Async: De-congestion and Yield Management in Cellular Data Networks

Vijay Gabale, Uma Devi, Vinay Kolar, Mukundan Madhavan, Ravi Kokku, Shivkumar Kalyanaraman
IBM Research, India

Abstract—We describe the design and implementation of a novel system called Async, which enables a mobile network operator (MNO) to efficiently manage the growth of mobile data by leveraging the delay-elastic nature of certain applications and the price-sensitive nature of certain users. Specifically, Async introduces an alternate “asynchronous” content delivery paradigm for heavy content such as videos, and facilitates an MNO to negotiate a delay in content delivery with users in exchange for appropriate incentives. Thus, Async allows an MNO to control when, and at what price the content is delivered to the users. The MNO uses the negotiated delays to actively manage Async flows to reduce congestion during peak hours and to improve the quality-of-experience (QoE) of both delayed and regular flows; thus increasing the spectrum yield. We demonstrate the capability of such network-based flow management in comparison to state-of-the-art TUBE [1] by showing that Async enhances QoE for more than 30% of the regular flows, with up to 60% improvement in per-flow QoE metric, while still meeting the negotiated delivery times of 95% of the delayed flows. We also show that Async lowers the delivery times of delayed flows by up to 67% and increases the robustness to traffic unpredictability significantly. We implement a prototype of Async by extending Apache’s mod_proxy and developing an Android application. Our design is robust to disconnections and does not require any modifications to existing network infrastructure and protocols. Our prototype deployment on live networks confirms the efficacy of Async in meeting EDTs for diverse deployment scenarios.

I. INTRODUCTION

Predictions show that mobile data traffic will grow at a compound annual growth rate of 66% from 2012 to 2017 [2]. Further, more than 50% of mobile data traffic is predicted to be video [2]. To support the increasing data traffic, mobile network operators (MNOs) are reacting in several ways such as (1) increasing the capacity by adding more base stations (BSs), introducing Femto cells, enabling WiFi offload, and (2) introducing variable pricing mechanisms such as moving from unlimited data plans to tiered mobile data packages.

This paper explores a complementary, less explored direction of better yield management through network-managed time-shifting of traffic. Despite the challenge of rise in traffic, networks often observe significant variation in utilization levels, mainly triggered by diurnal patterns of human activity [3],[4],[5]. In fact, our measurements show that flows observe high variation of throughput even at short timescales of a few seconds. Such variations cause low utilization at certain times of a day and overload during others. Overloaded conditions can have a drastic effect on the quality-of-experience (QoE) of users, especially when streaming videos, thus lowering the effective yield of the network resources. The yield of a given network deployment is a function of the number of “useful” bytes delivered in time without deteriorating the QoE to users.

Fig. 1. Async Approach

Building on the above observations, we introduce an asynchronous content delivery system called Async for flexible content delivery in a mobile operator’s network. Async facilitates an MNO to negotiate a delay in content delivery with users in exchange for appropriate incentives. The negotiated delays are used by the MNO to actively manage Async flows in such a way that QoE improves for both delayed and regular flows. We specifically focus on video content delivery to cellular clients since video traffic is dominating cellular data traffic [2], and video content (e.g., prime-time shows, movies, sports highlights) is amenable to asynchronous delivery. Nevertheless, the solution is equally applicable to other types of traffic that are elastic, such as synchronizing files to the cloud, video uploads, and software updates.

Fig. 1 shows the basic idea of Async. Async essentially introduces a “video sachet” model, in which users (or automated user agents) are allowed to choose the delivery price and expected delivery time (EDT) of different types of videos at fine granularity. On each video request, the Async system presents the user with different (price,EDT) options to choose from. In the case of a user agent making the choices, the user can set policies a priori to specify the range of tolerable price and EDTs for different types of traffic.

Async system is based on two simple but crucial observations: (1) the available capacity at BSs fluctuates significantly even at short timescales, hence, cellular network elements are in the best position to intelligently schedule the traffic with varying EDTs, and (2) users are better judges of the price they are willing to pay and the EDT they can tolerate. Often, user behavior depends not just on prices and delays, but also on the specific content (entertainment video vs. business video), and the context of content consumption (e.g., watching alone vs. watching in a group). Consequently, Async attempts to enable a variety of price and EDT options to cater to such diversity of content and context.

Further, by managing price and delivery time options at fine granularity, Async enables the MNO to continually operate at different load levels, and evolve as traffic increases. For instance, when the overall utilization is low, prices can be lower for video content to increase adoption, and prices for
delivering the same content can be increased as the average utilization increases with time. More generally, the flexible delivery model enables systematic de-peakng of network traffic and significantly increases the yield of a given deployment while satisfying users’ QoE expectations, thereby increasing the time-to-upgrade for network infrastructure.

Thus, Async takes a holistic view and effectively combines capacity utilization with dynamic pricing to increase the yield of a network. Using a user behavior model proposed in prior work [2], we simulate Async functionality in ns2 simulator for several input traffic patterns and degrees of traffic predictability. We demonstrate the capability of network-based flow management by showing that in comparison to state-of-the-art methods of deferring requests like TUBE [4], Async enhances QoE for ~30% of the regular flows, with up to 60% improvement in the per-flow QoE metric, while still meeting the negotiated delivery times of 95% of the deferred flows. We also show that Async’s flow management reduces the delivery times of delayed flows by up to 67% and is significantly more robust to traffic unpredictability. Through a prototype evaluation on an operational cellular network for more than two weeks, we show that Async is (1) deployable with no modifications to existing protocols and network infrastructure except for adding rules into a policy charging and rules functions (PCRF) [2] system for setting discounted pricing policies, and (2) can effectively meet the advertised EDTs in practice.

In summary, we make two main contributions.

1) We propose a novel content delivery model over cellular networks that can alleviate network congestion and increase the yield of network deployments, while empowering the users to make a choice of the QoE. Both qualitatively and quantitatively, we show that the network managed delivery in Async enables a more effective way of network yield management, compared to other variable-pricing solutions [1].

2) We present the design of the first end-to-end system called Async. We design a light-weight and distributed content delivery protocol that is adaptive to fluctuating traffic, and robust to disconnections. We then design an EDT-aware flow scheduling algorithm which can meet the promised EDTs while limiting the interference to non-Async (or regular) flows to a specified threshold. Finally, we propose a simple solution for efficient price and EDT computation in Async that has the key property that the prices offered are monotonically non-increasing with increasing EDTs, thereby simplifying the task of making choices by users or user agents.

The rest of the paper is organized as follows. In the next section (Sec. II), we present the opportunities, challenges and overview of the Async delivery model. In Sec. III, we describe the three key components of Async: a content delivery protocol, a threshold-based chunk scheduler, and a dynamic pricing module. Sec. IV discusses the implementation of our Async prototype. Sec. V presents our detailed simulation study and an evaluation of our prototype. Sec. VI discusses related work, while Sec. VII concludes.

II. Async: Opportunity, Challenges, and Overview

Async essentially considers that traffic is of two broad types: (1) traffic for which users do not prefer to wait for any reason, and (2) traffic for which users may choose to wait (time-shift), possibly for incentives, or because it leads to increased QoE for their other traffic.

User Study: To understand how users react to time-shifting, we conducted a Web-based survey of users’ responses to different questions [5]. The survey is ongoing with current responses from only about 100 people, and hence the results should be taken as only indicative and not conclusive. The set of people has diverse population including engineers, Internet-savvy non-engineers, people active on social networks such as Facebook, and other tech-savvy users, who use cellular network to access and consume the content.

Fig. 2 shows that more than 50% users, when choosing delays, are more sensitive to discounts offered, in comparison to specific content type or time of day. In fact, more than 20% users in our survey were willing to delay the download (by 1 to 6 hrs) for an EDT promise for better QoE, even without any discounts. This observation hints at the impression users carry about the lack of enough capacity in the network for displaying video with good QoE [4]. Fig. 3 shows that only about 10% of the people prefer to receive all content with no extra delay. Others show increased tolerance to different content and transfer types. As expected, people are more willing to tolerate longer delays for larger transfers. Also, more than 50% of the users are willing to delay video and photo uploads, possibly since they themselves do not consume the content. We also observed that most users are willing to wait for 2 hours if given 50% discount, but not as many users are willing to tolerate higher delays even for 75% discount.

a) Design Elements: We now identify some challenges in building a system for time-shifting traffic, and derive the design elements that Async is built on. Firstly, time-shifting entails moving traffic demand from one instant of time (esp. times of peak load) to a different time in the future. Time-shifting traffic can be done in two ways: (1) Request-shift: by asking (i.e. managing) users to return back at a certain period of time (when the access will be at a discounted price) and make their object requests again, and (2) Delivery-shift: by collecting the object requests immediately and providing a longer-time-frame estimate (for a reduced price of network access) of when the object will be delivered.

The Request-shift approach removes the requirement of any state maintenance for outstanding requests, and is proposed in the literature as TUBE [4]. However, due to lack of network-management, there is no control of how much loaded the network will be during the later low-price periods due to other non-managed users, and how many users will be able to return at that period; this could lead to either overload or underload in low price periods. For instance, a simple experiment of measuring achievable throughput over different
days at the same times of day at a commercially deployed base station (BS) shows significant variation and lack of any temporal predictability of traffic at fine granularity, as demonstrated by Fig. 4. Instead, Async chooses the Delivery-shift approach, which enables content transfers to utilize finer timescale fluctuations in capacity. Essentially, by choosing the Delivery-shift approach, Async decouples the choice of when the object gets delivered from when the transfer is done, and lets the user choose the former, while the network gets the flexibility to manage the latter.

Secondly, Delivery-shift leads to long gaps between when the request is made and when the content is actually delivered, thereby increasing the number of outstanding requests in the system. Further, mobility and other human activity can result in disconnection periods and change of network access points (BSs) and network addresses. To achieve robustness to such issues, Async includes an application layer protocol for the management of content transfer, with the state of the content transfer managed by the client in an App. Finally, users often find it hard to comprehend scheduling and pricing choices on a per-object basis, unless the model is very simple. Also, simplicity also enables automation through policies enacted by user agents. Consequently, Async includes a pricing mechanism that exhibits the property that increasing EDTs strictly lead to non-increasing prices.

Building on these design elements, Fig. 5 shows an overview of the deployment of Async. In contrast to non-Async applications that access content from content servers directly, Async apps on mobile devices access content through the Async proxy. The proxy fetches content from servers that either modify the overall content presentation to the user to be Async-friendly, or let the apps download video content directly and view with local players. The proxy can be positioned anywhere in the MNO’s network, as long as it is accessible to the mobile applications through the HTTP protocol. Async can be used in several deployment scenarios. For instance, operators in growth markets such as Airtel already distribute movie and educational content on their networks with pay-per-object model. Async allows them to deliver this content for reduced prices at off-peak times. In another scenario, service providers retain the content but adapt their services to deliver content asynchronously. For instance, Netflix is already providing Cablevision with servers directly in the latter’s network to manage Netflix content.

III. Async Design

Async categorizes requests into two types: those on which users are not willing to wait at any cost (hereafter termed regular flows), and those on which users can wait, given incentives (hereafter termed delayed flows). All flows setup by the Async apps are delayed flows, with different EDTs. De-peaking traffic by time-shifting essentially involves scheduling the delayed flows in such a way that the QoE of regular flows increases due to time-shifting of delayed flows, and the delayed flows complete by their expected delivery time.

In what follows, we first describe the messages exchanged between a client and the proxy for satisfying each request, and then describe the algorithms at the client and the proxy, followed by a brief description of our pricing algorithm.

A. Client-proxy protocol

For every download by delayed flows, the Async App initiates a request message to the the Async proxy, much like traditional Web proxies. The request includes a unique application-level clientID, BSID, and a video objectID. The clientID can be a function of subscriber and device identification numbers and is used by the proxy to track requests and other status messages on a per-client basis. The BSID can be easily extracted in operating systems such as Android and is used to track all the delayed flows through a BS. The video objectID is the URL to the video file which the user wants to access and is used to fetch the content from the original content server in the background. For a request message, Async proxy responds with a set of (price, EDT) options for the user (or the user agent) to choose from. The Async-App then sends a (price, EDT) choice to the proxy. These messages are exchanged in the negotiation phase on a control channel, which can be enabled using a set of HTTP-based scripts. The price chosen is used by the Async proxy to bill the flows appropriately (explained in Sec. III-D), and the EDT is used by both the Async Proxy and App to coordinate the data transfer (explained in Secs. III-B and III-C). Once a choice is made, both the client App and the proxy switch to the transfer phase in which the content is delivered over one or more HTTP sessions set up in response to HTTP range requests issued by the client. This is shown in Fig. 6.

Since EDTs can make the transfers spread over long periods of time, with potential user mobility, disconnections etc., we introduce a message, BS UPDATE. This message is sent by an Async-client to the proxy whenever the client perceives that the user device has moved to a new BS. This can be easily done in today’s smartphones; e.g., we implement an event on_cellID_change in Android, on which the Async-client can wait and send an update in response. After the necessary status update, the client and proxy coordinate to transfer the content through HTTP sessions. To provide robustness against connection failures, the client maintains the download state, and invokes HTTP range requests to resume the transfer.

B. Client: Deadline-aware Probabilistic Polling

A new HTTP range request is in fact a poll made by a client for the partial content that has not been received yet.
For a flow that requires transfer of \( B \) bytes, we define a reference delivery progress function \( D(t) \), which indicates the percentage of total bytes to be delivered by time \( t \). A number of functions can be used to represent \( D(t) \), including linear or quadratic functions such as \( \frac{Bt}{T} \) or \( \frac{Bt^2}{T^2} \). Note that a quadratic function lets the delayed flow's yield more to regular flows compared to a linear function when the EDT is farther away. A sample linear delivery function and a leading and lagging scenario for a flow are shown in Fig. 9(a). The Async client tracks the progress of each flow in terms of the number of bytes \( b(t) \) that have already been received at time \( t \). A flow is termed as lagging if at any instant of time \( t, b(t) < D(t) \), and leading otherwise. We then define a polling probability function given by: \( p(t) = \min\left(\frac{B-b(t)}{B-D(t)}, 1\right) \).

To enable low polling overhead at the Async proxy, we employ a probabilistic polling mechanism. When a flow is suspended, at each epoch boundary, the client picks a random number \( r \) in \([0.0, 1.0]\) and polls the proxy with an HTTP range request only if \( r < p(t) \). Thus, \( D(t) \) is used by the Async client to control the polling frequency while ensuring that the EDT is met. Such randomization in polling essentially helps avoid “poll-bursts” at the proxy when most of the delayed flows are suspended due to high load of regular flows at a BS. Once a poll is admitted, the flow moves from suspended state to active state, till the transfer completes or the proxy terminates the HTTP session. If a poll is disallowed, the client sleeps for this epoch, and repeats the same process at the start of the next epoch. This client-side polling algorithm is summarized in the pseudo-code in Fig. 7.

C. Proxy: Deadline- and interference-aware Scheduling

For each incoming poll from the Async clients, the proxy makes scheduling decisions (admit or disallow) on a per-BS basis. The scheduling decisions for a BS are governed by the degree of interference caused by the delayed flows on regular flows. Interference is termed as the reduction in throughput observed by a regular flow when a new delayed flow is introduced. Async attempts to bound the interference by disconnecting the flows during BS overloads. However, as mentioned in the previous section, BSs often face short time-scale variations in the load due to regular flows. Hence, once disallowed, a flow should be resumed as soon as possible whenever capacity becomes available at the BS to utilize the opportunities for content transfer.

To minimize interference and enable opportunistic content transfer, we introduce a parameter \( \tau \), representing a threshold throughput. To observe the load at a BS, the Async proxy continuously keeps track of the (EWMA of) throughput \( \theta \) achieved by each flow active at the BS. The proxy uses the following simple rule to make decisions: at any instant of time, continue the content transfer if the flow is achieving a throughput of \( \tau \) or more, or if the flow is lagging behind significantly that there is danger of missing the EDTs; otherwise, suspend the flow by closing the HTTP session. As shown in the evaluation section (Sec. V.B), it takes only a few seconds (order of 12
seconds) to observe the achievable throughput (AT) to make a scheduling decision thus limiting the interference to a short duration before a flow is disconnected. Further, after each epoch (order of a minute), a suspended client polls back again probabilistically as described in the previous section, allowing the proxy to observe the new AT of the flow, and thus utilize the short time-scale transfer opportunities.

Cellular BSs commonly use proportional fair (PF) scheduling to balance aggregate system throughput against flow fairness. Since one-way delays for a flow do not increase in PF scheduling until each flow achieves its fair share of the bandwidth, it is not clear how the capacity variations can be detected using commonly used techniques such as [12], [13]. In comparison, our approach ensures that if an Async flow is receiving a throughput of $\tau$ or more, then regular flows with similar channel conditions will also receive at least $\tau$ as long as they have traffic to transmit. Furthermore, since cellular access links are often the bottlenecks, Async proxy can be placed anywhere in the operator’s network to observe the throughputs and make the scheduling decisions.

To choose an appropriate value for threshold ($\tau$), we analyze the historical achieved throughputs (AT) observed at the BS. A high AT indicates that the flow has a good channel quality and there is large available bandwidth at the BS. Hence, we derive a cumulative distribution function (CDF) using the achievable throughput values measured over several days and choose the median value as $\tau$. Thus, $\tau$ is a statistical metric that we expect not to vary significantly across days. Once the system is instantiated by these measurements and the corresponding threshold, as more and more users use the Async system, the AT values measured by the proxy shape the CDF further and tune the $\tau$ value.

No interference: As shown in Fig. 10 (b), when $\tau > \frac{1}{2}C$, where $C$ = maximum capacity of the BS, with PF scheduling, it implies that whenever a flow is scheduled by Async, content is served without causing any interference to regular flows. Thus the degree of interference can be tuned in Async using an appropriate choice of $\tau$ relative to BS capacity. Algorithm 8 summarizes the deadline- and interference-aware scheduling.

D. Pricing

We now present a scheme for computing (price, EDT) options for flows arriving at the Async proxy for negotiation with the end users. While such options can be computed in several different ways, we give a simple scheme to illustrate and enable pricing of content in Async. The EDT choices presented to end users can be a small set of pre-defined durations, such as 2, 4, and 6 hrs, depending on the context and the length of the request or other criteria. For each EDT, our pricing framework computes a price by determining the congestion caused by the aggregate traffic due to the incoming request. To determine the cost of congestion, we compute an allocation of the incoming request over time considering an EDT in the manner of water-filling algorithm [14]. This is illustrated with an example in Fig. 11.

In Fig. 11, a new Async request arrives at time $t_i$. The aggregate traffic due to regular flows at any time $t$, denoted $F_i$, is as shown. This quantity is assumed to be known and can be estimated offline, for instance, by the same module that estimates $\tau$, or using aggregate traffic estimation. For EDTs 1, 2, and 3 considered for the incoming Async request, the distribution of the request’s content that minimizes the aggregate traffic level is also shown in the figure. This is accomplished by allocating the incoming content such that the total traffic is equalized at all times in which the incoming request receives allocation (by, for instance, using a strictly convex function for computing congestion cost). It is easy to see that the maximum traffic level cannot increase with increase in EDT. In our scheme, the price proposed for an EDT is a function of this resulting traffic level. Note that, the Async proxy spawns a separate instance of the pricing algorithm for every BS under its control since the traffic patterns at different BSs can be different.

Assumptions and definitions. Time is assumed to be divided into discrete time periods of configurable duration. $C$ denotes the BS capacity in terms of total bits that can be transmitted in a time period. The total number of Async flows that have arrived by time $t$ and active at $t$ is denoted $Y_t$, and the number of bits allocated to Async flow $i$ in period $t$ is denoted $y_{i,t}$. The total traffic at the BS at time $t$ is then given by $\lambda_t = F_t + \sum_{i=1}^{Y_t} y_{i,t}$.

We classify the congestion experienced at a BS into one of a set of $L$ distinct levels. For this, the BS’s transmission capacity $[0, C]$ is partitioned into $L$ discrete sub-ranges, given by $[T_0, T_1), \ldots, [T_{L-1}, T_L)$, where $T_{\ell-1} < T_{\ell} \forall 1 \leq \ell \leq L$. The congestion level at a BS in period $t$ is given by the sub-range in which the total traffic it sees in that period, $\lambda_t$, lies, and is said to be $\ell$ if $\lambda_t \in [T_{\ell-1}, T_\ell)$. For each level $\ell$, we assign a constant but monotonically increasing congestion weight $K_\ell$. We then define a convex congestion-cost function (which is piecewise linear) at time $t$ as: $D_t = \sum_{\ell=1}^{L} K_\ell \max (\min (\lambda_t, T_{\ell+1}) - T_{\ell}, 0)$. It is easy to see that this function is non-decreasing with $\lambda_t$. Such a cost function allows the MNOs to choose different level thresholds ($T_\ell$) and appropriate cost for the level ($K_\ell$) based on their experience.

Allocating Async flows to minimize congestion cost: To compute a price for an EDT, we first formulate and solve the problem of allocating $Y$ Async flows over $N$ time periods (in the manner of water-filling) so that the aggregate congestion cost over time periods is minimized. This optimization problem is formulated as follows.

**MINCONG**

\[
\begin{align*}
\text{Minimize} & \quad \sum_{t=1}^{N} D_t, \\
\text{such that} & \quad S_i = \sum_{t=1}^{N} y_{i,t}, \quad \forall i = 1 \ldots Y \\
& \quad \sum_{i=1}^{Y} y_{i,t} \leq C - F_t, \quad \forall t = 1 \ldots N \\
& \quad y_{i,t} \geq 0, \quad \forall t = 1..N, \forall i = 1..Y
\end{align*}
\] (1)

where $S_i$ is the size (in bits) of Async flow $i$. The constraints in (1) ensure that the BS capacity is not exceeded while the
flow is served completely. Note that even though the objective cost function is piecewise-linear, the problem can be solved optimally since the slope of the objective is bounded and all the constraints are linear. We assume that at least one of congestion weights and congestion level widths is strictly increasing and the other is monotonically increasing.

Computing per-flow (price, EDT) options: When a new flow arrives at the proxy, we solve the optimization problem MINCONG once for each EDT to be presented to the user (with \( N \) set to that EDT). In solving the problem, we use the allocations in the future periods to previously-accepted Async flows (obtained as solutions to instances of MINCONG solved when those flows arrived) and estimates of \( F \) (traffic due to regular flows). If the problem is infeasible for the largest acceptable EDT, then we declare that the incoming request cannot be scheduled and send a notification back to the client. We compute a price for EDT \( N, p_N \), as follows:

\[
p_N = \frac{\sum_{i=1}^{N} \sum_{t=1}^{T} D_i}{\sum_{i=1}^{N'} \sum_{t=1}^{T} D_i - \sum_{i=t}^{N} D_i} \quad \text{where } D_i = 0 \text{ if } y_{i,t} = 0
\]

To avoid a large value of \( F_i \) (regular traffic) for some \( t \) (with no Async traffic allocated) affecting the price computation; \( i \) is the new request under consideration. Note that \( p_N \) lies between 0 and 1, and can be used along with a base price. For instance, a fraction \( p_N \) of the base price may be charged for the request.

Proposition 3.1: The prices offered by Async pricing scheme are monotonically decreasing with increase in EDTs.

Proof: Let \( N \) and \( N' \) \((N' > N)\) be two EDTs, and let \( i \) be an incoming request. Now, consider MINCONG for EDT \( N' \) with the following additional constraints: \( y_{i,t} = 0, \forall t > N \). These constraints decrease the feasible space for MINCONG with EDT \( N' \), and \( p_{N'} = p_N \). Now, suppose the additional constraints are removed. Since the feasible space increases, and since MINCONG strives to minimize the objective value, \( \sum_{i=1}^{N'} D_i \leq \sum_{i=1}^{N} D_i \) where \( D_i = 0 \) if \( y_{i,t} = 0 \). Thus, it follows that \( p_{N'} \leq p_N \).

Proposition 3.1 ensures that Async users are offered higher incentives for waiting a longer amount of time, which, unlike models in prior work such as in TUBE [11], gives users an easy to understand interface: wait more, gain more [12]. Moreover, Async offers higher discounts to the requests arriving in low-congestion periods, thus incentivizing users to increase their usage.

IV. Async Prototype

We have realized Async design in a fully-functional prototype, relevant details of which are discussed below.

![Fig. 11. Components of the Async prototype.](image)

As shown in Fig. 11, the Async proxy consists of: Async control scripts in PHP which implement the proxy side of the content delivery protocol, a database module to store the achievable throughput values and compute \( r \) per BS, an Apache HTTP server, and the Async pricing module. The proxy listens on port 8081 for control messages and uses ports 8082 to 8084 for content transmission. The proxy can download an object from the content server using Apache’s mod_proxy module. As the content becomes available (partially or completely), it can be served to the Async clients. We assume that the download rate from the content server to the proxy is much greater than that between the proxy and the clients, since the latter includes the cellular link, which is more constrained. When the object download completes at the client, it can be consumed, e.g., using an embedded video player in the client App in the case of a video object.

For ease of integration of pricing in the operator’s network, we enable a small set of ports at the Async proxy for serving the content to the clients. Each port maps to a price-per-byte rule (src-id=proxy-IP and src-port=) into the PCRF. While presenting (price, EDT) options to users, we choose the closest matching price in the set of prices in the pricing rules, and assign the corresponding port to serve the Async flow. This approach is minimally intrusive to the cellular network since it involves one time addition of only a small set of rules to the PCRF. Furthermore, a control port can be enabled for the negotiation phase without charging for the negotiation messages.

V. Evaluation of Async

A. Simulation-based Study

In this section, we describe our simulation experiments to evaluate the efficacy of the Delivery-shift approach of Async and the Request-shift approach of TUBE [11] (refer Sec. II) in reducing congestion and improving QoE by time shifting the flows which can be delayed. We also implement a baseline scheme that does not manipulate or shape traffic in any way, referred to as unsmoothed.

Performance Measures: Given our objectives of a fast delivery of the delayed flows, and an improved experience for regular flows (by alleviate the congestion), we focus on two performance measures: (1) total delivery time and (2) buffer underruns. Total delivery time is defined as the difference between the time the user intends to issue a request and the time at which the request is fully served (i.e., all the bytes of the requested object are delivered at the client). Total delivery time is indicative of the extent to which a scheme time shifts traffic to ease congestion. Buffer underruns are indicative of the interference that a regular flow experiences due to congestion and we determine it as follows (for only regular flows): The total bytes received by each flow is sampled every 10 secs, and if found to be less than the expected amount (based on the playback/desired download rate), the underrun count for the flow is incremented by one. In the case of video, the fewer the buffer underruns for a flow, the fewer the number and/or duration of playback stalls, and hence the higher the QoE.

Simulation Setup: The content delivery and threshold-based scheduling components were implemented using ns-2 [12]. High Speed Downlink Packet Access (HSDPA) [12] was chosen as the cellular protocol for the wireless link and its features were simulated using Enhanced UMTS Radio Access Network Extensions (EURANE) [13] for ns-2. EURANE provides support for Universal Mobile Telecommunications Sys-
specified in the literature. Periods are called minute intervals called delivery without delay like a regular flow.)

the cumulative non-deferrable foreground traffic (regular traffic either a regular flow or an Async flow). Uniformly within an period. Each flow generated is marked as "regular traffic. Flow arrival times are distributed over the duration of the aggregate load. In generating the flow requests, we specify the percentage of load that comprises regular traffic. Flow arrival times are distributed uniformly within an period. Each flow generated is marked as either a regular flow or an Async flow and is of one of the four content types: web request, short video, long video, and file download. The mean sizes for the four types are chosen from different sets, e.g., 200 KB, 2 MB, 50 MB, and 10 MB. The playback rate for each video flow is uniformly distributed between 300 and 700 kbps and the desired download rate for a web request/file download ranges between 400 and 1100 kbps. These settings are used at run-time to track the progress to the flows and to quantify the interference that they are subjected to by determining the deficit to the total bytes delivered in comparison to the expected number of bytes.

At run-time, regular flows and non-delayed Async flows (that is, flows amenable for deferral but chose not to) are scheduled at their arrival time. The remaining Async flows (delayed flows) are scheduled using the method described in Sec. III. Epoch duration for the delayed flows is set to 1 min (based on measurements on temporal validity of achievable throughput described in Sec. 

The Async pricing component was implemented as a stand-alone module. This module takes as input details of flows that are amenable to deferrals (the Async flows), and an estimate of the cumulative non-deferrable foreground traffic (regular traffic given by $F_t$ in Sec. III.D), and determines the delivery method (NOW or deferred) and, if deferred, EDT, for each Async flow. (Async flows for which deliver NOW option is chosen will be delivered without delay like a regular flow.)

Simulations were performed as follows. We consider 30-minute intervals called periods. We first choose an aggregate traffic load for 48 continuous periods spanning 24 hours (as specified in the literature or synthesized to evaluate performance under various load patterns). We next generate flow requests per-period to match the aggregate load. In generating the flow requests, we specify the percentage of load that comprises regular traffic. Flow arrival times are distributed uniformly within an period. Each flow generated is marked as either a regular flow or an Async flow and is of one of the four content types: web request, short video, long video, and file download. The mean sizes for the four types are chosen from different sets, e.g., 200 KB, 2 MB, 50 MB, and 10 MB. The playback rate for each video flow is uniformly distributed between 300 and 700 kbps and the desired download rate for a web request/file download ranges between 400 and 1100 kbps. These settings are used at run-time to track the progress to the flows and to quantify the interference that they are subjected to by determining the deficit to the total bytes delivered in comparison to the expected number of bytes.

At run-time, regular flows and non-delayed Async flows (that is, flows amenable for deferral but chose not to) are scheduled at their arrival time. The remaining Async flows (delayed flows) are scheduled using the method described in Sec. III. Epoch duration for the delayed flows is set to 1 min (based on measurements on temporal validity of achievable throughput described in Sec. 

To effectively compare Async’s Delivery-shift approach and the Request-shift approach of TUBE, we implement the pricing scheme described in [8] to determine TUBE’s per-period discounts for the specified aggregate loads. We also implement the user model described therein to determine the per-flow shifts, that is, the number of periods after which a flow is willing to return. For fair comparison, we use the same user behavior model in Async to select a (price,EDT) option. Flow transmissions were simulated using ns-2 and EURANE using the outputs obtained from the Async and TUBE pricing modules. It should be pointed out that the approach described in [8] computes byte-level shifts, i.e., the fraction of bytes that shift from one period to a later period. Since a flow cannot be partially shifted, we adapted their approach to compute per-flow shifts.

We performed our evaluation under both when the actual traffic (1) followed predictions and (2) deviated from the predictions. Evaluation under predictable traffic conditions is required primarily for comparison against TUBE, since TUBE’s per-period prices are computed offline using estimated traffic.

Results under predictable traffic conditions: Performance under Async with two different interference thresholds ($\tau$), 700K and 900K (referred to as Delivery-shift-700K and Delivery-shift-900K, respectively), the Request-shift approach, and a
basic scheme that does not manipulate or shape traffic in any way (referred to as unsmoothed), are plotted in Fig. 14. In this first experiment, all flows experience ideal channel conditions and the input traffic pattern is as shown in Fig. 14. We chose the threshold values based on the BS capacity of $3.5$ Mbps under ideal channel conditions and the maximum per-flow bandwidth of $1.2$ Mbps in EURANE. Fig. 14(a) plots the cumulative bandwidth consumed by all the flows on a period-by-period basis. During the traffic peaks seen until period 27, the two Delivery-shift (Async) schemes and the Request-shift scheme lower traffic to different extents. As can be seen, the Request-shift scheme is the most aggressive of the three in reducing traffic and Delivery-shift-700K, the least. Consequently, Request-shifting shifts the most amount of traffic to the non-peak period after period 35. This is as expected since Async lowers traffic during peaks that are detected at run-time by limiting the transmission of deferrable flows (based on their EDTs) but not turning them off entirely. Further, as discussed earlier in Sec. 1, Async makes use of capacity that becomes available at shorter time scales to make progress with Async flows. On the other hand, under TUBE’s Request-shift paradigm, certain flows arriving at high-price periods decide to go away and return to issue their requests at low-price, and hence, presumably, low-traffic periods that are determined offline a-priori.

**Buffer underruns, delivery times, and meeting EDTs:** Insets (b) and (c) of Fig. 14 plot the CDF of buffer underruns and delivery times experienced by the flows, respectively. As can be seen, more than $15\%$ fewer flows experience buffer underruns under Delivery-shift than under Request-shift, with $\sim 30\%$ flows experiencing fewer underruns. The maximum number of underrun instances for any flow under Delivery-shift is $\sim 33\%$ of that under Request-shift. This is due to the following reason: The throughput plotted in Fig. 12 is smoothed over 1-hour duration. The throughput is more bursty over shorter timescales, exhibiting many short-term peaks. Under Async, delayed flows back-off during such times, minimizing the interference and impact to regular flows. Such flow discrimination and control is absent under Request-shifting, and hence, the flows experience larger number of underruns. Thus, Delivery-shift de-peaks the congestion in an effective manner and performs better than state-of-the-art in lowering the interference and enhancing the QoE. Also, as shown in inset (c), the extent to which flows are deferred under Request-shifting is significantly higher than under Delivery-shifting. This is because, as discussed earlier, Delivery-shifting does not wait until an advertised low-price, low-traffic period (which is determined at coarse granularity) to start a deferrable flow, but rather makes use of transmission opportunities that become available at shorter time-scales to make progress with deferrable flows. Thus, under the metrics chosen, Delivery-shifting is more effective in alleviating the effects of congestion since spectrum does not remain unused when there is opportunity for transmission. Finally, although the Delivery-shift approach delays flow deliveries, most of the flows are delivered by their EDTs, and these EDTs are comparable to (and, in many cases, shorter than) the per-flow shifts under TUBE. Both the number of flows not delivered by their EDTs ($< 1\%$) and the amount of additional time taken are negligible. Just as Async EDTs are comparable to TUBE’s per-flow shifts, the incentives provided by the MNO are also comparable in both cases. Thus, Async de-peaks the congestion in a better fashion within the same price budget as that of TUBE. We omit the revenue results due to lack of space.

**Results under unpredictable traffic:** In the above experiments, deferral times under Request-shift and EDTs under Delivery-shift are determined assuming complete predictability, i.e., full knowledge of aggregate per-period traffic for Request-shift and regular traffic for Delivery-shift. Such absolute predictability cannot be assured in practice, and hence, we next study the impact of deviations from this assumption. For this, we performed experiments wherein the actual traffic is perturbed in comparison to the assumed traffic. We tested with two types of perturbations. In the first type of perturbation, referred to as overload, during the non-peak period after period 30, the actual aggregate traffic is higher, but follows a pattern similar to the assumed traffic; traffic in earlier periods remains unchanged. In the second type of perturbation, referred to as shift, the actual traffic is shifted by a certain number of periods in comparison to the assumed traffic.

For the overloaded case are shown in Figs. 14(d)-(f). Observe that, due to lack of network-side control, the Request-shift approach is unable to react when the actual traffic deviates from the estimated one and prevent deferral of flow requests to periods later than 35. As a result, flows shifting to perceived low-traffic periods suffer from significantly higher buffer underruns. On the other hand, degradation under Delivery-shifting is much less severe. This is due to (1) the run-time determination of EDTs and discounts with Delivery-shifting, which is better able to react to deviation to traffic, and (2) network managed delivery, which is better able to utilize the resources that become available at short time scales.

Fig. 13. Traffic pattern for the experiments with results in Fig. 14. The throughput results of TUBE pricing evaluation presented in 10 are smooth and not as bursty as those in Fig. 14. This is partly because (1) their approach operates at byte-level and assumes that a fraction of traffic seen in one period can be shifted to a later period; such perfect shifts are not possible when operating at flow-level, at which shifts occur in practice, and (2) their aggregate traffic is much larger due to aggregation at a much higher level than a BS.
Results with traffic shifts are shown in Fig. 12. As can be seen, under Request-shifting, both the mean number of under-runs per flow and its variance increase as the extent of shifting increases. On the other hand, the Delivery-shift approach is more robust, for reasons provided above.

**Results with non-ideal channels:** The above experiments were repeated for varying channel conditions and different degrees of mobility for users. Results obtained exhibited similar trends as above. We omit presenting detailed plots due to lack of space.

**Impact of Async's interference threshold (τ):** In the above results, as expected, opportunities for transmission are lower for a higher τ, and hence, delivery times are longer for τ = 900K in comparison to that under 700K. Consequently, interference to regular flows is lower with the higher τ of 900K.

We repeated the experiments for other input traffic patterns, and, in general, obtained consistent results.

**B. Prototype Evaluation**

Using the prototype that we developed (described in Sec. VI), we characterize the performance of our Async system in live networks. We conducted a pilot study and collected data from five different users over 17 days, with Android-based Samsung Galaxy phones. Our study involves four 3G BSs of two leading service providers (supporting more than 100 million users) in Bangalore, India. We instrument the Async client App to automatically download a video file with an EDT of {1, 2, 3} hour for three times a day between 8 a.m. and 7 p.m. For lack of space, we only report our measurements for two BSs (referred to as B1 and B2), from different service providers. B1 is in a residential area while B2 is in a commercial area. Through our prototype-based evaluation, we answer the following questions: What is an appropriate duration to observe a reliable achievable throughput (AT) value and is it reasonable enough that it is non-interfering with regular flows? What is a reasonable estimate of threshold τ? How do the CDF of AT and τ values vary over days? How many polls does a client make to download a content for a given threshold τ and an EDT value? How many downloads meet the EDT?

**Duration to observe AT:** For this experiment, we download a 77MB video file by varying the observation duration from 2 sec to 20 sec. We let the client app continuously poll without any gap and repeat the experiment on two days at different hours. As shown in Fig. V-B, for probing durations ≥ 12 sec, AT reaches its limit and remains fairly stable. This value can be used by the Async proxy to observe AT values and limit the interference to a short duration. Thus, for the rest of the experiments, we choose 12 sec as the probing duration. Based on another set of measurements, we observe that most of the connections maintain the AT values for a duration of one to two minutes before facing any deviation. Based on this observation, we set the epoch duration approximately to one minute. We thus use epoch duration of a minute in our simulations.

**Achievable throughput CDF and τ:** In Figs. V-B and V-B, we plot the CDF of AT on three weekdays for B1 and four weekdays for B2, respectively. These measurements show that, for a given BS, the CDF of achievable throughput values remains reasonably predictable. Based on the measurements, we compute the median value of the CDF as the threshold τ. For B1, τ = 412 Kbps while for B2, τ = 554 Kbps, which also confirms a well-known observation that a BS in commercial areas is more loaded than a BS in residential areas.

**Meeting EDTs** In a controlled experiment, we initiated multiple transfers of a single file at the same instant by three devices with three different EDTs through B1. As shown in the Figs. V-B and V-B, the Async system distributed the polls and hence the downloads over time and aligned them as per the EDTs, in an attempt to minimize the overhead of polls at a given time. As shown in Fig. V-B, for a given τ, the EDT required to deliver the content can be reasonably met for different requests (of different file sizes). Thus, a service provider can vary τ to control interference while trying to meet the promised EDTs. Over 17 days across five users, the Async proxy experienced 57 downloads and only 7 downloads exceeded the promised EDT (6 having a 1 hr EDT and one a 2 hr EDT), resulting in a success rate of 87%. All of the exceeded requests completed within 15 mins after the promised EDTs.

**VI. RELATED WORK**

We discuss prior work related to Async in three broad categories: (1) managing demand for data in cellular networks, (2) background transfer with controlled interference, and (3) enabling dynamic pricing in communication networks.

**Managing data growth in cellular networks.** A simple and commonly employed option to reduce peak-load is capping the usage. While this is unpopular with users to begin with, as reported in [3], the bandwidth consumption gap between tiered and unlimited data plans is narrowing with the increased consumption of cloud-based services. Another approach is to use WiFi or Femto-offloading [20]. This approach is complementary, and also benefits from Async to maximize the yield of such alternate network deployments.

**Background transfers.** Background transfer mechanisms such as TCP-Nice [22] and TCP-LP [23] may be extendable to achieve the same notion as Async in wireless networks. However, the techniques depend on increased delays as an indication to backoff background flows for causing minimal interference to foreground flows. This idea does not work well in cellular networks for two reasons: (1) delays also vary due to wireless channel scheduling, channel quality fluctuations, etc., (2) proportional fair scheduling at basestations results in background flows getting their fair share of resources before they start perceiving congestion-related delays, thereby causing interference that scales with number of background flows.

**Congestion-aware pricing.** Differential pricing to combat congestion and share network resources efficiently is a well-studied topic in multiple disciplines [24], [25] and [26] propose to smooth traffic by providing incentives of the form of higher access rates during off-peak hours. However, these schemes miss fine-timescale opportunities of capacity availability in cellular networks. Decoupling selection of EDTs from scheduling of flows before the EDTs (i.e. delivery-shift approach) significantly improves the chance of utilizing the opportunities. More recently, TUBE [6] has set a nice background for renewing this direction of research in the cellular domain. However, as mentioned in Sec. I, TUBE falls under the Request-shift category, operating at coarse granularity and does not also incorporate delivering content within any expected time.
Fig. 15. Stabilization of probe throughput over time.

Fig. 16. Evolution of CDF of achievable throughput for B1.

Fig. 17. Evolution of CDF of achievable throughput for B2.

Fig. 18. Progression of downloads as per EDTs.

Fig. 19. Download rates as per EDTs.

Fig. 20. EDT choices for different file sizes.

VII. CONCLUSION

In this work, we proposed an approach to enable asynchronous content delivery for smart data pricing in mobile networks. We believe that the approach proposed in the paper is only illustrative, and various versions of delayed delivery are interesting avenues for future work. The applicability of delayed delivery for better network resource management is broad. For instance, with the emergence of significant content or personal cloud updates.

REFERENCES