Coordinating Rolling Software Upgrades for Cellular Networks

Mubashir Adnan Qureshi*, Ajay Mahimkar†, Lili Qiu*, Zihui Ge†, Max Zhang†, Ioannis Broustis†
The University of Texas at Austin*, AT&T†

***Abstract*—Cellular service providers continuously upgrade their network software on base stations to introduce new service features, fix software bugs, enhance quality of experience to users, or patch security vulnerabilities. A software upgrade typically requires the network element to be taken out of service, which can potentially degrade the service to users. Thus, the new software is deployed across the network using a rolling upgrade model such that the service impact during the roll-out is minimized. A sequential roll-out guarantees minimal impact but increases the deployment time thereby incurring a significant human cost and time in monitoring the upgrade. A network-wide concurrent roll-out guarantees minimal deployment time but can result in a significant service impact. The goal is to strike a balance between deployment time and service impact during the upgrade. In this paper, we first present our findings from analyzing upgrades in operational networks and discussions with network operators and exposing the challenges in rolling software upgrades. We propose a new framework *Concord* to effectively coordinate software upgrades across the network that balances the deployment time and service impact. We evaluate *Concord* using real-world data collected from a large operational cellular network and demonstrate the benefits and tradeoffs. We also present a prototype deployment of *Concord* using a small-scale LTE testbed deployed indoors in a corporate building.**

I. Introduction

Today, cellular networks are one of the top drivers for communication, Internet access, and multimedia applications. Smartphone users heavily rely upon them and expect high availability at all times. The cellular service providers aim to provide excellent quality of experience for billions of smartphone users by continuously monitoring the network and service performance. They introduce upgrades to their network on a regular basis in order to improve service experience, roll out new service functionalities, fix software bugs, or patch security issues. Network upgrades typically involve new software releases, firmware upgrades, hardware modifications, configuration parameter changes, and topology changes.

Some of these upgrades (*e.g.*, software upgrades) require the network element to be transiently taken out of service, which can potentially induce a service impact. The service providers carefully plan the network upgrades during off-peak time to minimize the service impact. Typically, night time (often referred to as maintenance windows) is preferred due to low traffic volumes. By executing upgrade sequentially (one element after another), one can maximize the spare capacity in the network to offload the traffic when the network element is undergoing an upgrade. This guarantees minimal impact, but increases network-wide deployment time. A higher deployment time means increased human cost in monitoring the upgrade. For large-scale cellular networks with thousands of base stations, the sequential roll-out is definitely not an option because of the cost and delayed availability of the upgrade across the network.

978-1-5090-6501-1/17/$31.00 ©2017 IEEE

Another option for network-wide deployment is simultaneously upgrading all the network elements. This will guarantee the fastest roll-out but incurs a significant impact to service and thus is undesirable. The cellular service providers try to strike a balance between deployment time and service impact for rolling out the upgrade. They roll-out the upgrade in such a way that there is spare capacity in the network to accommodate most of the traffic demands. For example, upgrade K out of N base stations simultaneously such that the remaining $N-K$ base stations can serve the users with similar quality of service. In the next round, one can then upgrade some of the $N-K$ base stations while the other base stations (including those that finish upgrade) serve the users. This method of deploying the upgrade is commonly known as a rolling upgrade model. The rolling model often leverages multiple rounds. There are some interesting challenges with deploying the upgrades using the rolling model:

1. How do you choose the network elements to be upgraded in each round such that the service impact is minimized?
2. How do you choose the number of rounds such that the deployment time and cost is minimized?

We interacted with operations teams from a large cellular service provider to understand their network upgrade deployment process. We made a few key observations:

1. The deployment time is driven by the OSS-limits on how many concurrent upgrades can be deployed on the network elements. An OSS (Operational Support System) serves multiple network elements in a region and the current software implementations impose an upper bound on how many concurrent instructions can be executed.
2. The upgrades comprise multiple steps such as pre/post health checks, traffic migrations, mechanisms (*e.g.*, software download and installation for the upgrade), configuration snapshot, and occasionally a reboot of the element. Traffic migration and reboot can significantly impact service and should be executed at off-peak times to minimize the impact.
3. The upgrades are executed for elements that are geographically nearby. By focusing on elements within a region,

allocation for human resources for monitoring the upgrade is simplified. For example, the operations team in New York City monitors upgrades on base stations locations in New York City.

4. Within a time slot (maintenance window typically ranges from midnight to 6 AM local times), upgrades are executed concurrently for a fraction of elements within a region (*e.g.*, K out of N base stations in New York City) and the concurrence limit is bounded by the OSS-limits.
5. Different operation teams are responsible for different types of upgrades (*e.g.*, different teams for major software upgrades versus configuration parameter changes). Explicit coordination is required for scheduling of these upgrades so as to minimize any conflicts that could have occurred in multiple teams trying to grab the same time for the upgrade. In some cases, one has to carefully capture the dependencies, such as applying global parameter changes on the base stations after major software releases.

We also use real-world data collected from operational cellular network to analyze the upgrade roll-out process and their service impacts. The data sets include radio coverage data using ATOLL [3], traffic and service performance measurements. We make the following observations:

1. Upgrades occur very frequently. They are predominantly executed on weekdays and non-holidays primarily because of the staffing around that time.
2. For the same type of upgrade (*e.g.*, software upgrade from version X to version Y), different base stations take different amount of time to complete the upgrade. The different steps in the upgrade can take different amount of time across different network elements.
3. We capture the service impact of the upgrade by comparing service performance within the impact scope of the upgrade. For example, we compare performance on the upgraded elements before the upgrade with the aggregate performance where the traffic gets re-routed during the upgrade. We observe a slight degradation in performance during some of the upgrade. We analyze the impact as a function of the number of elements undergoing the upgrade and the tradeoff. This presents an interesting opportunity to determine the optimal set of the network elements to be upgraded concurrently such that the impact is minimized.

Our approach and contributions: We present a new framework *Concord* to effectively coordinate upgrades across the network that balances the deployment time and service impact. In this paper, we focus on scheduling upgrades on the LTE and UMTS cellular base stations using the rolling model. *Concord* provides the tradeoff between deployment time and service impact, and lets the operations team choose what best suits their needs. Our service metric is a combination of coverage (capturing if any users lose connectivity to their cellular network during the upgrade) and achievable data throughput (capturing any service performance degradation). We formulate the problem as a joint optimization for coverage and capacity constraints. We propose a new approach to model the dependency between the cellular base stations using coverage maps, traffic and performance measurements. This is different from traditional models based on neighbor relations or geographical distances. Our novelty lies in (i) identifying a new and important problem in cellular network management, (ii) proposing new models and demonstrating their effectiveness, and (iii) adapting the optimization algorithms for the new models.

We evaluate *Concord* using real-world data collected from a large operational cellular network and demonstrate the benefits and tradeoffs. We conduct experiments using a large number of base stations in a region primarily covering urban and suburban settings and across multiple dimensions including day versus night time for upgrade, coverage and capacity constraints, and across multi-layer technologies (LTE and UMTS). We show that *Concord* can achieve more than 6x improvement in completion time over sequential upgrade process and in different settings, it can achieve up to 91% improvement in completion time over basic concurrent schemes. We also present a prototype deployment of *Concord* using a small-scale LTE testbed deployed indoors in a corporate building. Our testbed consists of LTE base stations, user equipment (UE), and an evolved packet core (EPC). The prototype system presents a proof-of-concept evaluation of *Concord*.

Paper outline: The rest of the paper is organized as follows. In Section II, we present our analysis and findings using upgrades and performance data collected from a large operational cellular network. In Section III, we present the design and implementation of *Concord*. We present our evaluation results using real-world data in Section IV and using testbed experiments in Section V. We review related work in Section VI, and conclude in Section VII.

II. Motivation

Operational network data analysis: We use one year worth of operational data collected from cellular network to infer software upgrades on LTE base stations. We make the following observations. First, there are planned upgrades every day throughout the year. Weekdays and non-holidays see more upgrades likely because upgrades are currently involving semi-automated processes and there are more staff during weekdays and non-holidays to manage upgrades. Second, the upgrade takes time: often around a few hours on a single base station (BS) and can take on the order of weeks to months for tens of thousands of base stations in a cellular network. Third, if not scheduled carefully, it results in a service disruption.

These observations motivate us to automate the scheduling process that not only speeds up the deployment of the new software across the network but also minimizes the disruption to service performance. We envision our scheduler to develop schedules for both (a) compile-time - planning in advance - days before the upgrades are actually executed, and (b) run-time - pre-execution check - minutes or hours before the upgrade to take into account any network changes.

Schedule robustness: Pre-planned upgrade scheduling may lack robustness, which can cause longer roll-out upgrade times. Consider a pre-planned upgrade schedule for a set of base stations. For simplicity, assume that the BSes are split in two groups $\mathcal{A}$ and $\mathcal{B}$ and all BSes in one group are collectively upgraded in a single round. If $\mathcal{A}$ is scheduled first and some jobs in $\mathcal{A}$ remain unfinished when we have to schedule $\mathcal{B}$, then the operators are left with two options: (i) wait for the jobs in $\mathcal{A}$ to finish and then reschedule $\mathcal{B}$ on some other day or (ii) go ahead with the planned schedule and upgrade $\mathcal{B}$, which may lead to serious service impact. Another way to resolve the problem might be to space out the scheduling of two sets by some suitable time where there is no chance of scheduling conflict, but this can drastically increase the overall completion time of the upgrade process.

Figure 1 shows the CDF of time spent in upgrading a LTE base station. We collected this data during September 2016 - February 2017 for tens of thousands of base stations. To protect proprietary information, we have removed the X-axis. X-axis is the upgrade duration in linear time scale. The important thing to learn is that even for the same upgrade, different base stations took different amount of time to complete. Difference between T1 and T2 is in granularity of minutes.

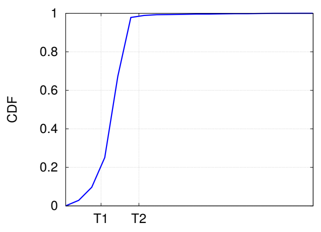


Fig. 1. CDF of upgrade time per eNodeB

Nodes dependencies in cellular domain: There is rich literature on scheduling dependent jobs in cloud environment and multiprocessors. Generally, dependencies between jobs are specified using DAG (*Directed-Acyclic-Graph*), where an edge from a parent node to a child node specifies that the child node cannot proceed until the parent node finishes execution. However, DAG cannot model wireless interference constraints since DAG cannot capture the spatial constraints between base stations.

Consider four base stations with IDs from 1 to 4. They cover multiple regions but let us focus on two regions. One region α is covered by the BS set $\{1, 2, 3\}$ and other region β is covered by BS set $\{1, 2, 4\}$. If we consider only coverage constraints, then BS 1 can be upgraded under three scenarios, (i) BS 2 is down and BS 3, 4 are up, (ii) BS 2 is up, (iii) BS 2, 3, 4 are up. This is a relatively simple case; traffic constraints along with a larger number of nodes can lead to an exponential number and more complex relationships between nodes. *DAG* cannot capture these relationships. Moreover, exploring all these relationships between nodes for a whole topology might be too expensive to compute. So we need a lightweight and scalable framework to tackle these dependencies.

Need for fine-grained modeling of BS coverage regions: Here we motivate the need of a fine-grained model of regions covered by individual BSes to define dependency relationship between nodes. A practical way to schedule a base station in an LTE network (also referred as an eNodeB in the rest of the paper) is by using X2 neighbors. X2 interface is established between eNodeBs to exchange information like handovers. We can use a simple greedy approach to schedule eNodeBs using X2 neighbor relationship. We start from an empty list $\mathcal{L}$ and iteratively add nodes if none of their X2 neighbors are in $\mathcal{L}$. We continue adding nodes until we exhaust the list. We schedule these eNodeBs and repeat this process until all eNodeBs have been upgraded.

To see how this scheme would work, we collected X2 neighbors data for an urban area with 105 eNodeBs. We used our coverage model based on ATOLL (Section III-B) to get the traffic demand in each 100m$\times$100m area served by these 105 eNodeBs and also the maximum throughput that can be sustained for that 100m$\times$100m area by corresponding eNodeBs. We used the greedy approach to schedule eNodeBs and found that because of coarse and conservative definition of X2 neighbor relationship, completion time of the upgrade process increases by 50%-65% when compared to *Concord* with similar service impact. This shows that our fine-grained approach using ATOLL based coverage model helps speed up the upgrade deployment and maintain similar service impact as the X2 neighbor based approach.

III. Scheduling Upgrades

A. Overview

The upgrade workflow consists of the sequence of instructions to execute the upgrade. When the upgrade workflow is ready, the operation team queries the scheduler at planning time to seek a range of dates and the order for the upgrade. Once the schedule is ready and approved by the operations team, it will be passed to an orchestrator that prepares for the execution of the upgrades. The orchestrator queries the scheduler at run-time and obtains the schedule to be followed for executing the upgrades across multiple base stations in the cellular network. Since this operation is conducted at run-time, the scheduler seeks the most recent view of topology, traffic, and performance from the underlying cellular network and provides an order of the nodes for executing the software upgrade. Depending on the schedule, the orchestrator then instructs the controllers to execute the instructions on the appropriate base stations. Our focus is the design of an automated scheduler. We also develop simple solutions for the other components in our online testbed experiments, and defer in-depth study of these other components to the future work.

Scheduling upgrades in cellular networks is important since it not only impacts the time to complete upgrade, but also significantly impacts the user performance during the upgrade. There are a number of desirable goals:

- Minimize time to complete upgrade in order to save operational costs;
- Minimize coverage holes and congestion during upgrades to ensure a maximum number of clients can be served by remaining BSes while the other BSes are brought down for upgrades. Some clients cannot be served when

their corresponding BSes go down due to power/terrain limitations;

- Support networks with different generations (*e.g.*, 2G, 3G, 4G clients and BSes).

Job scheduling has been studied extensively. [16], [11], [19] provides an excellent survey of the scheduling algorithms. Abstractly, a scheduling problem is defined by the machine environment (*e.g.*, one machine vs. multiple machines that can accept jobs), optimality criterion (*e.g.*, average completion time, worst-case completion time, weighted average completion time, or minimizing the number of jobs missing their deadlines), and constraints (*e.g.*, preemptive vs. non-preemptive, or precedence constraints). Thousands of scheduling problems have been studied depending on the specifications of these three components. Scheduling upgrades in cellular networks differs from the existing scheduling work due to the new constraints (*i.e.*, minimizing coverage holes and congestion). Here the job schedule is not only limited by the capacity constraints, but also affected by the the locations of the base stations and clients (*e.g.*, not all BSes covering the same area can be upgraded together to ensure coverage).

Our main contribution is the design of a framework where we incorporate a cellular network topology and capacity into optimization for upgrade scheduling. We first present the data sets used in the paper, followed by the formulation and algorithms in our new framework *Concord*.

B. Trace Description

Our scheduling algorithm requires network topology and traffic demands as the input. We use operational network traces ranging from coverage data, traffic and performance measurements collected by the cellular service provider. Coverage information is calculated and stored using a modeling tool called as ATOLL [3]. Traffic and service performance information is collected from the network element management systems every 15 minutes.

For modeling cellular network coverage, ATOLL divides the geographical area into 100m x 100m grids. It assumes that users within the grid have same coverage and service performance. ATOLL provides the path loss information for each grid and sector pair. Path loss captures the signal attenuation from the sector towards the grid and is measured in dBm. ATOLL uses standard propagation model (SPM) for calculating path loss and accounts for the distance between the sector and the grid, sector antenna height, terrain type (*e.g.*, mountains, trees, etc.), carrier frequency, and antenna tilt/gain.

We calculate the received power (or, signal strength) for each grid by subtracting the path loss from the transmission power of the sector. Let us denote g as the grid, L_s^g as the path loss from the sector s towards grid g in dBm, and T_s as the transmission power from the sector s in dBm. Then, the received power R_s^g is given by $R_s^g = T_s - L_s^g$. Then the signal to noise ratio (SNR) is calculated by subtracting noise (-104.5 dBm) from the received power.

If the SNR is below a threshold, we conclude that the grid is out of service. We map the SNR to the maximum achievable data rate using the Modulation and Coding Scheme (MCS) based model defined in 3GPP LTE standard.

We use uplink and downlink PDCP (Packet Data Convergence Protocol) byte volume to represent traffic demands and user throughput to represent service performance. To handle fluctuations in traffic, we average the byte volume over 10 weekdays. Traffic and service performance data is currently collected by the cellular service provider on a per sector basis and is not available on a per grid basis. To check if there is network congestion, we require traffic demand for each grid. To generate the per-grid traffic demand, we identify the best sector for each grid that gives the maximum SNR. We then proportionally map the traffic demand at the best sector to the traffic demand for each grid that the sector is serving. For example, a sector has traffic demand of 100 Mbps. It serves two grids: grid 1 has a rate of 2Mbps and grid 2 has a rate of 1 Mbps. We set the grid 1's traffic demand to 66.7 Mbps and set the grid 2's traffic demand to 33.3 Mbps.

C. Concord Basic Formulation

We first consider the basic version of the problem where we need to schedule upgrades while ensuring coverage and no congestion. We explain these constraints incrementally to make them easily understandable.

We start with ensuring coverage constraints such that while some BSes go down for upgrade, we want all serviceable regions to remain covered. Some areas cannot be served when their serving base stations go down since their next closest base stations are too far away. However, such areas are very small in number. Suppose we want to upgrade BSes in a given area. The area has W BSes. Among them, we need to upgrade U BSes. We use our ATOLL model to map this area to 100x100m grids, where for each grid we measure the received signal strength (RSS) from all BSes it can hear. If the grid receives RSS higher than a given threshold from a BS, we say the grid can be served or covered by the BS. Based on this information, our goal is to minimize the amount of time it takes to complete upgrades while ensuring all grids in the area is covered by at least one BS. For simplicity, we first assume all upgrades take the same amount of time and aim to minimize the number of rounds during the upgrade. In Section III-F, we minimize the upgrade time where different BSes may have different upgrade time. We observe that if a grid is covered by only two BSes (*e.g.*, A and B), then these two BSes cannot be upgraded at the same time to ensure coverage. In general, for a grid that is covered by N BSes, these N BSes cannot be upgraded at the same time.

The above formulation only ensures every grid is covered by at least one BS. In cellular context, we should ensure not only coverage but also no congestion (*i.e.*, the traffic from all grids can be served by the remaining active BSes). In order to tackle the no congestion scenario, we partition all BSes that require upgrade into the minimum number of partitions, where after removing each partition the coverage and capacity

$\forall i \in Grids, j \in BSes$
$\triangleright\ Input: Rate(i,j),\ Demand(i),\ Cap(j)$
$\triangleright\ Output:\ T(i,j),\ x(i,j)$
maximize: $\sum_{i,j} T(i,j)$
subject to:

$$\begin{array}{lll}
[C1] & \sum_j T(i,j) \le Demand(i) & \forall i \\
[C2] & T(i,j) \le x(i,j) Rate(i,j) & \forall i,j \\
[C3] & \sum_i x(i,j) \le 100\% & \forall j \\
[C4] & \sum_i T(i,j) \le Cap(j) & \forall j \\
[C5] & x(i,j) > 0, T(i,j) > 0 \forall i,j & \\
[C6] & \text{Controller Level Constraints} &
\end{array}$$

Fig. 2. Problem formulation to maximize traffic that can be served by a given set of base stations, where $T(i,j)$ is the amount of traffic from grid i that can be served by BS j, $Rate(i,j)$ is the data rate between grid i and BS j, $Demand(i)$ is grid i's traffic demand, $Cap(j)$ is BS j's server or wireline capacity whichever is smaller.

constraints should still be satisfied. In order to minimize the number of partitions (or upgrade rounds), we try to maximize the size of schedulable set in each round. The issue here is to compute the amount of traffic that can be served by a given set of BSes.

Maximizing traffic served by a set of BSes: We formulate the following linear program to maximize the traffic demand that can be served by a set of BSes. This captures the state of the network when nodes go through upgrade, determining the maximum amount of traffic that can be served by the remaining nodes. If the maximum demand is equal to the total generated traffic demand, then there is no congestion. We use our ATOLL model from Section III-B to calculate SNR that each base station can sustain for each 100m$\times$100m grid and the traffic generated in that grid. To compute the throughput a set of BSes can support, we construct a linear program as shown in Figure 2. Its objective is to maximize the total traffic that can be served by a set of BSes. [C1] reflects the total traffic that can be served from a grid is no more than its traffic demand. Here we assume that different BSes may serve a grid, which is common in reality. [C2] reflects the total served traffic from grid i cannot exceed $Rate(i,j)x(i,j)$, where $Rate(i,j)$ is the data rate between grid i and BS j and $x(i,j)$ denotes the fraction of time the BS j spends serving the grid i. $Rate(i,j)$ is determined by the signal-interference-noise ratio (SINR). We map SINR to $Rate$ according to 3GPP LTE standard. [C3] reflects that the amount of time BS can serve all grids cannot exceed 100%. [C4] indicates total traffic each BS can serve is bounded by its capacity (*e.g.*, the minimum between the server capacity and wireline capacity at the BS). [C6] reflects the controller level constraints. For example, we impose the restriction on the number of instructions that can be processed through an OSS based on the set of scheduled nodes.

D. Algorithm

It is desirable to minimize upgrade time for the following reasons. First, it reduces operation cost since the earlier the upgrade finishes, the less time the staff needs to spend in monitoring upgrade. Second, end users across the network can start enjoying the new features and improved performance that upgrades bring earlier. Therefore, we use upgrade time along with the impact on service quality as our performance metrics.

We minimize the number of upgrade rounds by developing a greedy heuristic, which maximizes the number of nodes to upgrade in each round. Every round, we pick a large schedulable set such that all nodes in the set can be upgraded together without any coverage holes or congestion. As we will show, this greedy heuristic performs close to the optimal.

Finding independent schedulable set: Our goal is to identify the biggest scheduling set $\mathcal{S}$ that consists of nodes which can be updated simultaneously with the aim of minimizing the upgrade time. In order to add nodes to $\mathcal{S}$, we can either pick the nodes in a random manner or use some metric based selection. We employ the following metric. Each node is assigned with a node degree, defined as $max_g \frac{1}{N_g}$, where N_g is the number of nodes serving grid g and $\frac{1}{N_g}$ is the amount of coverage the node provides to the grid g. A node with a higher degree is less likely to be upgraded together with other nodes than a node with a lower degree. For example, a node with degree 1 means it is the only covering node for some grid and cannot be upgraded without coverage holes. Inspired by the heuristic to find a large independent set, we sort all BSes to upgrade in an increasing order of their node degrees. In later sections, we will use a different sorting metric that suits the differing formulation.

We compute $\mathcal{S}$ in two steps.

- We add the first node from the sorted list to $\mathcal{S}$ and then incrementally add one node at a time as long as upgrading all nodes in the list does not cause any grid to have no node covering it. To achieve that, every time a node i is added to $\mathcal{S}$, we remove it from the grids it covers. We just need to check if there exists a grid such that i is the only node covering it. We continue until we exhaust the sorted list.
- We then formulate our optimization problem from Figure 2 to check if congestion constraints are satisfied. If not, we remove the nodes one by one from the end of $\mathcal{S}$ until the congestion constraints are satisfied.

This forms a set of nodes to upgrade in the first round. After these nodes finish upgrading, we remove them, since they no longer need upgrade. We repeat the process until all the nodes have been upgraded.

Lower bound: It is desirable to know how far our algorithm is from the optimal. It is hard to exactly compute the minimum upgrade rounds. However, we can derive a lower bound. It is not difficult to see that if we find a set of nodes such that none of any two BSes in this set can be upgraded together (called our notion of clique), the size of this set imposes a lower bound on the number of upgrade rounds. This is because all nodes in this set need a separate upgrade round. The size of any clique is a lower bound. The larger the clique we find, the tighter the lower bound. However, even the maximum clique may be a loose bound because (i) scheduling another clique after the first clique may require adding one or more new rounds, (ii) Cliques are made using only coverage constraints. In principle, one can find cliques using no congestion constraints, but this is much more expensive.

Finding largest cliques: We need to define conflict relationship between nodes in order to determine the largest clique. Two nodes are in conflict if they cannot be upgraded together. To define conflict relationships between nodes, we make use of a binary conflict matrix $\mathcal{M}$. $\mathcal{M}_{ij} = 1$ indicates there is a conflict between the nodes i and j. We populate $\mathcal{M}$ using our ATOLL model where if a region is covered by only two nodes i and j, then we mark $\mathcal{M}_{ij} = 1$. In order to find a clique, we start with a random order of nodes and one initial node in clique set $\mathcal{C}$. We add a node j to $\mathcal{C}$ if for all nodes i in $\mathcal{C}, \mathcal{M}_{ij} = 1$. We continue this until the list of nodes is exhausted. To tighten the bound, we enumerate multiple cliques and use the size of largest clique as the lower bound.

E. Supporting Multiple Upgrades Per BS

We learn from the operation team that a BS may require multiple upgrades one after another. To support multiple upgrades with ordering requirements on a BS, we make the case that it is preferable to schedule nodes with more upgrade tasks early since all their upgrade tasks have to finish before the upgrade is complete. Therefore, instead of sorting nodes based on the node degree, here we first sort nodes in terms of the number of upgrade tasks and further sort the nodes with the same number of upgrade tasks in an increasing order of their node degree to increase of the chance of getting a larger schedulable set.

F. Supporting Variable and Uncertain Upgrade Time

So far, we assume all BSes can be upgraded in the same amount of time so that we can just minimize the number of upgrade rounds. Often the time required to upgrade BSes vary significantly due to difference in vendor or processing power. Due to some unforeseen circumstances, certain nodes may also take an uncertain amount of time to complete upgrade.

We make two important modifications to support this variant. First, we sort the jobs in a decreasing order of their upgrade time so that we are more likely to schedule larger jobs earlier since the upgrade can only finish when the last job finishes. Note that this does not hamper other smaller jobs from being scheduled if they have no conflict with the already scheduled jobs. Second, since not all these jobs finish at the same time, whenever one job in the current round finishes, we may potentially schedule new job(s) since the BS that just finishes upgrade may serve some grids and allow new BSes to be upgraded without violating coverage or congestion requirements. This scheduling also takes care of uncertain upgrade times of jobs, because even if a job takes longer than expected, it does not affect the scheduling of nodes that are not in conflict with this job. We continue adding nodes one at a time to the set of nodes that is still going through upgrade as long as all nodes are schedulable after insertion. In this way, we achieve online scheduling even in the presence of uncertain upgrade time.

G. Practical Considerations

Trade-off between upgrade time and performance: So far, we enforce no coverage hole or no congestion during upgrade. In practice, one may tolerate minor performance degradation during upgrade. This is because sometimes it might be desirable to complete the upgrade process as soon as possible while tolerating some performance hit based on the necessity of the required upgrade. For example, some important tweak in security configuration may be required on an urgent basis, so operators may be willing to incur some performance hit. Our scheduling algorithm can be adapted so that it allows trade-off between coverage holes/congestion and performance by introducing two tolerance threshold parameters. For coverage, we consider a set of nodes as schedulable if the number of coverage holes is within a threshold (denoted as $thresh_{holes}$). For congestion, we consider a set of nodes as schedulable if the percentage of unsatisfied traffic demand is no larger than threshold, $thresh_{cong}$. $thresh_{holes}$ and $thresh_{cong}$ can be adjusted according to the service requirement.

Supporting Different Generations: A base station may run 2G, 3G, and 4G, which can be upgraded independently. For example, 4G part of the base station is going through upgrade, while 2G and 3G are serving clients. A challenge is how to pick base stations to support the clients that ensure no congestion across generations. The new challenge comes from the coupling between the generations (*e.g.*, 4G clients may be served by 2G, 3G, or 4G). To ensure no congestion across generations, we construct a connectivity graph using BSes and grids as follow. A base station running 2G, 3G, 4G are represented as 3 nodes and the outgoing edges from these nodes have capacities set to the BS capacity of that generation. Moreover, each grid has 2G, 3G, 4G traffic; 2G base station can serve 2G, 3G, 4G clients, 3G BSes can service 3G and 4G clients, 4G BSes can only serve 4G clients. To reflect these properties, we use 3 nodes to represent a grid: one for each generation. We connect a 2G grid to 2G BSes to reflect that 2G traffic demands can be served only by 2G BSes. Similarly, we connect a 3G grid to 2G and 3G BSes to reflect that both 2G and 3G BSes can possibly serve 3G traffic, and connect a 4G grid to 2G, 3G, and 4G BSes. We can use the algorithm in Section III-D to maximize the amount of traffic to serve by a given set of BSes. The only difference is that the underlying graph becomes larger since we use different nodes to represent a grid that generate traffic demands for different generations and use different nodes to denote BSes supporting different generations.

IV. Evaluation

In this section, we evaluate our approaches in *Concord* using trace-driven simulation. We first introduce our evaluation methodology and then present performance results.

A. Evaluation Methodology

Baseline schemes: We compare our scheduling algorithms with the following baselines:

- *Serial upgrades (serial):* In this scheme, base stations are upgraded one by one. This is not the practice in operations today because it would take too long to complete the deployment. However, it ensures the least performance impact and the total upgrade rounds is equal to the number of base stations being upgraded.
- *Basic parallel upgrades (basic-para):* A better scheme is to use the cellular topology and simultaneously upgrade base stations that do not violate coverage or capacity constraints. We start from a complete list of eNodeBs that is assumed to be the schedulable set. We remove nodes from the current upgrade list according to a random permutation until all constraints are satisfied. Note that this is already better than the current practice, which does not explicitly take into account the topology and traffic demands to minimize the upgrade time while ensuring no service disruption. Current operational standard is to wait for all nodes in the round to complete before moving onto the next round because conflict resolution is not employed in real time.
- *Lower bound (lowbound):* As described in Section III-D, the largest clique size is a lower bound. Our evaluation enumerates 1000 cliques and pick the largest out of them.

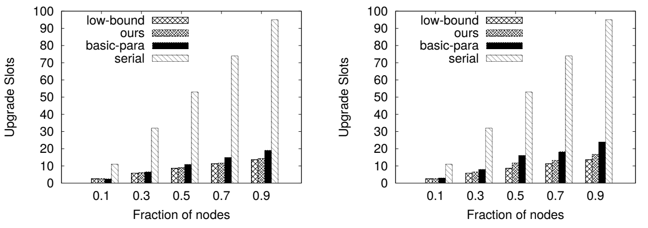
B. Performance Results

The topology that we consider comes from one of the largest cities in US, it consists of 105 nodes covering majorly urban and some suburban area.

Figure 3(a) shows the total upgrade slots if upgrade process happens only during midnight hours from 00:00 - 6:00 and (b) shows the upgrade process for daytime hours from 9:00-17:00. During daytime, upgrade process needs more slots because there is higher traffic during daytime so in order to satisfy "no congestion" constraints, we need more nodes to remain ON in a round. However in both cases, our algorithm's performance is very close to the lower bound. The lower bound here only ensures coverage but not "no congestion" so the lower bound is not tight. As compared to *basic-para*, we can update the eNodeBs at a 30% faster in some scenarios when upgrade time is the same for all eNodeBs. When upgrade times are not equal, *Concord* achieves significant improvement over *basic-para*, as shown later.

All the results presented from this point onwards use traffic demand data from the midnight hours because preferential operational practice is to perform the upgrade process during night time, when traffic is low and the impact is small.

Multiple upgrades per BS: Figure 4 shows the number of upgrade slots required when eNodeBs can have multiple upgrade jobs. Our ordering of eNodeBs according to the job order yields 50% reduction in the overall completion time.

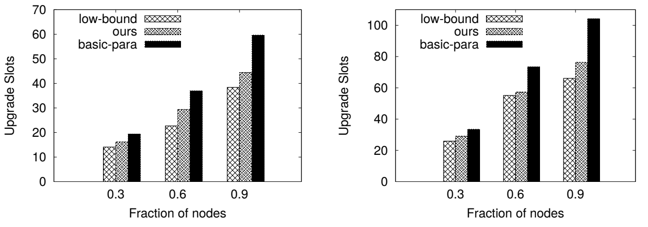


(a) Night Time (b) Day Time

Fig. 3. Comparison of upgrade time while ensuring no congestion. As traffic increases (low traffic at night time versus high traffic at day time), # required upgrade slots also increases.

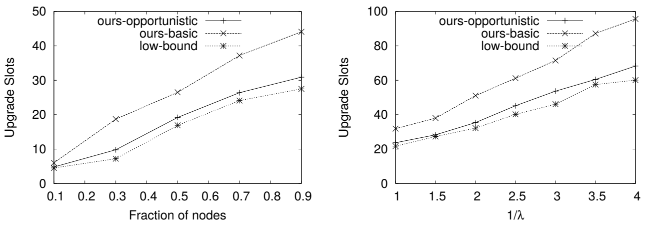
Heterogeneous upgrade time: Figure 5(a) shows the results by setting the upgrade time according to the real traces, and Figure 5(b) shows the results using exponentially distributed upgrade time. In both cases, we know exactly what the upgrade time will be. Our opportunistic insertion of jobs gains up to 46% improvement because as the nodes get upgraded and re-enter the grid, conflicts get resolved thus allowing more nodes to be scheduled for upgrade.

Figure 6(a) shows results when actual upgrade time of a node is set using normal distribution where mean is the expected upgrade time from traces and variance in number of upgrade slots is varied 1-7. Figure 6(b) shows the results when variance is fixed at 7 but the number of nodes to upgrade is varied. *Concord* yields 30%-91% improvement in completion time over *basic-para* because *basic-para* does not employ opportunistic insertion. The upgrade time of *Concord* is within 5% of that of the lower bound.



(a) 5 Maximum Jobs (b) 10 Maximum Jobs

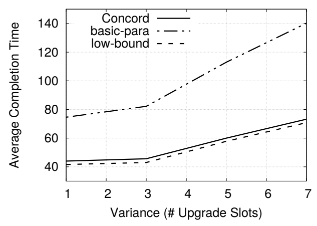
Fig. 4. Comparison of upgrade time while also ensuring ordering constraints when maximum # upgrade jobs assigned to a base station are 5 and 10.



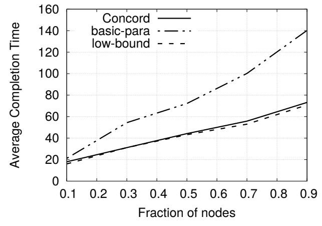
(a) real traces (b) exponential distribution

Fig. 5. Comparison of upgrade time under known variable upgrade time.

Tradeoff between upgrade time and performance: Figure 7(a) shows the number of upgrade slots as we relax the coverage constraint – allowing more grids to be uncovered during the upgrade progress. The three curves in the figure correspond to 30%, 60% and 90% of eNodeBs requiring upgrade. The number of upgrade slots decreases sharply as we go from 0 to 50 coverage holes, and tapers off around 100 coverage holes since the upgrade slot is already very small



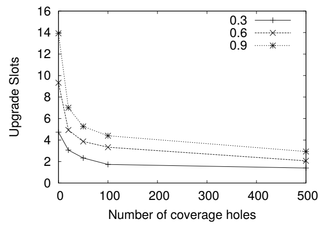
(a) Differing Variance



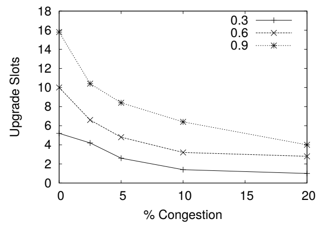
(b) Varying Nodes

Fig. 6. Comparison of upgrade time under uncertain upgrade time.

around 100 coverage holes and further reducing the upgrade slots results in a large number of grids to go uncovered. Figure 7(b) shows the number of upgrade slots as we relax the "no congestion" requirement. We use *congestion fraction* to refer to a fraction of total flow that cannot be served. As we would expect, increasing the congestion fraction reduces the number of upgrade slots. As before, we observe a similar slow-down in the reduction of the upgrade slots for the same reason.



(a) Coverage Constraints Only



(b) Congestion Constraints

Fig. 7. Tradeoff between performance and upgrade completion time. Increasing the tolerance for service impact reduces # slots needed for the upgrade.

Supporting Different Generations: Figure 8 compares the upgrade time between homogeneous and heterogeneous settings when updating 4G nodes in the network. The evaluation assumes each eNodeB can serve both 3G and 4G clients and each client can be served by both 3G and 4G eNodes. In homogeneous setting, only 4G nodes can cover for 4G nodes so upgrades are planned according to this model with 3G nodes providing no support for 4G users. In heterogeneous setting, 3G nodes can also cover for 4G nodes so there is more traffic handling capacity in the network. Because of additional capacity, we can complete the upgrade process at most 10% faster in heterogeneous setting.

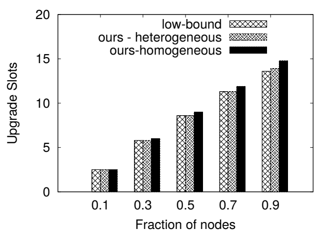


Fig. 8. Incorporating lower generations (UMTS/3G) support when upgrading LTE eNodeBs speeds up the upgrade process.

V. Testbed Experiments

In this section, we evaluate our scheduling approaches using a small-scale LTE testbed deployed indoors in a corporate building.

Setup: Our testbed consists of 4 LTE base stations (eNodeBs), 9 user equipments (UEs), and an Evolved Packet Core (EPC). Each eNodeB is a re-programmable LTE small cell and uses an exclusive 10-MHz experimental license for transmission in band 7, where the downlink and uplink frequencies are centered at 2635 MHz and 2515 MHz, respectively. The antennas on the eNodeBs are omni-directional. The UEs are deployed in the building such that each one is reachable by at least two eNodeBs. Thus, if one eNodeB is taken down for an upgrade, the next best eNodeB serves the UEs. The UEs are USB dongles and hosted by a Core-I3 Intel box with 4 GB memory that runs Ubuntu 14.04x64 Linux. The EPC consists of MME (Mobility Management Engine), SGW (Serving Gateway), PGW (Packet Data Network Gateway), HSS (Home Subscriber Server), and PCRF (Policy and Charging Rules Function Server). Figure 9 shows the testbed deployment with 4 eNodeBs (eNB-1, eNB-4, eNB-5 and eNB-6) and 9 UEs numbered (7-16) excluding 14. The EPC (not shown in the figure) is hosted at a single server and connected to the eNodeBs using Ethernet.

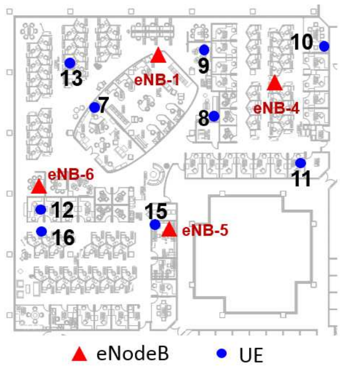


Fig. 9. LTE indoor testbed consisting of 4 eNodeBs and 9 UEs.

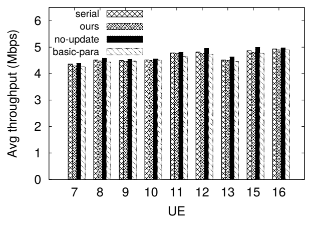
Methodology: We implement a system that takes the input the traffic demands, topology, and all eNodeBs that require upgrade. In our testbed scenario, each UE is mapped to a unique grid. We measure SNR for each UE-eNodeB pair by turning on eNodeBs one by one while keeping others OFF and asking UE to report received signal strength and noise to compute SNR. SNR is then mapped to throughput using Modulation and Coding scheme (MCS) based model defined in 3GPP LTE standard. We generate traffic for each UE according to its traffic demand specified in the input using a UDP *iperf* server at each UE and a corresponding UDP *iperf* client at the app-server on the UE. The system applies the scheduling algorithm in Section III-D to ensure no congestion. Then according to the output upgrade schedule, it takes down the eNodeBs that require upgrades in that round to mimic upgrades. This is achieved by logging in to the eNodeB interface and writing 0x0E to RF hardware register that turns its transmitter off so that the associated UEs will handoff to other eNodeBs. After the upgrade, it brings up the eNodeBs by writing 0xDE to the same register to turn their transmitters on and takes down the eNodeBs to be upgraded in the next round from our schedule. Each UE will automatically switch to the best eNodeB at any time using a hard handoff.

We continuously measure UDP throughput before, during, and after upgrade to quantify the performance impact as well as measuring the total upgrade time. The upgrade starts after 15 seconds so that we can observe the throughput behavior during the normal conditions.

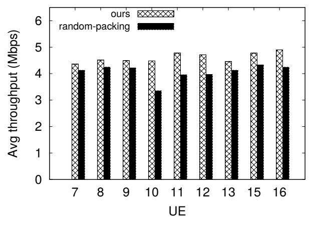
A. *Experiments*

Updating all nodes: First we evaluate the scenario where all the eNodeBs need to be updated under high traffic. Each UE is generating UDP traffic at 5Mbps. We compare throughput under three schemes *basic-para*, *serial* and *ours* when eNodeBs undergo upgrade and the time it takes to complete the upgrade process. Each algorithm outputs a schedule that is implemented on testbed using a series of steps. For example, given a schedule that says "Update enb-1 and enb-4, then Update enb-5 and enb-6", we first turn enbs 1 and 4 OFF and measure throughput over all UEs. After upgrading eNodeBs 1 and 4, we turn enbs 5 and 6 OFF and bring 1 and 4 back ON. After upgrade process is completed for eNodeBs 5 and 6, we turn ON enbs 5 and 6. So the process takes two rounds.

Results: Figure 10(a) compares the average throughput under different schemes during upgrade. There is no considerable difference between throughput using our upgrade scheduling versus no upgrade. On average, *Concord* takes 60sec, *basic-para* takes $\sim$83sec, and *serial* takes 120sec to complete the upgrade process. This indicates our scheme reduces upgrade time by 50% over the serial scheduler and 30% over the basic parallel scheduler with little to no performance impact compared to no upgrade.

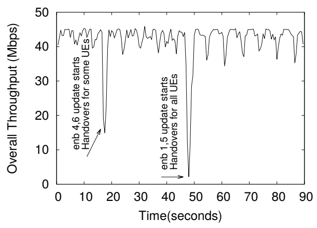


(a) Average throughput for UEs

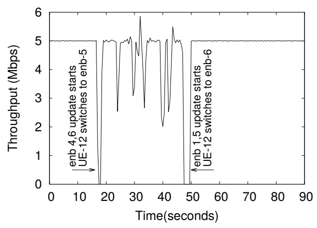


(b) Random packing in 2 slots

Fig. 10. Average throughput and upgrade time needed in high traffic scenario.



(a) Compiled Throughput



(b) UE 12

Fig. 11. Compiled throughput time series across all UEs and for UE-12.

Figure 11(a) further shows the time-series of the compiled throughput across all UEs during the upgrade process using our upgrade scheduler. As we can see, the throughput stays high throughout the upgrade except the times when the eNodeBs are just being taken down for upgrade and the UEs need to perform hard hand off to switch to the best eNodeB. This happens around 17 and 47 second marks. Figure 11(b) shows the throughput from UE-12 during the upgrade. Before 17 second, UE-12 was connected to eNodeB 6. Then eNodeB 6 went down for upgrade. So UE-12 switched to eNodeB 5. UE-12 channel with eNodeB 5 is not as good as compared to that with eNodeB 6, so there are some low throughput notches between 17-47 seconds. At 47 second, eNodeB 5 went down for upgrade so UE-12 switched back to eNodeB 6.

Serial upgrade process also causes throughput dip for users because each user has to switch to a different eNodeB when the base station, to which it is attached, goes down. Average throughput of UEs under *Concord* and serial process are similar as we can see in Figure 10(a).

Random Packing in two rounds: Next, we show the performance impact if we randomly pack eNodeBs for upgrade in two slots. We perform experiments for all possible upgrade schedules. As shown in Figure 10(b), the average throughput is much worse. UE 10 even goes out of service when we upgrade eNodeBs 1, 4, 5 in one round because eNodeB 6 cannot cover it. This highlights the importance of carefully scheduling upgrades to avoid service disruption.

VI. Related Work

Scheduling algorithms: Scheduling has been a widely studied topic. [16], [11], [19] provide excellent surveys of scheduling algorithms. Many scheduling problems are NP-hard and various heuristics have been proposed for them. Many heuristics sort jobs in terms of priorities derived based on running time or deadline. For example, shortest-job-first minimizes average completion time. Earliest-deadline-first minimizes the number of jobs that misses deadline. There are new dimensions in upgrade scheduling problem in cellular networks because of constraints arising from the topological relationships.

Cellular performance modeling: There has been significant work on measuring, modeling and optimizing cellular networks. For example, [14] deploy 3GTest to collect and analyze performance measurements. [12] extends the tool to analyze LTE from users of four major US cellular carriers. Mattar *et al.*[26] uses active TCP/UDP measurements to collect information across RF, MAC, and transport layers from a CDMA2000 network and analyze the impact of a wireless scheduler and RF on TCP parameters. The DARWIN+ group collects IP packets at a GPRS/UMTS network, and analyzes various issues, such as TCP performance and traffic anomalies [30]. [13] studies the interactions between applications, transport protocol, and the radio layer in a large LTE network in US. It reports 52.6% of TCP flows are throttled by TCP receive window. [18] develops a novel open platform to monitor and analyze LTE radio performance over both time and space. [32] develop diagnosis tools that uncover problematic interactions between control-plan protocols. Significant work has also been devoted to optimizing cellular network performance [4]. For example, [17], [5] examine resource management for OFDMA-based femtocell networks. [28] studies adaptive interference coordination in multi-cell OFMDA systems. There are several proposals [1], [2], [6], [7], [31], [27], [20], [34], [21] for dynamic tuning of cellular network configuration using SON in response to changing traffic and network conditions.

Upgrade management: Managing software upgrades and maintenance activities in large operational networks is extremely challenging. Researchers have made great efforts in designing solutions for minimizing disruptions during planned upgrades in ISP networks [8], [9], [33], data center networks [15], [22], [10], Software Defined Networks [29], [15], and LTE cellular networks [35]. In IP networks, [8] tries to avoid disruption of OSPF routing configuration upgrades, [9] avoids connectivity loss during BGP link maintenance and R3 [33] tries to recover from link failures. Software Defined Networking has been proposed to ease management of network configuration and upgrades. Reitblatt [29] presents a system that guarantees both packet level and flow level consistency during configuration changes. Dionysus [15] speeds up the consistent network updates to reduce upgrade impact. zUpdate [22] provides congestion free migration of traffic during data center network updates. [10] provides desired correctness during virtual machine (VM) migration. Magus [35] focuses on a single upgrade in LTE cellular networks and aims to proactively minimize service disruption by migrating users to neighboring base stations. Once upgrades are implemented in the network, it is important for the operations teams to carefully monitor the performance impacts [25], [23], [24], [36]. Our problem scope is different from the above mentioned works. To our knowledge, this paper is the first that studies scheduling upgrades in cellular networks.

VII. Conclusion

Managing upgrades in cellular networks is an important problem. In this paper, we propose a new framework *Concord* and develop a series of upgrade scheduling algorithms to account for different performance objectives and usage scenarios. Using testbed experiments and evaluation based on traces from a major cellular network in US, we show that our approaches significantly reduce upgrade time while incurring little to no performance impact during upgrade period. We believe that our framework can help the network operators in meeting upgrade completion deadlines while also avoiding serious service impact. Though techniques we develop are in the context of cellular networks, the general idea is applicable in other contexts by incorporating domain specific constraints.

Acknowledgment: We thank Deborah Cathey, Robert Riding, Chad Bray, Robert Mudiyappan, Jennifer Yates, Mazin Gilbert, Sarat Puthenpura, and the ICNP anonymous reviewers for their insightful feedback on the paper. We also appreciate the interactions with the AT&T Network Operations and Engineering teams that helped us in better understanding and formulation of the problem. This work is supported in part by NSF Grant CNS-1718089.

References

[1] 3rd Generation Partnership Project, TS 21.101 V8.0.0, Feb. 2009.
[2] 3rd Generation Partnership Project, TS 32.541 V10.0.0, March 2011.
[3] Atoll: a Wireless Network Design and Optimisation Platform, http://www.forsk.com/atoll/.
[4] 3GPP. Technical specification group radio access networks; 3G home NodeB study item technical report (release 8). *TR 25.820 V1.0.0 (2007-11)*, Nov. 2007.
[5] M. Y. Arslan, J. Yoon, K. Sundaresan, S. V. Krishnamurthy, and S. Banerjee. FERMI: a femtocell resource management system for interference mitigation in OFDMA networks. In *MobiCom*, 2011.
[6] U. Barth. Self-X RAN: Autonomous Self Organizing Radio Access Networks. In *IEEE WiOpt*, 2009.
[7] S. C. Borst, A. Buvaneswari, and et al. Dynamic Optimization in Future Cellular Networks. *Bell Labs Technical Journal*, 10(2), 2005.
[8] P. Francois. Disruption Free Topology Reconfiguration in OSPF Networks. *IEEE INFOCOM*, 2007, 2007.
[9] P. Francois, P.-A. Coste, B. Decraene, and O. Bonaventure. Avoiding Disruptions During Maintenance Operations on BGP Sessions. *IEEE Transactions on Network and Service Management*, 4(3):1–11, 2007.
[10] S. Ghorbani and M. Caesar. Walk the Line: Consistent Network Updates with Bandwidth Guarantees. In *HotSDN*, 2012.
[11] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals. of Discrete Mathematics*, pages 287–326, 1979.
[12] J. Huang, F. Qian, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. A close examination of performance and power characteristics of 4G LTE networks. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 225–238. ACM, 2012.
[13] 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 *Proc. of ACM SIGCOMM*, 2013.
[14] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing application performance differences on smartphones. MobiSys '10. ACM, 2010.
[15] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer. Dynamic Scheduling of Network Updates. In *ACM SIGCOMM*, 2014.
[16] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. http://people.csail.mit.edu/karger/Papers/scheduling.pdf.
[17] S. Kittipiyakul and T. Javidi. Subcarrier allocation in OFDMA systems: Beyond water-filling. *Signals, Systems, and Computers*, 2004.
[18] S. Kumar, E. Hamed, D. Katabi, and L. E. Li. LTE radio analytics made easy and accessible. In *Proc. of SIGCOMM*, 2014.
[19] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. *Handbooks in Operations Research and Management Science*, pages 445–522, 1993.
[20] F. Li, X. Qiu, L. Meng, H. Zhang, and W. Gu. Achieving Cell Outage Compensation in Radio Access Network With Automatic Network Management. In *IEEE GLOBECOM*, 2011.
[21] W. Li, P. Yu, Z. Jiang, and Z. Li. Centralized Management Mechanism for Cell Outage Compensation in LTE Networks. *IJDSN*, 2012.
[22] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz. zUpdate: Updating Data Center Networks with Zero Loss. In *ACM SIGCOMM*, 2013.
[23] A. Mahimkar, Z. Ge, J. Wang, J. Yates, Y. Zhang, J. Emmons, B. Huntley, and M. Stockert. Rapid detection of maintenance induced changes in service performance. In *ACM CoNEXT*, 2011.
[24] A. Mahimkar, Z. Ge, J. Yates, C. Hristov, V. Cordaro, S. Smith, J. Xu, and M. Stockert. Robust assessment of changes in cellular networks. In *ACM CoNEXT*, 2013.
[25] A. Mahimkar, H. H. Song, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and J. Emmons. Detecting the performance impact of upgrades in large operational networks. In *ACM SIGCOMM*, 2010.
[26] K. Mattar, A. Sridharan, H. Zang, I. Matta, and A. Bestavros. TCP over cdma2000 networks: a cross-layer measurement study. In *Proceedings of the 8th international conference on Passive and active network measurement*, PAM'07, 2007.
[27] C. Prehofer and C. Bettstetter. Self-Organization in Communication Networks: Principles and Design Paradigms. *Communications Magazine, IEEE*, 2005.
[28] T. Quek, Z. Lei, and S. Sun. Adaptive interference coordination in multi-cell OFDMA systems. In *IEEE PIMRC*, 2009.
[29] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for Network Update. In *ACM SIGCOMM*, 2012.
[30] P. Romirer-Maierhofer, A. Coluccia, and T. Witek. On the use of TCP passive measurements for anomaly detection: a case study from an operational 3G network. TMA'10, pages 183–197, Berlin, Heidelberg, 2010. Springer-Verlag.
[31] C. Shen, D. Pesch, and J. Irvine. A Framework for Self-Management of Hybrid Wireless Networks Using Autonomic Computing Principles. *3rd Annual Communication Networks and Services Research Conference*, 2005.
[32] G.-H. Tu, Y. Li, C. Peng, C.-Y. Li, H. Wang, and S. Lu. Control-plane protocol interactions in cellular networks. In *Proc. of SIGCOMM*, 2014.
[33] Y. Wang, H. Wang, A. Mahimkar, R. Alimi, Y. Zhang, L. Qiu, and Y. R. Yang. R3: resilient routing reconfiguration. In *ACM SIGCOMM*, 2010.
[34] L. Xia, W. Li, H. Zhang, and Z. Wang. A Cell Outage Compensation Mechanism in Self-Organizing RAN. In *WiCOM*, 2011.
[35] X. Xu, I. Broustis, Z. Ge, R. Govindan, A. Mahimkar, N. Shankaranarayanan, and J. Wang. Magus: Minimizing cellular service disruption during network upgrades. In *ACM CoNEXT*, 2015.
[36] S. Zhang, Y. Liu, D. Pei, Y. Chen, X. Qu, S. Tao, and Z. Zang. Rapid and robust impact assessment of software changes in large internet-based services. In *ACM CoNEXT*, 2015.