Legilimens: An Agile Transport for Background Traffic in Cellular Networks

Muhammad Usama Chaudhry*, Shibin Mathew*, Shanyu Zhou*,
Vijay Gopalakrishnan†, Emir Halepovic†, Hulya Seferoglu*, and Balajee Vamanan*
*University of Illinois at Chicago (UIC); {mchaud30, smathe36, szhou45, hulya, bvamanan}@uic.edu
†AT&T Labs – Research; {gvijay, emir}@research.att.com

***Abstract*—Large data transfers can result in significant congestion and performance degradation for interactive end-user applications such as web browsing and streaming. While there are existing TCP congestion control algorithms for delivery of large volume data (e.g., LEDBAT, TCP-LP), our results show that these protocols are not effective in cellular networks due to variability in radio channel conditions and the use of cellular schedulers in base stations. We propose *Legilimens*, an agile TCP variant for *cellular downlink* transfers, which not only retains desirable properties of existing approaches, but also exploits the properties of the cellular schedulers to estimate load and capacity and addresses the challenges in cellular networks. As a result, *Legilimens* is able to deliver traffic using only the *spare* capacity on the downlink. We conduct extensive evaluations of *Legilimens* in multiple settings—in a large cellular network for real-world performance, on the PhantomNet emulator for controlled experiments, and ns-3 simulator for scaled experiments—all of which demonstrate that *Legilimens* is superior to existing protocols in transferring large volumes of data without interfering with regular user traffic. Compared to existing low-priority protocols, *Legilimens* improves the throughput of background flows by 2x on average (up to 5x) without degrading the performance of foreground flows across all the three testbeds.**

***Index Terms*—Cellular networks, Proportional fair scheduler, LTE, Background traffic, Low priority transport, TCP**

I. Introduction

Cellular networks are increasingly used not just for interactive end-user applications such as web browsing, chat, social networks and video streaming but also for transferring large volumes of data, especially on the downlink (e.g., cloud data sync and software updates [1]). Large data transfers can degrade throughput of users in the same radio cell by over 70% and negatively impact nearby cells [1, 2]. When such transfers use fair-share protocols such as CUBIC, they compete with existing flows and may affect them adversely. While existing flows may not always result in unusable performance, they still perform worse than they would without such transfers; naturally, their effect is more pronounced at high loads. Therefore, techniques that allow for efficient delivery of large data transfers without impacting users' experience have become necessary.

There exist several possible approaches to address this problem. Some services, applications, and mobile operating systems allow users to restrict or schedule transfers. Unfortunately, these approaches have limited applicability because the approaches are agnostic of prevailing network conditions and users may prefer the convenience of up-to-date data. Another possibility is to statically rate limit large transfers. However, static rate limiting can cause under-utilization at low loads and overwhelm the network at high loads. Dynamically adjusting these rate limits is hard to realize correctly in practice, given the bursty nature of the traffic. Existing traffic management for cellular networks, in the form of QoS class identifier (QCI), is not sufficient because QCI requires tight integration between network and application, must be provisioned *a priori* by operators, and has no flexibility to move flows between classes.

Transport-layer approaches, exemplified by protocols like LEDBAT and TCP-LP, introduce the concept of *two-class* service prioritization. The key idea is to have a "low-priority" mechanism for delivering large volume or time-insensitive data. This allows typical interactive applications (considered foreground flows) to have fast response times using a *fair-share* TCP variant such as CUBIC or BBR, while simultaneously making progress on large volume transfers (considered background flows) using the lower priority TCP variant. LEDBAT and TCP-LP rely on either round trip times (RTT) or one-way delays (OWD) to infer congestion but our experiments with such Low Priority Transport (LPT) protocols show that they are not effective in cellular networks (Section II). We conjecture that they are not effective due to the nature of cellular networks—highly variable delays and the use of *Proportional Fair* (PF) schedulers with per-device queues for downlink transmissions.

Drawing inspiration from LPT protocols, we propose *Legilimens*, an agile LPT variant for cellular networks that delivers background traffic without adversely affecting foreground traffic. *Legilimens* shares the same goals as other LPT protocols—to use all available bandwidth when foreground traffic is not present, to not interfere with foreground traffic when it is present, and to quickly yield (rampup) when foreground traffic starts (stops)—but includes novel features that make it effective in cellular networks. Specifically, a well-designed LPT protocol for cellular networks must operate with minimal queuing, so that its packets do not compete with those of foreground flows at the scheduler.

Since cellular downlink is typically the bottleneck [3, 4], we design *Legilimens* for *downlinks*. *Legilimens* uses a novel approach to quickly estimate capacity and load based on packet inter-arrival times, not RTT or OWD. *Proportional Fair* (PF) schedulers are predominant on broadband data bearers in LTE (and 5G [5]) networks as they provide a balance between achieving peak data rates and serving users with poor signal quality [6, 7]. Our overall design leverages and is geared towards the generalized class of (PF) schedulers — however, we believe it can also generalize to other fair queuing mechanisms to some extent. The *key novelty* of *Legilimens* lies in how we leverage the downlink PF scheduler's unique *strength* of providing fairness at short time scales to counteract its *weakness* of interfering with the operation of traditional LPT protocols. Based on the estimated capacity and load, *Legilimens* strives to deliver background traffic using *only* spare capacity. *Legilimens* operates in one of two modes: normal mode and probing mode. If the load is low and spare capacity is available, *Legilimens* operates in normal mode and quickly captures available bandwidth. If the load is substantial, *Legilimens* enters the probing mode, where it yields all scheduling opportunities to other traffic most of the time, while periodically sensing the network until spare capacity becomes available.

We implement *Legilimens* in Linux as a sender-only modification to the network stack, enabling simpler and incremental deployment, without changes to the cellular infrastructure. We discuss deployment in detail in Section III-C. We extensively evaluate *Legilimens* in realistic (uncontrolled) and controlled settings, including a large U.S. cellular network, using PhantomNet [8] emulator and ns-3 [9] simulator.

In summary, we make the following contributions:

- We design a novel capacity and load inference algorithm for senders in cellular networks, which overcomes and leverages PF scheduling properties that impede traditional LPT protocols and we show that our algorithm is robust to sender- and receiver-side optimizations (e.g., batching, delayed ACKs).
- We design and implement *Legilimens*, a sender-side LPT protocol for cellular networks, which allows the network to balance the conflicting goals of foreground and background traffic while achieving high utilization and low congestion.
- We demonstrate that *Legilimens* achieves better overall performance than existing fair-share and LPT protocols. We show that fair-share protocols (e.g., BBR, CUBIC) adversely interfere with foreground traffic and degrade its performance, and that existing LPT protocols are not efficient in using spare capacity for background transfers. Our experiments in a real network show that *Legilimens* improves the throughput of background flows by a factor of 2.8 on average (up to 5.2x) over existing LPT protocols (i.e., LEDBAT and TCP-LP) without hurting foreground flows.

II. Background and Motivation

A. *Background on* PF *scheduler*

Figure 1 depicts the key infrastructural differences between wired/Wi-Fi networks and cellular networks. Cellular networks (Figure 1b) differ from wired/Wi-Fi networks (Figure 1a) in three key aspects: (1) cellular links exhibit time-varying capacities and delays; (2) cellular networks have separate queues to isolate flows from different devices; (3) cellular networks

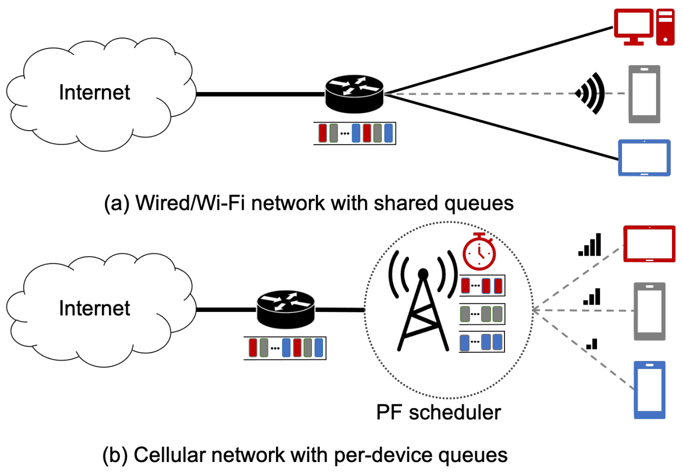

Internet
(a) Wired/Wi-Fi network with shared queues
Internet
PF scheduler
(b) Cellular network with per-device queues

Figure 1: Cellular vs. wired/Wi-Fi networks

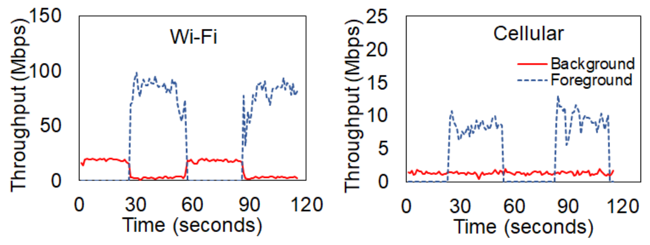
employ schedulers in base stations to provide fairness but the scheduler is agnostic of applications' needs. While the first two are self explanatory, we expand on cellular schedulers.

PF schedulers are normally used for cellular downlinks because they provide a balance between fairness and peak cell throughput [6, 10]. Because the PF scheduler forms the bedrock of most operator' networks, we present a quick background on a general form of PF schedulers [11, 12]. Consider a particular radio cell in a typical cellular network (Figure 1b). There is a separate queue for each client that connects to this cell. The scheduler operates at the granularity of Transmission Time Intervals (TTI), which is typically 1 $millisecond$ in LTE. Further, the available capacity is split into many frequency bands, which consist of sub-carriers and time slots as Physical Resource Blocks (PRB). During each scheduling interval, the scheduler allocates resources in the form of PRBs to client devices. To achieve a good balance between fairness and utilization, the PF scheduler maximizes the ratio of the expected data rate (normally derived from signal measurements) to the moving average of achieved throughput for each client. The winning client gets all PRBs that it needs in the current interval. If the winner does not have enough data to exhaust all the resources, the next winner will get the remaining resources, and so on. So, if all clients have the same signal quality and infinite data to send, scheduling will effectively become round-robin. Also, note that even clients with poor signal quality will eventually get their turn when their signal peaks (i.e., fairness) and that clients with good signal quality achieve high throughput (i.e., utilization).

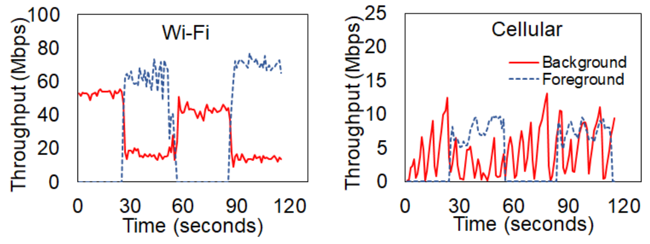
B. *Opportunity study*

LEDBAT [13] and TCP-LP [14] are two popular LPT protocols. The key idea of these background transport protocols is to detect congestion earlier than regular TCP using use one-way packet delays [13, 15]. We performed a simple experiment to study the performance of LEDBAT and TCP-LP in Wi-Fi and cellular networks.

We use a simple Wi-Fi testbed, where two clients connect to an access point. The access point is connected to a server using



(a) LEDBAT



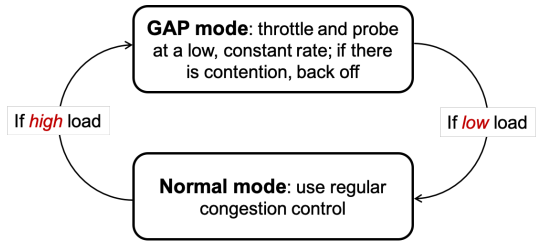
(b) TCP-LP

Figure 2: Behavior of existing protocols

a high speed wired connection. There are two flows in the network, one for each client. The first flow is a persistent long flow that uses LEDBAT, representing the background flow; the second flow uses CUBIC, representing the foreground flow. The foreground traffic is an on-off traffic; the flow joins the system every 60 seconds, transmits for 30 seconds and sleeps for 30 seconds. We repeat the experiment for a cellular network, in which the two clients connect to a base station, instead of a Wi-Fi access point.

Figure 2(a) shows the throughput of foreground flow (using CUBIC) and background flow (using LEDBAT) for Wi-Fi (Figure 2a(left)) and cellular (Figure 2a(right)) testbeds. While LEDBAT shows expected qualitative performance in Wi-Fi by clearly yielding to foreground flow, it performs poorly in cellular (i.e., the background flow does not use any spare capacity when foreground flow is absent). Figure 2(b) shows the results when we use TCP-LP for the background flow. While TCP-LP also performs well in Wi-Fi, it competes with and removes significant capacity from the foreground flow in the cellular network (Figure 2b). Therefore, we conclude that existing LPT protocols achieve sub-optimal performance in cellular networks.

In wired and Wi-Fi networks, delay-based background transport protocols perform well as all traffic passes through the same bottleneck queues and the delay reliably signals the aggregate congestion (queuing) on the path. However, cellular networks use per-device queues and the scheduler *fairly* provides opportunity to *all* clients across scheduling intervals based on radio signal and historical throughput. Therefore, the delay would not increase as steeply as it would in wired and Wi-Fi networks during congestion. Consequently, the measured delay in LEDBAT and TCP-LP does not fully capture the overall congestion of the radio cell and offers limited insight into the cell capacity, which is dynamic. In addition, delay profiles differ across cells impairing the ability to establish accurate parameter settings and base delay estimates.



GAP mode: throttle and probe at a low, constant rate; if there is contention, back off
If *high* load
If *low* load
Normal mode: use regular congestion control

Figure 3: High-level overview

III. Design

We propose *Legilimens* for background applications to utilize spare capacity without significantly affecting other foreground traffic. We provide a high-level overview of *Legilimens* in Figure 3. *Legilimens* estimates capacity and cell load and sends the background traffic to fill the spare channel capacity. This is the *normal* operation mode. *Legilimens* uses AIMD-style congestion control during normal mode.

On the other hand, if the load estimate indicates that there is increased competition, *Legilimens* enters a dormant mode (i.e., GAP mode in Figure 3) during which the flow waits and yields to the foreground traffic. While waiting, *Legilimens* intermittently sends short *bursts* of packets to probe for spare capacity. Because we use regular data packets during probing, there is no bandwidth overhead. Choosing the correct burst size for probing is non-trivial – while large bursts provide more accurate estimates, they degrade the performance of foreground flows at high loads. Therefore, instead of employing a single burst size, we start with the small burst and gradually increase it upon detecting lower load. We call this the *gradually aggressive probing* (GAP) mode. The GAP mode prevents spurious oscillations to normal mode and back, and serves as hysteresis between the modes.

We require three key mechanisms to realize the design from Figure 3. First, we need to estimate capacity, so we can provide room for transient bursts of foreground traffic and to establish a bound on sending rate. Second, we need an agile mechanism to detect the load level (i.e., when there is other traffic). Finally, we need a robust and efficient congestion control scheme to achieve both high performance for all applications and fairness among background applications.

Scheduling mechanisms in LTE are different and independent for uplink and downlink [16], and their optimizations can evolve separately. While we have designed *Legilimens* for downlink *only*, we believe that background-traffic management could be beneficial for uplink, and that our high-level ideas for detecting capacity and load, and using gaps in transmission to yield can be applied to uplink when low-priority effect is desired. However, this requires careful investigation that is beyond the scope of this paper. Also, we constrain our design to not require network- or receiver-side changes to be deployment friendly. Finally, *Legilimens* does *not* make any assumptions about signal quality and load variations.

A. Estimating capacity and busyness

Intuition: Quickly and accurately estimating capacity and the presence of other competing traffic is hard in wired and Wi-

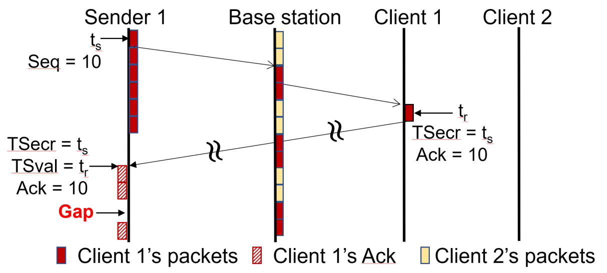


Figure 4: Scheduler and timestamps

Fi networks due to shared FIFO queues. Our **key insight** is that the PF scheduler in cellular networks enables us to accurately detect capacity and competition in *only* a handful of TTIs. For *each* TTI, the scheduler selects a receiver with the highest ratio of its expected peak rate and recent throughput; hence the scheduler is likely to service those receivers that it has not recently serviced. In other words, because fairness is built into PF schedulers (see Section II-A), if there is competition, a receiver will lose its turn eventually and another receiver will gets its turn. When a receiver loses its turn, there will be gaps in the packet arrivals at the receiver, which we exploit to detect competing traffic. Also, the scheduler is likely to allocate *all* resource blocks to a single receiver in a given TTI, provided the receiver's queue has enough data (i.e., if a sender sends a sufficiently large burst of data, it is likely to receive full capacity for at least one TTI). Therefore, (1) we estimate capacity by observing the maximum number of packets that were serviced in each TTI; and (2) we detect competing traffic by analyzing packet inter-arrivals at the receiver. We avoid making changes to client-side software by utilizing TCP timestamp option [17] so that the received timestamps of data packets can be observed via ACKs at the sender. TCP timestamp option is commonly enabled in clients [18, 19].

We use Figure 4 to explain how we can detect gaps in the schedule — this indicates competition or *busyness* — by observing timestamps. In this simple example, two senders send data to two cellular clients that share the base station (e.g., same LTE band); sender 1 sends to client 1 and sender 2 to client 2; we show only sender 1 and client 1. Sender 1 sends a stream of packets, which are queued in the base station before they are scheduled to be sent to client 1. Let us assume that the scheduler picks client 1. Therefore, client 1 is likely to receive a series of back-to-back packets until the scheduler switches to sender 2. Sender 1 observes the TCP options fields TS_{ecr} and TS_{val} of ACKs: TS_{ecr} indicates the timestamp at sender 1 when the packet was sent; TS_{val} indicates the timestamp at client 1 when the packet was received. If there is a "gap" in the schedule, then packets that were sent together (i.e., packets that have continuous or same TS_{ecr} values) will have a discontinuity in TS_{val} values. Our idea of detecting capacity and competition by observing timestamps does *not* make any assumptions about signal quality and load. Our evaluations show that *Legilimens* performs well in a real network and large-scale simulations under representative workloads.

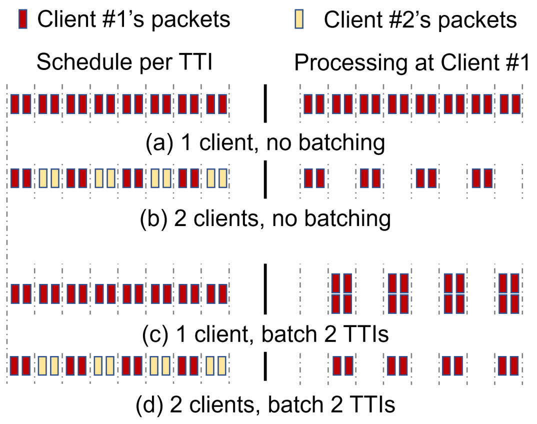

Client #1's packets
Client #2's packets
Schedule per TTI
Processing at Client #1
(a) 1 client, no batching
(b) 2 clients, no batching
(c) 1 client, batch 2 TTIs
(d) 2 clients, batch 2 TTIs

Figure 5: Scheduler with batching

Practical issues: While simply observing received timestamps allows us to detect competition, delayed ACKs [20] and packet batching optimization at the receivers significantly complicate the detection logic. When the receiver combines and generates a cumulative ACK for multiple contiguous packets, the sender would infer an inflated capacity and spuriously detect gaps in the received stream. Figure 5 shows the effect of batching in the received packet stream. Without any batching or delayed ACKs, we clearly see gaps in the received packet stream in Figure 5(b) for the case of 2 senders, whereas there is no gap in Figure 5(a) when there is only one sender. In Figure 5(c), although there is only one sender (to client 1), the client 1 combines two TTIs worth of data in layer-2 before processing in higher layers, and, therefore, there is a gap in the observed timestamps. However, there is no contention and we should not throttle the sender. But, if we look for gaps in the received timestamps to identify competition, we would not distinguish between the cases of two senders without batching (Figure 5(b)), one sender with batching (Figure 5(c)) and two senders with batching (Figure 5(d)). It is hard to guess the batching behavior of receivers. Therefore, we further refine our detection logic in algorithm 1. Instead of *only* looking for gaps in the received stream, we reconstruct the schedule at the sender by observing the timestamps. The algorithm considers both the data volume and packet inter-arrival times to infer capacity and competition.

Capacity and busyness estimation algorithm: Algorithm 1 compares the recently received sequence number and timestamp (i.e., when the corresponding data packet was received at the client) to the previous ACK. If the timestamps match, then we accumulate the number of acknowledged segments for the previous slot (line 22). If the timestamps do not match, *first* we see if there is a gap (i.e., the current timestamp differs from the previous timestamp by *more* than 1 TTI), as in line 8. If there is no gap, we infer that the ACKs are back-to-back and that the base station is *not* busy. If the ACKs are not back-to-back, then we compute the amount of data sent per TTI (i.e., the instantaneous rate in terms of packets per TTI) for the previous slot (line 7). Since the scheduler is expected to allocate full capacity for at least one TTI, we infer capacity by looking at the amount of data sent per TTI and by taking

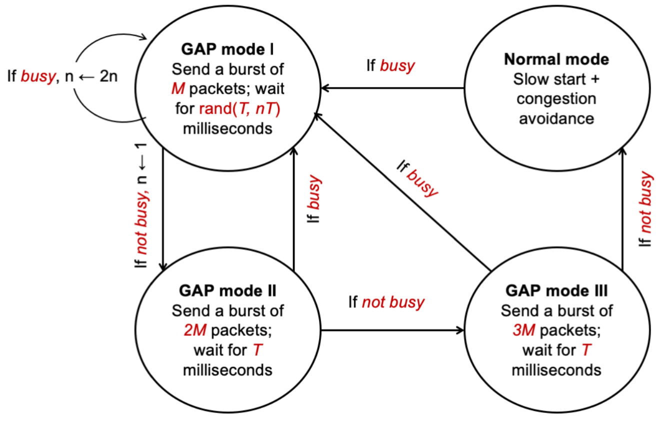
Algorithm 1 Estimate capacity and busyness

```
Input: ACK sequence number ($s$), timestamp ($t$)
Output: Capacity ($capacity$), IsBusy ($busy$)
1: function ESTIMATE($s, t$)
2:   if $t > t_0$ then
3:     $i \leftarrow i + 1$
4:     $slot[i].t \leftarrow t$
5:     $slot[i].n \leftarrow (s - s_0)$
6:     $duration \leftarrow (t - slot[i-1].t)$ / TTI
7:     $rate \leftarrow slot[i-1].n$ / $duration$
8:     if ($duration > 1$) then                ▷ If there is a gap
9:       if ($rate \geq \lambda * capacity$) then
10:        $busy$ ←FALSE
11:      else
12:        $busy$ ←TRUE
13:      end if
14:      if ($rate > capacity$) then
15:        $capacity \leftarrow \frac{4}{5} * capacity + \frac{1}{5} * rate$
16:      end if
17:    else
18:      $busy$ ←FALSE
19:      $capacity \leftarrow \frac{4}{5} * capacity + \frac{1}{5} * rate$
20:    end if
21:  else
22:    $slot[i].n \leftarrow slot[i].n + (s - s_0)$
23:  end if
24:  $t_0 \leftarrow t$, $s_0 \leftarrow s$
25: end function
```

the maximum value (line 15). If there is no gap, we update capacity unconditionally so that the capacity can increase or decrease based on channel conditions (line 19). We detect busyness using a simple heuristic: if the instantaneous rate is more than the capacity by a factor λ, then we infer that the base station is *not* busy. Intuitively, if there is no other traffic, we expect instantaneous rate to be very close to capacity. If there are other senders or if there are not enough packets at the base station to consume all the resource blocks for one TTI, the instantaneous rate would be less than capacity. Therefore, λ lies between 0 and 1. Setting $\lambda = 0$ would *always* allow *Legilimens* senders to send in the normal mode (i.e., *Legilimens* would never yield). While $\lambda = 1$ would allow *Legilimens* senders to send traffic in the normal mode *only* when there are no other senders, it would be sub-optimal if there are other senders but they do not have enough data to saturate the capacity; we would want *Legilimens* to use spare capacity in this case. Therefore, we chose $\lambda = 0.5$ (Table I) in our experiments, which achieves a good trade-off between foreground and background performance. Finally, we employ *exponential* averaging with a weight of $\frac{1}{5}$ to the newly inferred capacity to filter outliers (line 15 and line 19);

B. Congestion control

Congestion window (cwnd) adaptation: We show *Legilimens*'s complete state machine in Figure 6. *Legilimens* relies



If busy, n ← 2n
GAP mode I
Send a burst of M packets; wait for rand(T, nT) milliseconds
If busy
Normal mode
Slow start + congestion avoidance
If not busy, n ← 1
If busy
If busy
If not busy
GAP mode II
Send a burst of 2M packets; wait for T milliseconds
If not busy
GAP mode III
Send a burst of 3M packets; wait for T milliseconds

Figure 6: *Legilimens* congestion control

Table I: Parameter values

Parameter	Description	Value
λ	*Busy threshold*	0.5
M	*Probe burst size*	50 KB
T	*Time between* GAP *modes*	250 ms

on estimated capacity and busy signals (section III-A) to switch between sending (i.e., *normal mode*) and probing (i.e., GAP mode). Because normal mode aggressively sends data, a series of three probe modes is used (i.e., GAP modes I, II, and III), each with a larger burst size than the previous, to increase the confidence in our capacity estimates before entering *normal mode*. During each GAP mode, we send a burst of multiples of M packets and wait for T milliseconds to estimate capacity and busyness by observing ACKs using algorithm 1. If the base station is not busy, we enter the next GAP mode, in which we increase the burst size and wait for ACKs. After the third GAP mode (i.e., GAP III), we enter normal mode if no busyness is detected. If, at any point, we sense the presence of other senders, we revert back to the GAP mode I. If busyness is sensed in GAP mode I, then exponential random back-off is applied in multiples of T.

During *normal mode*, *Legilimens* uses AIMD-style congestion control. *Legilimens*'s behavior soon after entering normal node is similar to that of loss-based TCP variants after a timeout — the protocol starts with a small window but quickly ramps up the rate using slow start until a threshold and adjusts the rate more gradually using congestion avoidance thereafter.

Fairness: *Legilimens*'s primary objective is to satisfy demands of foreground applications and provide fairness between competing background flows. To this end, if any *Legilimens* sender estimates that there are other competing senders, background or foreground, the sender exponentially backs off (i.e., doubles the waiting interval, T). This back-off can happen anytime during the probing burst as the sender estimates busyness on each ACK. Our mechanism is somewhat similar to CSMA in IEEE 802.11, and, therefore, we achieve fairness between background traffic in a similar way. We study fairness in our evaluation in Section VI-D. Table I lists our design parameters along with their default values; we analyze their sensitivity in Section VI-E.

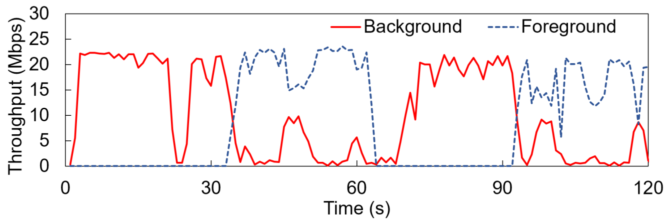


Figure 7: *Legilimens* vs. CUBIC in the real network

Summary and discussion: In cellular networks, packet delay can increase either due to (1) degradation in signal quality or (2) contention at the base station. Existing LPT protocols do not correctly identify the root cause of delay increase but respond by either completely backing off or not at all in both cases (see Figure 2). *Legilimens*'s capacity and busyness estimation algorithm, leveraging the nature of PF schedulers, deconstructs the schedule by observing timestamps and isolates the two cases. For (1), *Legilimens* correctly reduces the rate but does not completely back off. For (2), *Legilimens* backs off, enters GAP mode and waits for the base station to become free. Thus, differentiating between the two cases is key to *Legilimens*'s performance gains over existing LPT protocols.

C. Deployment model

Because *Legilimens* is designed only for cellular downlinks, it should be deployed only on servers providing data to cellular devices. A potential path for adoption could be deploying *Legilimens* on cellular proxy servers and TCP splitters that most cellular operators employ today [21], or on CDN edge nodes that specifically peer with cellular gateways or are placed inside cellular networks (e.g., mobile edge cloud) — all of these serve traffic to *only* mobile devices over cellular links. Further, network and CDN operators can utilize existing traffic classification techniques to identify background traffic and direct its delivery via servers that run *Legilimens*. This would significantly reduce the number of servers that need to support *Legilimens*, while still allowing cellular networks to enjoy its benefits. For general server and CDN deployments, servers can use mapping techniques between IP addresses and known cellular gateways, or even HTTP headers that indicate that clients are behind cellular gateways or proxies.

IV. Evaluation overview

We use *three* avenues to analyze *Legilimens's* performance (latency, bandwidth, fairness) under realistic conditions (varying signal quality, flow sizes, load) as well as understanding its behavior (queuing, sensitivity to design parameters): a real implementation on a major U.S. carrier network during idle and busy hours (not isolated from user traffic), *PhantomNet* (isolated from user traffic), and large-scale ns-3 simulations.

V. Evaluation using a real implementation

A. Implementation

We implemented *Legilimens* on a Linux server (kernel version 4.4). Our implementation takes about 300 lines of additional code in the Linux kernel's modular congestion control framework. The framework eases the implementation of new congestion control algorithms by providing hooks in the kernel for taking actions whenever there is a congestion control event (e.g., ACKS, timeouts). Our design requires TCP timestamps [17] to be enabled.

B. Testbeds

Real network testbed: Our real testbed consists of 6 Samsung Galaxy smartphones, all running Android 7.0. The devices are band-locked to use the same LTE carrier (i.e., they share the same bottleneck radio link). For the stationary experiments, test devices are located on the second story of a 3-story concrete building with wall-to-wall windows. The test devices are connected to the standard macro cell using a 10 MHz carrier. The devices register Reference Signal Received Power (RSRP) between -92 and -95 dBm and Reference Signal Received Quality (RSRQ) between -10 to -13 dB.

PhantomNet testbed: Our PhantomNet testbed emulates a cellular network using a server running open air interface (OAI) inside the LTE packet core and eNodeBs (base stations). eNodeB is based on Software-Defined Radio (SDR) running OAI (Intel NUC + USRP B210). PhantomNet provides Nexus 5 phones accessible via Android Debug Bridge (ADB), target server and a GUI interface. Log-distance path loss model is used to produce realistic varying Signal to Noise Ratio (SNR). We use remotely connected servers for generating test traffic.

C. Testing methodology

Because real network conditions are dynamic (especially during busy hours), we run our experiments three times to get sufficient confidence. Each experiment typically lasts for 4–8 hours and transfers about 100 GB of data.

D. Behavior as a low-priority protocol

Recall from Section II-B that LEDBAT and TCP-LP do not work as intended in cellular networks. We perform a similar experiment in the real network and in PhantomNet. We create a simple workload that generates one background and one foreground flow. The background flow is an always-on, long flow that uses *Legilimens*. The foreground flow uses CUBIC (our baseline) and sends data in an on-off pattern. We study the per-second throughput of foreground and background flows over time to see whether *Legilimens* (background) yields to foreground flows.

Figure 7 shows how *Legilimens* shares the cellular link with CUBIC. Unlike LEDBAT and TCP-LP (Figure 2), *Legilimens yields to* CUBIC, *backs off completely in* GAP *mode (to allow scheduling opportunity to other flows), and efficiently recaptures spare capacity when available.* We also performed the same experiment on PhantomNet and the results are similar (PhantomNet results not shown). This study confirms that *Legilimens* functions correctly as intended in cellular networks.

E. Performance with realistic workloads

Next, we would like to quantify the gains of using *Legilimens* for large background transfers. Our workload for this study consists of a *varying* number of foreground flows and background flows. We model our workload as heavy tailed with a small number of large flows generating the vast majority

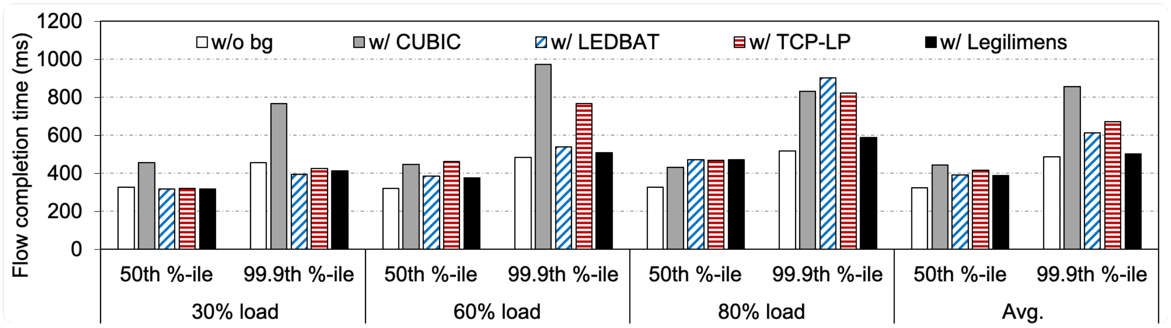
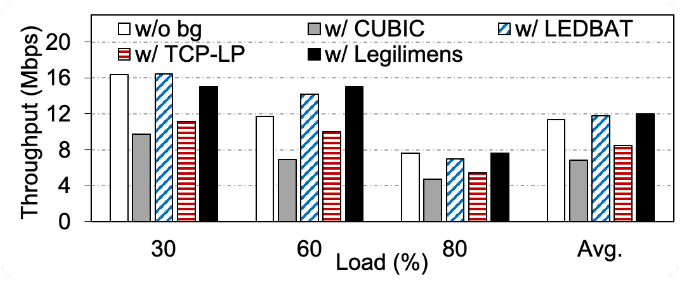
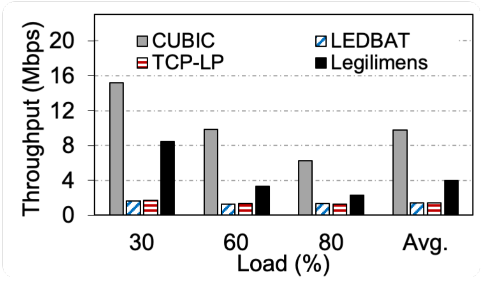


Figure 8: Performance (FCT) of short foreground flows in a real network under realistic workload



(a) Performance (throughput) of long foreground flows



(b) Performance (throughput) of background flows

Figure 9: Performance of long flows (foreground and background flows) results under realistic workload

of bytes [22, 23]. Specifically, we model foreground traffic as a mix of short, medium, and long flows: (1) 64 KB *short* flows represent web objects and mobile app communication, (2) 1 MB *medium* flows represent transfers such as video (e.g., chunk size in adaptive streaming), and (3) 32 MB *long* flows represent app updates and cloud data sync. The short, medium and long flows contribute to 10%, 30%, and 60%, respectively, to overall (foreground) load. The background flows are long and remain *always on*.

Controlling the load in a real (operational network): While we have access to measuring the resource utilization (i.e., PRB utilization) of a cell, generating traffic at different load levels is a challenge in a real (operational) network because of other (non-experimental) traffic. Specifically, we want to evaluate at three load levels: (1) PRB utilization of 30% (low load), (2) PRB utilization of 60% (medium load), and (3) PRB utilization of 80% (high load). One way to precisely control the load is to run our experiments during idle hours. We analyzed reports from the cellular infrastructure and we found that the typical quiet hours ($utilization < 10\%$) are between midnight and 6 AM. Therefore, we run our experiments during these hours. Note that while we calibrate the average cell load to be around our desired levels, instantaneous load constantly varies due to the dynamics created by signal variation and traffic fluctuation, which is typical in real networks.

Because *Legilimens* is for background flows only, we fix the protocol for foreground flows to be CUBIC and we change the protocol for background flows as CUBIC, LEDBAT, TCP-LP, and *Legilimens*. In addition to these, we also run without any background traffic ("*w/o bg*"), which is *ideal* for foreground flows. We evaluate the protocols on our workload based on the following metrics:

- Median and tail (e.g., 99.9^{th} percentile) flow completion times (FCT) of short flows
- Throughput of medium and long foreground flows
- Throughput of background flows

Figure 8 shows the median and 99.9^{th} percentile FCT of short flows in which the foreground traffic (CUBIC) shares capacity with each background protocol; Figure 9(a) and Figure 9(b) show the throughput of foreground long and background (long) flows, respectively; we do not show the throughput of medium flows for brevity. The figures show performance at the three load levels as well as their average performance across loads. As load increases, all the schemes suffer longer flow completions and low throughput due to contention. From the figures, we observe that *Legilimens* does *not* interfere with foreground flows—when background uses *Legilimens*, the foreground flows achieve performance close to *w/o bg* (ideal). Being a fair-share protocol, it is not surprising to see that CUBIC achieves the lowest performance for foreground flows and achieves the highest performance for background flows. This is clearly not desirable. LPT protocols (LEDBAT and TCP-LP) achieve good foreground performance but suffer in background performance. For some loads (e.g., 60%), we see that *Legilimens* is even better than "*w/o bg*", which is clearly an experimental artifact as we cannot precisely control the load in a real network. Nevertheless, such variations are negligible in scale when compared to *Legilimens*'s gains over other LPT protocols. *Legilimens achieves 2.8x higher throughput on average (up to 5.2x) over existing* LPT *protocols (*LEDBAT *and* TCP-LP*) while enabling foreground flows to achieve similar or better throughput (better on average).*

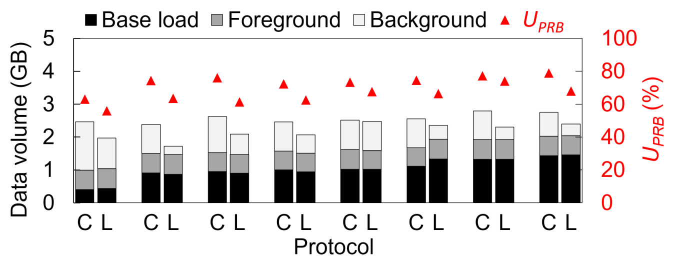


Figure 10: Cell load and utilization during busy hours

We repeated the same experiment on PhantomNet and observed similar results (not shown). In PhantomNet, *Legilimens* achieves 2x higher throughput on average (up to 3.4x) over existing LPT protocols while also enabling foreground flows to achieve slightly better performance.

F. Performance during busy hours

To study the behavior of *Legilimens* during busy hours, we conduct test runs over 3 days between 13:00 and 17:00 hours, when our carrier network is loaded with user traffic, over which we have no control. We run the workload from Section V-E. We supply foreground traffic in lower volume, at an overall rate of about 5 *Mbps*. We record the total downlink data volume and PRB (U_{PRB}) measurements from the network. In each hour, we have four 15-minute timebins, each containing one of these loads: (1) base load only, (2) foreground traffic is added, (3) background flow using CUBIC is additionally added, and (4) *Legilimens* is used for a background flow. The first two cases are merely to confirm the volume of base load and foreground traffic, while we study the cases with background traffic in Figure 10. We select 15-minute bins with similar base load to compare the total data volume of CUBIC (C) and *Legilimens* (L) to their respective U_{PRB}. We discard the timebins with base load outliers because the load fluctuation was significant in those bins. Figure 10 shows the pairs of CUBIC/*Legilimens* timebins and their data volumes as stacked bars, sorted by increasing base load (bottom bar sections). We can see that *Legilimens* generates less traffic (top bar sections) and results in lower PRB utilization (U_{PRB}) than CUBIC, across the range of base loads during busy hours. While CUBIC generates the same amount of background load regardless of the base load, *Legilimens* modulates its background throughput favorably depending on base load. *On average, Legilimens results in* 13.8% *lower* PRB *utilization than* CUBIC *during busy hours, which is desirable.*

G. Operation in a mobile scenario

To further evaluate *Legilimens* under fast-varying load and radio signal over wide ranges, we conduct a mobile test. The test lasts for 37 minutes over 24 km distance. It starts with walking for about 8 minutes, then driving at moderate speed (30-70 km/h) for about 3 minutes, followed by freeway speed (70-110 km/h) for 10 minutes, and complets the remainder of the test at moderate driving speed. Two J7 phones are used, placed in a laptop bag, which is carried and then placed on the vehicle seat. One phone receives a continuous *Legilimens* background flow, and another one receives a foreground CUBIC flow with one-minute ON/OFF pattern.

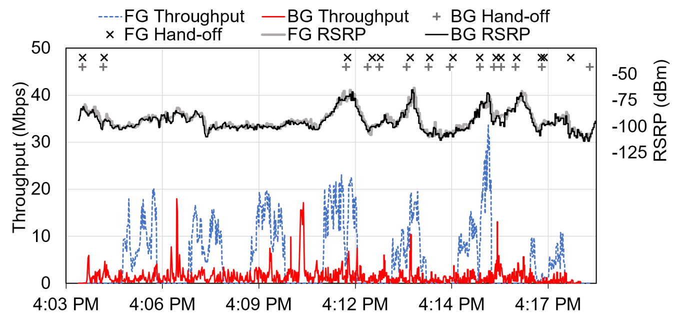


Figure 11: Throughput of *Legilimens* (background) vs. CUBIC (foreground) while moving

Figure 11 shows the first 14 minutes of the test. Signal strengths, measured as RSRP, varied from -111 to -68 dBm. A total of 13 hand-offs occurred. While we do not expect to see a clear pattern of yielding by *Legilimens* because we only see two flows among a large volume of regular traffic in the network, *we clearly see that Legilimens is significantly more conservative than* CUBIC, which competes with other traffic. *Legilimens* also does not appear to suffer major disruption by changing signal strength, especially in the second half of the plot, where driving causes frequent hand-offs.

VI. Evaluation using simulations

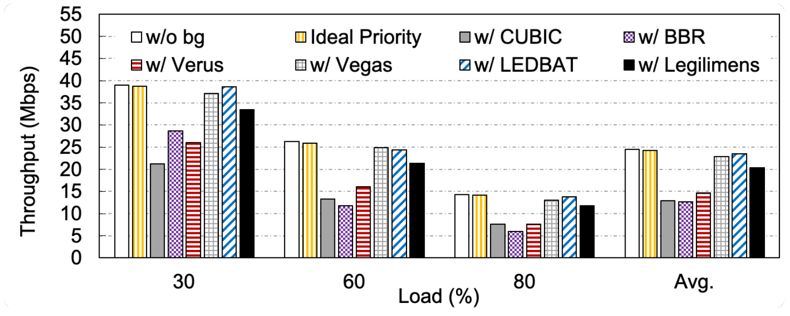
We rely on simulations to analyze *Legilimens* at large scales (hundreds of devices and several flows per device), and to compare to a larger set of protocols. We also rely on simulations to further analyze queuing behavior, fairness among background flows, and parameter sensitivity.

A. Methodology

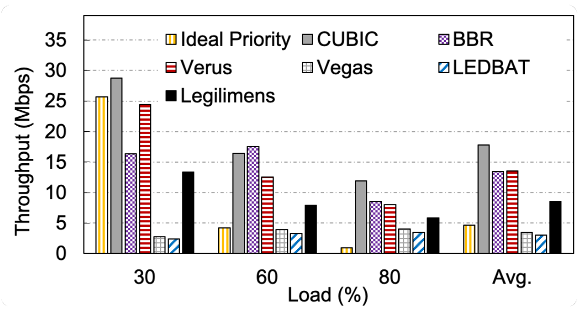
We use ns-3 v3.27 simulator with log-distance propagation model for the radio signal. The radio channel is configured with a single carrier with 10 MHz bandwidth and closed-loop MIMO. Network topology consists of multi-hop wired links that connect remote servers to Packet Data Network Gateway (PGW) using 1 Gbps with 10 ms delay links. The link speed between the base station and the SGW/PGW is also 1 Gbps. The cellular uplink and downlink are bottlenecks in our topology. We run our experiments with carrier aggregation.

B. At-scale performance vs. other protocols

In this study, we simulate a workload similar to the one we used in Section V-E and run it at 30%, 60%, and 80% load. For scale, we simulate a substantially larger client base with 100 devices and a large number of foreground and background flows to each device to exercise the impact of the PF scheduler behavior under variety of RF signal characteristics. The key difference between the real testbed and simulations is the scale (i.e., the scheduler in our simulated network must serve flows from 100 queues as opposed to 4 in our real testbed). The 100 devices are randomly placed in a cell and they generate a mix of foreground and background flows. We run each 15-minute test 3 times with different random seeds for client selection.



(a) Throughput of long foreground flows



(b) Throughput of background flows

Figure 12: Simulation results for realistic workload

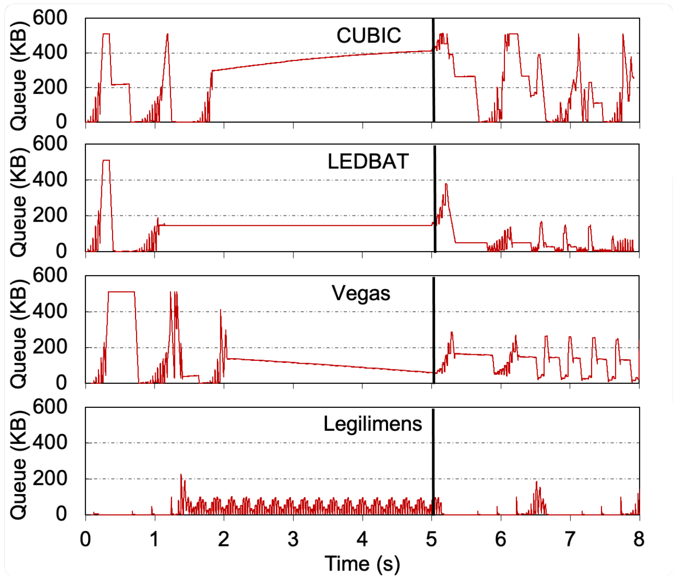


Figure 13: Queuing behavior

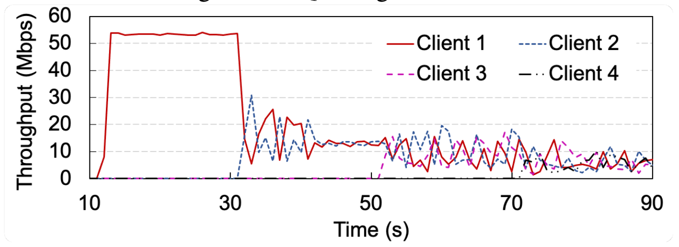


Figure 14: Fairness analysis among *Legilimens* flows

Because LEDBAT and TCP-LP perform similarly, we omit TCP-LP in our simulation study. Instead, we compare to VEGAS [24], and more recent BBR [25] and VERUS [26]; we use their open-source ns-3 implementations [27, 28]. We set the design parameters as recommended in the original papers and we verified the implementations by reproducing the bottom-line results from their papers. In addition, we compare to an *ideal* scheduler that strictly prioritizes foreground over background traffic but performs PF scheduling within each class. While such a scheduler is not available in practice, it serves as a good comparison point.

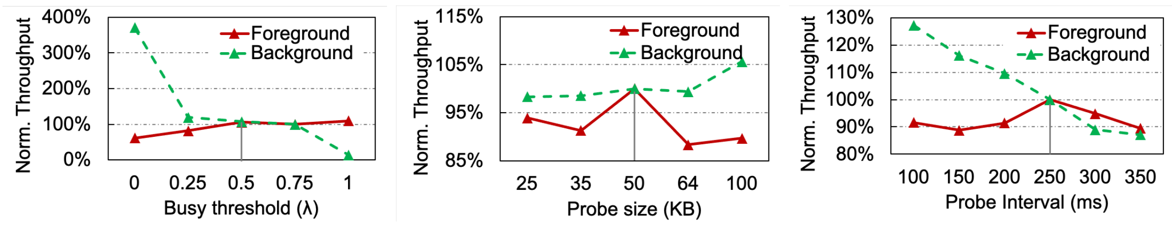
Figure 12(a) and Figure 12(b) show the throughput achieved by foreground and background flows in our simulation study. We do not show numbers for short and medium foreground flows because the trends are similar. The figures show performance at the three load levels as well as their average performance across loads. As load increases, the throughputs of all the schemes decrease due to contention. The trend, however, is similar to Figure 9. Fair share protocols such as CUBIC, BBR, and VERUS do *not* prioritize foreground flows and suffer from poor foreground throughput. Delay-based protocols that back off when delays increase, such as VEGAS and LEDBAT, achieve good foreground throughput but are not able to exploit spare capacity to send background data. *Legilimens* performs within 12% (on average) of the ideal schemes (i.e., "w/o-bg" and "Ideal Priority") in foreground throughput. Interestingly, *Legilimens* achieves about 1.84x higher background throughput than "Ideal Priority". Because "Ideal Priority" causes starvation of low priority background flows, the flows suffer several timeouts and their throughput suffers. In contrast, *Legilimens* uses the GAP mode to pause and resume transmissions much more quickly. *Overall, Legilimens achieves 2.2x higher throughput on average (up to 5.8x) over existing* LPT *protocols (including* VEGAS*) while enabling foreground flows to achieve similar (within 10%) throughput.*

C. Queuing behavior

To further understand protocol behavior, we analyze their queue lengths. Figure 13 shows the queue lengths of CUBIC, LEDBAT, VEGAS, and *Legilimens*. Each background flow starts alone and a foreground CUBIC flow joins at 5 second mark, as indicated by vertical lines. While CUBIC, LEDBAT, and VEGAS compete with foreground flows for scheduling opportunity and exhibit higher queue lengths, *Legilimens* keeps nearly empty queues for long periods. Thus, *Legilimens provides significantly more opportunity for foreground traffic.*

D. Fairness

We study fairness among background flows by initiating multiple long background flows to the same device and observing their time-averaged throughput. For this experiment, we start 4 *Legilimens* flows in a staggered manner every 20 seconds. Since any single *Legilimens* flow considers all other traffic as foreground, even if other background flows are present, it has to transition between operation modes. Figure 14 shows that *Legilimens* flows adapt their sending rate with increasing competition and converge to fair shares. The Jain's Fairness Index for the periods of 2, 3, and 4 concurrent flows are 0.999, 0.993, and 0.996, respectively,



(a) Sensitivity to λ (b) Sensitivity to probe size (M) (c) Sensitivity to probe interval (T)

Figure 15: Sensitivity analysis

indicating high fairness. We have also tested fairness of 8 and 24 concurrent *Legilimens* flows, and they resulted in fairness indices of 0.992 and 0.999, respectively. *From this experiment, we conclude that Legilimens achieves near-perfect fairness among background flows.*

E. Sensitivity to design parameters

We study *Legilimens*'s performance sensitivity to design parameters (Table I) using the same workload from Section VI-B. Figure 15 shows the sensitivity in terms of throughput of background and foreground flows, normalized to their throughput with default values of parameters used in *Legilimens*.

Figure 15(a) shows the sensitivity as busy threshold (λ) varies from 0 to 1 while the other two parameters are set to their default values. At $\lambda = 0$, *Legilimens* behaves like a fair-share protocol and foreground performance suffers; at $\lambda = 1$, background flows starve. There is a wide *stable* range between 0.25 and 0.75 where both foreground and background flows achieve good throughput, and we pick a default value of 0.5.

Figure 15(b) analyzes probe size (M) impact, as it varies from 25 KB to 100 KB. Smaller probes are inaccurate and very large probes impose overhead — both impact foreground throughput. However, between 25 KB and 100 KB, variations are within 20%. We pick a probe size of 50 KB as default.

Figure 15(c) shows impact of probe interval (T). Probing too frequently (smaller values) or infrequently adversely affects foreground performance, whereas background flows do better at lower frequency, which gives them more time to send data. *Legilimens* achieves stable, good performance for both foreground and background between 200 ms and 300 ms. We pick an interval of 250 ms as default.

VII. Related work

Legilimens is related to fair-share and LPT protocols as well as to capacity estimation. Fair-share protocols can be categorized as loss-based and delay-based. RENO [29], NEWRENO [30], TAHOE [31]), and CUBIC [32] are some of the well-known loss-based TCP variants. Most of the conventional, loss-based TCP variants, incur high queuing delays due to "*buffer bloat*" in cellular networks [33]. AQM-based protocols [34–38] require support in routers and are hard to implement in practice. Delay-based protocols (e.g., VEGAS [24]) do not suffer from buffer bloat as much as loss-based protocols but they do not work well with cellular networks' variable delays. Although LPT protocols (e.g., TCP-LP [14], LEDBAT [39], NICE [40]) perform well in wired and Wi-Fi networks, they do poorly in cellular networks [4, 41–44]. We have extensively discussed TCP-LP [14] and LEDBAT [39].

SPROUT [12] and VERUS [26] explicitly consider the time-varying capacity of cellular links. BBR [45], PropRate [46], and ExLL [47] use *both* bandwidth estimation and round trip times/one-way delays to maximize throughput while minimizing delay. While these *fair-share* protocols improve performance in cellular networks, their underlying goal is to equally share the bottleneck capacity. In contrast, *Legilimens* is a LPT with a goal of efficiently utilizing *spare* capacity without affecting foreground flows. Further, we use novel mechanisms to achieve our goal. Our evaluations show that VERUS and BBR do not perform well as LPT. Loadsense [48] schedules background traffic based on *passive* estimation of cellular load by observing the power of channel and pilot signal. However, passive estimation requires support at clients. In contrast, *Legilimens* performs active measurements and does not require support at clients.

There is a large body of work on capacity estimation. While several of these ideas [49–53] are applicable to a broader class of networks, our algorithm optimizes for PF schedulers in cellular downlinks, which enables us to be simple and efficient (e.g., we use data packets for probing, so there is no overhead). QProbe [54] leverages the scheduler to estimate congestion at the base station. However, QProbe uses the estimation to identify bottleneck links, whereas *Legilimens* uses the estimation to schedule background traffic. To the best of our knowledge, none of the existing papers estimate *busyness*, which is central to *Legilimens*'s goal of prioritizing foreground flows over background flows.

VIII. Conclusion

We propose *Legilimens*, an agile transport protocol for background traffic in cellular networks. *Legilimens* employs a *novel* algorithm to quickly and accurately estimate capacity and busyness, using them to achieve two conflicting objectives — quickly yield to foreground traffic and efficiently capture spare capacity. This makes *Legilimens* superior to existing LPT protocols, which achieve one of the objectives but not both. As background applications become more reliant on cellular networks, schemes such as *Legilimens* are needed to enforce high-level objectives that accommodate the needs of users (foreground applications), content providers (background applications), and network operators (resource utilization). We plan to explore techniques to optimize *Legilimens* for uplink cellular traffic and evaluate its performance in wired and Wi-Fi networks in the future.

References

[1] C. E. Andrade, S. D. Byers, V. Gopalakrishnan, E. Halepovic, M. Majmundar, D. J. Poole, L. K. Tran, and C. T. Volinsky, "Managing massive Firmware-Over-The-Air updates for connected cars in cellular networks," in *Proceedings of the 2nd ACM International Workshop on Smart, Autonomous, and Connected Vehicular Systems and Services*, ser. CarSys '17, 2017, pp. 65–72.

[2] S. Zhou, M. U. Chaudhry, V. Gopalakrishnan, E. Halepovic, B. Vamanan, and H. Seferoglu, "Managing Background Traffic in Cellular Networks," in *Proceedings of the International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2019.

[3] Y. Zhang, Å. Arvidsson, M. Siekkinen, and G. Urvoy-Keller, "Understanding HTTP flow rates in cellular networks," in *2014 IFIP Networking Conference*, June 2014, pp. 1–8.

[4] F. Lu, H. Du, A. Jain, G. M. Voelker, A. C. Snoeren, and A. Terzis, "CQIC: Revisiting cross-layer congestion control for cellular networks," in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, 2015, pp. 45–50.

[5] Y. Huang, S. Li, Y. T. Hou, and W. Lou, "GPF: A GPU-based design to achieve˜ 100 μs scheduling for 5G NR," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 207–222.

[6] R. Margolies, A. Sridharan, V. Aggarwal, R. Jana, N. Shankaranarayanan, V. A. Vaishampayan, and G. Zussman, "Exploiting mobility in proportional fair cellular scheduling: Measurements and algorithms," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 355–367, 2014.

[7] H. Holma and A. Toskala, *WCDMA for UMTS: Radio access for third generation mobile communications*. John Wiley & Sons, 2005.

[8] A. Banerjee, J. Cho, E. Eide, J. Duerig, B. Nguyen, R. Ricci, J. Van der Merwe, K. Webb, and G. Wong, "PhantomNet: Research infrastructure for mobile networking, cloud computing and software-defined networking," *GetMobile: Mobile Computing and Communications*, vol. 19, no. 2, pp. 28–33, 2015.

[9] "NS-3 network simulator," http://www.nsnam.org/.

[10] "LTE eNodeB scheduler and different scheduler type," http://www.techplayon.com/lte-enodeb-scheduler-and-different-scheduler-type, 2018.

[11] R. Srikant and L. Ying, *Communication networks: an optimization, control, and stochastic networks perspective*. Cambridge University Press, 2013.

[12] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic forecasts achieve high throughput and low delay over cellular networks," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'13. USENIX Association, 2013, pp. 459–472.

[13] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, "LEDBAT: The new bittorrent congestion control protocol." in *ICCCN*, 2010, pp. 1–6.

[14] A. Kuzmanovic and E. W. Knightly, "TCP-LP: A distributed algorithm for low priority data transfer," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 3, 2003, pp. 1691–1701.

[15] G. Carofiglio, L. Muscariello, D. Rossi, and C. Testa, "A hands-on assessment of transport protocols with lower than best effort priority," in *Local Computer Networks (LCN)*, 2010, pp. 8–15.

[16] N. Abu Ali, A.-E. Taha, M. Salah, and H. Hassanein, "Uplink scheduling in LTE and LTE-Advanced: Tutorial, survey and evaluation framework," *Communications Surveys & Tutorials, IEEE*, vol. 16, pp. 1239–1265, 01 2014.

[17] V. Jacobson, B. Braden, and D. Borman, "TCP extensions for high performance," Internet Requests for Comments, RFC Editor, RFC 1323, May 1992. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1323.txt

[18] D. Murray and T. Koziniec, "The state of enterprise network traffic in 2012," in *2012 18th Asia-Pacific Conference on Communications (APCC)*, 2012, pp. 179–184.

[19] E. Halepovic, J. Pang, and O. Spatscheck, "Can you GET me now?: Estimating the time-to-first-byte of HTTP transactions with passive measurements," in *Proceedings of the 2012 Internet Measurement Conference*, ser. IMC '12. ACM, 2012, pp. 115–122. [Online]. Available: http://doi.acm.org/10.1145/2398776.2398789

[20] R. Braden, "Requirements for internet hosts - communication layers," Internet Requests for Comments, RFC Editor, STD 3, October 1989. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1122.txt

[21] X. Xu, Y. Jiang, T. Flach, E. Katz-Bassett, D. R. Choffnes, and R. Govindan, "Investigating transparent web proxies in cellular networks," in *PAM*, 2015.

[22] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck, "An in-depth study of LTE: Effect of network protocol and application behavior on performance," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13, 2013, pp. 363–374.

[23] B. Vamanan, H. B. Sohail, J. Hasan, and T. Vijaykumar, "Timetrader: Exploiting latency tail to save datacenter energy for online search," in *Proceedings of the 48th international symposium on microarchitecture*, 2015, pp. 585–597.

[24] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, 1995.

[25] M. Claypool, J. W. Chung, and F. Li, "BBR: An implementation of bottleneck bandwidth and round-trip time congestion control for ns-3," in *Proceedings of the 10th Workshop on NS-3*, ser. WNS3 '18. ACM, 2018, pp. 1–8. [Online]. Available: http://doi.acm.org/10.1145/3199902.3199903

[26] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg, "Adaptive congestion control for unpredictable cellular networks," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. ACM, 2015, pp. 509–522. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787498

[27] M. Claypool, "An implementation of bottleneck bandwidth and round-trip time congestion control for ns-3," https://github.com/mark-claypool/bbr.

[28] "Verus-ns3," 2020. [Online]. Available: https://github.com/SoonyangZhang/verus-ns3

[29] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno and SACK TCP," *SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 3, pp. 5–21, Jul. 1996. [Online]. Available: http://doi.acm.org/10.1145/235160.235162

[30] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, "The NewReno modification to TCP's fast recovery algorithm," Internet Requests for Comments, RFC Editor, RFC 6582, April 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6582.txt

[31] V. Jacobson, "Congestion avoidance and control," in *Symposium Proceedings on Communications Architectures and Protocols*, ser. SIGCOMM '88. ACM, 1988, pp. 314–329. [Online]. Available: http://doi.acm.org/10.1145/52324.52356

[32] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger, "CUBIC for fast long-distance networks," Internet Requests for Comments, RFC Editor, RFC 8312, February 2018.

[33] H. Jiang, Y. Wang, K. Lee, and I. Rhee, "Tackling bufferbloat in 3G/4G networks," in *Proceedings of the 2012 Internet Measurement Conference*, ser. IMC '12. ACM, 2012, pp. 329–342. [Online]. Available: http://doi.acm.org/10.1145/2398776.2398810

[34] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM 2010 conference*, ser. SIGCOMM '10. ACM, 2010, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851192

[35] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-Aware Datacenter Tcp (D2TCP)," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12. Association for Computing Machinery, 2012, p. 115–126. [Online]. Available: https://doi.org/10.1145/2342356.2342388

[36] H. Rezaei, M. Malekpourshahraki, and B. Vamanan, "Slytherin: Dynamic, Network-assisted Prioritization of Tail Packets in Datacenter Networks," in *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*, Jul 2018, pp. 1–9.

[37] H. Almasi, H. Rezaei, M. U. Chaudhry, and B. Vamanan, "Pulser: Fast Congestion Response using Explicit Incast Notifications for Datacenter Networks," in *Proceedings of the International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2019.

[38] J. Xue, M. U. Chaudhry, B. Vamanan, T. N. Vijaykumar, and M. Thottethodi, "Dart: Divide and Specialize for Fast Response to Congestion in RDMA-Based Datacenter Networks," *IEEE/ACM Transactions on Networking*, vol. 28, no. 1, pp. 322–335, 2020.

[39] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low extra delay background transport (LEDBAT)," Internet Requests for Comments, RFC Editor, RFC 6817, December 2012. [Online].

Available: http://www.rfc-editor.org/rfc/rfc6817.txt

[40] A. Venkataramani, R. Kokku, and M. Dahlin, "TCP Nice: A mechanism for background transfers," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 329–343, 2002.

[41] J. Wang, A. Huang, WeiWang, Z. Zhang, and V. K. N. Lau, "On the transmission opportunity and TCP throughput in cognitive radio networks," *Int. J. Commun. Syst.*, vol. 27, no. 2, pp. 303–321, May 2012.

[42] A. Gurtov and R. Ludwig, "Responding to spurious timeouts in TCP," in *Proc. of IEEE INFOCOM*, vol. 3, 2003, pp. 2312–2322.

[43] R. Ludwig and R. Katz, "The Eifel algorithm: Making TCP robust against spurious retransmissions," *ACM Computer Communication Review*, vol. 30, pp. 30–36, January 2000.

[44] X. Liu, A. Sridharan, S. Machiraju, M. Seshadri, and H. Zang, "Experiences in a 3G network: Interplay between the wireless channel and applications," in *MOBICOM*, 2008, pp. 211–222.

[45] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Queue*, vol. 14, no. 5, p. 50, 2016.

[46] W. K. Leong, Z. Wang, and B. Leong, "TCP congestion control beyond bandwidth-delay product for mobile cellular networks," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '17. ACM, 2017, pp. 167–179. [Online]. Available: http://doi.acm.org/10.1145/3143361.3143378

[47] S. Park, J. Lee, J. Kim, J. Lee, S. Ha, and K. Lee, "ExLL: An extremely low-latency congestion control for mobile cellular networks," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '18. ACM, 2018, pp. 307–319. [Online]. Available: http://doi.acm.org/10.1145/3281411.3281430

[48] A. Chakraborty, V. Navda, V. N. Padmanabhan, and R. Ramjee, "Coordinating cellular background transfers using Loadsense," in *Proceedings of the 19th annual international conference on Mobile computing & networking*, 2013, pp. 63–74.

[49] S. Keshav, "Congestion control in computer networks," Ph.D. dissertation, EECS Department, University of California, Berkeley, Sep 1991. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1991/6386.html

[50] A. Morton and S. V. den Berghe, "Framework for metric composition," Internet Requests for Comments, RFC Editor, RFC 5835, April 2010.

[51] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy, "Bandwidth estimation: metrics, measurement techniques, and tools," *IEEE network*, vol. 17, no. 6, pp. 27–35, 2003.

[52] C. Dovrolis, P. Ramanathan, and D. Moore, "Packet-dispersion techniques and a capacity-estimation methodology," *IEEE/ACM Transactions on Networking*, vol. 12, no. 6, pp. 963–977, Dec 2004.

[53] N. Hu, L. E. Li, Z. M. Mao, P. Steenkiste, and J. Wang, "Locating Internet Bottlenecks: Algorithms, Measurements, and Implications," in *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '04. ACM, 2004, pp. 41–54. [Online]. Available: http://doi.acm.org/10.1145/1015467.1015474

[54] N. Baranasuriya, V. Navda, V. N. Padmanabhan, and S. Gilbert, "QProbe: Locating the bottleneck in cellular communication," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '15. ACM, 2015, pp. 33:1–33:7. [Online]. Available: http://doi.acm.org/10.1145/2716281.2836118