/LFhTN.

[16] PIB. 2025. India's UPI Revolution. https://www.pib.gov.in/PressNoteDetails.aspx?NoteId=154912&ModuleId=3.

[17] Government of India Press Information Bureau. 2025. Advancing Cashless India. https://www.pib.gov.in/PressReleasePage.aspx?PRID=2114335&reg=3&lang=2 Press Release.

[18] Shailesh Rastogi, Chetan Panse, Arpita Sharma, and Venkata Mrudula Bhimavarapu. 2021. Unified Payment Interface (UPI): A digital innovation and its impact on financial inclusion and economic development. *Universal Journal of Accounting and Finance* 9, 3 (2021), 518–530.

[19] Akrati Saxena. 2023. RaboBank Transaction Dataset. https://github.com/akratiiet/RaboBank_Dataset. Accessed: 2025-04-12.

[20] Zixing Song, Yuji Zhang, and Irwin King. 2023. Towards fair financial services for all: A temporal GNN approach for individual fairness on transaction networks. In *Proceedings of the 32nd ACM international conference on information and knowledge management*. 2331–2341.

[21] Consultative Group to Assist the Poor (CGAP). 2019. China: A Digital Payments Revolution. https://www.cgap.org/research/publication/china-digital-payments-revolution.

[22] Pengfei Zhu, Guangting Chen, Yong Chen, and An Zhang. 2025. On the pure fixed charge transportation problem. *Discrete Optimization* 55 (2025), 100875. doi:10.1016/j.disopt.2024.100875