Capacity Augmentation in Energy Efficient Vehicular Roadside Infrastructure

Naby Nikookaran and Terence D. Todd
Department of Electrical and Computer Engineering
McMaster University
Hamilton, Ontario, CANADA
Email: {nikookn,todd}@mcmaster.ca

George Karakostas
Department of Computing and Software
McMaster University
Hamilton, Ontario, CANADA
Email: karakos@mcmaster.ca

***Abstract*—This paper considers the problem of capacity augmentation in energy efficient road-side unit (RSU) deployments. RSU placements for a road network, and a set of vehicular traffic flow design traces are used as inputs. The objective is to find an RSU radio capacity assignment that minimizes the long-term operating expenditure costs subject to meeting packet deadline constraints with a given packet loss rate target. A procedure, referred to as the capacity augmentation (CA) algorithm, is proposed that iterates over the RSUs, selecting candidates for capacity augmentation based on their packet loss rate sensitivities. A variety of results are presented that characterize and compare the performance of the CA Algorithm using a greedy online packet scheduler. In particular, we show how to counterbalance the lack of causality in designing the RSU network when it is used for the online scheduling of incoming transmission requests. The comparisons include those using energy-optimal offline scheduling obtained by solving an integer linear program (ILP) formulation. It is shown that the CA Algorithm is an efficient way to assign RSU radio capacity that achieves the desired performance objectives.**

I. Introduction

Many future in-vehicle applications will be enabled using vehicular ad hoc communications and networking. Roadside infrastructure is a key component of these systems and will eventually provide a platform for new local vehicular services. In these types of systems, road-side unit (RSU) costs include that of both RSU installation, i.e., CAPEX (capital expenditure) costs, and long-term operating, i.e., OPEX (operating expenditure) costs. The latter costs include those associated with wired energy usage over long time periods [1], [2]. An RSU deployment that minimizes the sum of these cost components must jointly consider both the initial RSU placement and their associated long-term service costs [3].

A characteristic of many network design methods is that they do not take into account the *causality* present in the traffic design traces that are used for the offline design, i.e., they consider the set of design trace requests as known at the beginning of the design process, and then proceed into designing a network that can accommodate them. This is typically done by solving an Integer Program. As a result, and because of the causal nature of the stream of incoming requests, the causal online scheduling during the operational life of the network may be quite suboptimal. In this paper, we consider *RSU capacity augmentation* as a method of adjusting the initial network design, and to counterbalance its failure of taking causality into account. By capacity augmentation we mean the upgrade of radio capacities of RSUs that have already been placed. Capacity augmentation in general, is not a novel idea. Once RSUs have been deployed for example, capacity augmentation is needed to update the system, thus accommodating evolving traffic conditions. Similarly, the output of an RSU design placement algorithm may not meet the packet loss targets specified in the original design specifications, which is the case considered in this paper. RSU capacity augmentation can be used in this case to provision the RSUs, thus meeting their original performance objectives. The paper introduces a procedure referred to as the capacity augmentation (CA) algorithm that can be used to perform this function.

We are given, as input, the RSU locations and their initial radio capacities, as well as any set of historical vehicular traffic flow traces used by the design algorithm (*design traces*). These traces are representative of the expected traffic flow to be accommodated by the augmented design. The objective of the design is to obtain a minimum total cost RSU radio capacity assignment that meets a given packet loss rate target, and subject to packet deadline constraints. The CA Algorithm iterates over the RSUs, selecting candidates for capacity augmentation, based on a combination of the RSU loss rate sensitivities and their capacity augmentation costs. The selection of the RSU whose capacity is to be augmented is done in every iteration by running the request scheduling algorithm on the design traces, treating them *as an online (causal) input*, i.e., under actual operational conditions. The CA Algorithm terminates when the request drop ratio improvement is below a preset threshold.

The intuition behind the CA algorithm is to trade off CAPEX (paying for the extra capacity) for decreasing the drop ratio during operation. A variety of results are presented that characterize and compare the performance of the CA Algorithm using a simple greedy online packet scheduler. The comparisons also use energy-optimal offline scheduling obtained by solving an integer linear program formulation. It is shown that the CA Algorithm achieves a very significant decrease of the drop ratio with only a very moderate (if any) increase of the network total cost.

II. Related Work

Capacity augmentation has been previously studied for a variety of different networking scenarios. Reference [4] for example, discusses capacity augmentation in wireless mesh networks in order to maximize the aggregate throughput for all network flows, and in reference [5], augmentation is proposed using free-space optical (FSO) links to enhance the capacity of RF wireless mesh networks.

Two combinatorial optimizations are used in [6] for link set capacity augmentation in networks supporting switched multi-megabit data service (SMDS). The goal is to determine the amount of additional capacity required and its location. The objective of the first formulation is to minimize the total routing cost subject to a budget constraint, while in the second, the total capacity augmentation cost is minimized subject to a set of shortest-path-routing constraints.

In reference [7] an RSU deployment and configuration problem is formulated as an integer linear program with different antenna types and power levels. The total deployment cost is minimized subject to covering a minimum desired percentage of streets with limited multi-hop packet forwarding.

To the best of our knowledge, our paper is the first that proposes a method for road-side unit capacity augmentation in vehicular networks. Our approach is unique in that the objective is to minimize the sum of capital expenditure and long-term operating costs, such that a packet loss target is achieved subject to delay deadline constraints. This is done by incorporating energy aware scheduling into the design process.

III. System Model

Let $\mathcal{S}$ be the set of candidate RSU locations, and $\mathcal{N}_s = \{1, \ldots, N_s\}$ be the set of RSU configurations. Different site locations are allowed to have a different set of configurations, e.g., different capacities, but at most one of these configurations can eventually be installed at each site location, and let $\mathcal{N} = \cup_{s \in \mathcal{S}} \mathcal{N}_s$. Let $\mathcal{V}$ be the set of vehicles serviced by the installed RSUs, each with a set of requests $\mathcal{R}_v$ for a total of $\mathcal{R} = \cup_{v \in \mathcal{V}} \mathcal{R}_v$ requests. Request r has a release date of a_r and a deadline (due date) of d_r. In this work, we assume that any job request of size ℓ_r time slots is splittable into ℓ_r unit-size (in time slots) requests with the same deadline, that can be serviced by different RSUs.

The problem that is addressed in this paper is *capacity augmentation* of an existing RSU network. Once an RSU configuration (placement and capacity provisioning) is given, the scheduling of requests are done so that at most one request of each vehicle is being serviced by any RSU, each RSU serves at most one request during each time slot, and requests are serviced before their deadline in order not to be dropped.

The energy cost for servicing a request is defined by a distance-dependent exponential path-loss model with log-normal shadowing [8]. The transmission power between a transmitter and a receiver, $P_{t,r}$, can be expressed by $P_{t,r} = P_{t,0} P_{sh} \left(\frac{d_{t,r}}{d_{t,0}}\right)^{\alpha}$, where $d_{t,0}$ is the reference distance, $P_{t,0}$ is the reference power at the reference distance, P_{sh} is a random variable that models the shadowing effect of the channel, α is the path loss exponent, and $d_{t,r}$ is the distance between the transmitter and the receiver. The shadowing effect of the radio channel is modeled as a random variable with log-normal distribution which has a zero mean (in dB) and a standard deviation of $\sigma_{\text{dB}} = 4$.

After getting an initial placement and provisioning of an RSU network (by solving the ILP formulation in [3], for example), our capacity augmentation algorithm CA (described in the following section) is run.

IV. Capacity Augmentation Algorithm

The starting point of our proposed algorithm is a placement of RSU's and their provisioning with capacities calculated by a placement and provisioning algorithm. Although any starting placement and provisioning can be used, in this work we will use the initial placement $\mathcal{N}_{\mathcal{O}}$ of RSU's, and capacities $U = u_n, \forall n \in \mathcal{N}_{\mathcal{O}}$ calculated by the algorithm in [3]. The algorithm used for the on-line scheduling of vehicle demands in a traffic will be referred to as the *Scheduling Algorithm*, while we will refer to our algorithm as the *Capacity Augmentation* algorithm. Throughout its running, the set of RSU locations $\mathcal{N}_{\mathcal{O}}$ will never change. In each iteration, our algorithm will increase the capacity of one RSU, and will test the new capacities U, by running the Scheduling Algorithm: if there is no 'substantial' improvement in the loss rate for the traffic case we are using, then the algorithm terminates.

More specifically, before every iteration (lines 6-24 in Algorithm 1), the Scheduling algorithm is run with the current capacities, and its loss rate is calculated (lines 3-5 and 21-23). In case this loss rate is smaller than the target loss rate, defined by the input parameter ξ (line 6), or the loss rate improvement in the previous iteration is not more than input parameter ζ (line 8), the algorithm terminates. The loss rate improvement is defined as the decrease of the Scheduling Algorithm loss rate within a 'window' of M iterations (where M is another input parameter).

For every RSU n, we calculate the *distributed loss rate* z_n as follows (line 14):

$$z_n = \sum_{r \in \mathcal{R}} \left(Z_r \cdot \frac{u_n \, |\mathcal{T}_{nr}|}{\sum_{k \in \mathcal{N}_{\mathcal{O}}} u_k \, |\mathcal{T}_{kr}|} \right), \tag{1}$$

where $\mathcal{T}_{nr}$ is the set of time slots during which request r can be served by RSU n. Z_r is defined to be 1 if request r is lost and zero otherwise. The idea behind this calculation is to distribute the loss of request r over all RSU's that could serve r. Each RSU n receives a fraction of r equal to the fraction of total potential (capacity unit, time slot) pairs which can be used to service r which belongs to n. Therefore, the bigger the ability (more capacity and/or more time a dropped request spends within range) of an RSU to service r, the bigger its responsibility for dropping r will be. Nevertheless, the total responsibility of RSU n for dropped requests must take into account the capital cost of increasing its capacity (to the next bigger available capacity). This cost δ_n is calculated in line 15 (in case the capacity of n cannot increase further, we set

Algorithm 1 Capacity Augmentation (CA) Algorithm

```
Input:
    • Placement of opened RSUs $\mathcal{N}_\mathcal{O}$, capacities $U = \{u_n, \forall n \in \mathcal{N}_\mathcal{O}\}$
    • Traffic trace with requests $\mathcal{R}$, time slots $\mathcal{T}$, and communication cost matrix $C = [c_{ntr}]_{\mathcal{N}_\mathcal{O} \times \mathcal{T} \times \mathcal{R}}$
    • Parameters $\xi, \zeta, M$
Output: Adjusted capacities $U = \{u_n, \forall n \in \mathcal{N}_\mathcal{O}\}$
 1: $\mathcal{T}_{nr} :=$ {time slots during which request $r$ can be served by RSU $n$} for all $n, r$
 2: $i := 0$                                    ▷ Iteration Counter
 3: Run Scheduling Algorithm ($\mathcal{N}_\mathcal{O}, U, R, T, C$)
 4: $Z_r := 1$ if request $r$ is dropped, 0 otherwise, $\forall r \in \mathcal{R}$
 5: $loss[0] := \frac{\sum_{r \in \mathcal{R}} Z_r}{|\mathcal{R}|}$          ▷ loss rate of Scheduling Alg.
 6: while $loss[i] > \xi$ do
 7:     if $i \geq M - 1$ then
 8:         if $\frac{loss[i-M+1] - loss[i]}{loss[i-M+1]} < \zeta$ then
 9:             break
10:         end if
11:     end if
12:     $i := i + 1$
13:     for all $n \in \mathcal{N}_\mathcal{O}$ do
14:         $z_n := \sum_{r \in \mathcal{R}} (Z_r \cdot \frac{u_n |\mathcal{T}_{nr}|}{\sum_{k \in \mathcal{N}_\mathcal{O}} u_k |\mathcal{T}_{kr}|})$
15:         $\delta_n$ := (capital cost after increasing RSU $n$ capacity) - (capital cost with current RSU $n$ capacity)
16:         $ratio_n := z_n / \delta_n$      ▷ loss rate per unit of cost increase
17:     end for
18:     $\mathcal{H} := \{n \in \mathcal{N}_\mathcal{O} : u_n < u_n^{\max}\}$
19:     $n_0 := \arg\max_{n \in \mathcal{H}} \{ratio_n\}$
20:     Increase $u_{n_0}$ to the next available capacity for RSU $n_0$
21:     Run Scheduling Algorithm ($\mathcal{N}_\mathcal{O}, U, R, T, C$)
22:     $Z_r := 1$ if request $r$ is dropped, 0 otherwise, $\forall r \in \mathcal{R}$
23:     $loss[i] := \frac{\sum_{r \in \mathcal{R}} Z_r}{|\mathcal{R}|}$
24: end while
```

$\delta_n := \infty$). The more expensive it is to increase the capacity of n, the less responsible it should be for the loss rate. Therefore, we assign to each RSU n a score $ratio_n = z_n / \delta_n$ (line 16). The RSU with the highest score is picked (line 19), and its capacity is increased to the next available capacity level (line 20).

V. Performance Results

In this section, we evaluate the performance of the proposed capacity augmentation algorithm. The set of opened RSUs and their capacities $\mathcal{N}_\mathcal{O}, U$ are the ones resulting from solving the (offline) ILP formulation of the problem, using a given *design traffic trace* as described in [3].[1] Given the RSU placement and provisioning, a greedy non-preemptive online scheduler is used. The scheduler tries to minimize the total service cost of scheduled job requests, by assigning each job request to the energy-wise cheapest time-slot amongst all RSU's with available capacity, as long as the deadline job constraints are met. In addition to the greedy online scheduler, in our results we will also use the optimal *offline* scheduling algorithm, which produces the minimum cost schedule that satisfies the deadline constraints when the job requests are known ahead of time. The optimal offline schedule is not implementable in our setting, since the requests are not known ahead of time, but its performance is a lower bound for *any* online scheduling algorithm (not just the greedy online scheduler used here). We will see that the simple greedy scheduler we use is not far from the optimal offline scheduler in its performance, and, therefore, we will be able to assign any performance improvements to the RSU capacity adjustments performed by algorithm CA (Algorithm 1) rather, than to the use of a particular scheduler.

The performance evaluation is done using 10 vehicular traffic traces as input. The vehicular arrivals in each trace are generated by a Poisson process with a predetermined mean arrival rate (here 1.25 vehicles per second, i.e., 2.5 vehicles per time slot). Vehicles also generate job requests according to a Poisson process, with mean arrival rates uniformly selected between 0.01 to 0.02 per time slot. The sizes of vehicular requests are generated from an exponential distribution, with mean value selected uniformly between 4 and 8 time slots. Note that each request of size bigger than one time slot is divided into multiple requests of size one with the same release and due dates, since we have assumed that job requests are splittable. In order to define the deadline for each request, the number of time slots from its release date to its due date (i.e., the request time-to-live) is picked uniformly at random between 80 to 160 time slots. The traces used in our simulations are 30 minutes in duration. The number of vehicle arrivals ranges between 1705 and 2286 (with an average of 2174), and the number of total requests ranges between 28335 and 35693 (with an average of 33939). The first trace is used by both the initial placement algorithm of [3], and by algorithm CA in order to do its capacity augmentations. The reason for this choice is the fact that both algorithms are offline algorithms, run on known past traces before an RSU placement is implemented. Then, the other nine traces are used to evaluate the effects of algorithm CA on both the cost and the drop rate.

The vehicle routes were generated by using SUMO [9]. The source and destination of vehicle trips are selected uniformly from the set of intersections, and each vehicle follows the shortest path from its source to its destination. The average travel time of each street is calculated according to its length, speed limit, and expected traffic density. Vehicles travel on a Manhattan grid with three horizontal and five vertical streets, which are all bidirectional. All intersections are controlled by traffic lights. The smallest block has a 1 km square area, which gives a total deployment region of 11.25 km^2. All RSUs are placed either at intersections or at the middle point between two intersections. There are three available RSU types, with capacity of 2, 4, or 6 respectively. All three types have a coverage range of 250 m on each side. Following [10], [11], the model we use for the CAPEX f_s of an RSU s is an affine function of capacity $f_s = f_{0s} + f_{1s} \times u_s$ for all $s \in \mathcal{N}_s$, where f_{0s} is the fixed cost for opening an RSU, and $f_{1s} \times u_s$ is the part of CAPEX which depends on RSU capacity u_s. In our simulations, we use the same CAPEX coefficients for all RSUs; these coefficients are $f_{0s} = f_0 = 4000, f_{1s} = f_1 = 2000$, in accordance with [12], [13]. This means that

[1]See `http://owl.mcmaster.ca/todd/ntk.pdf` for a full description of the ILP in [3].

the CAPEX for any RSU s is $f_s = 4000 + 2000 \times u_s$.

In assessing the total cost (CAPEX plus OPEX) in our simulations, we would like to study different weightings of CAPEX in relation to OPEX. In order to do that, we will multiply the CAPEX of an RSU with a factor we call the *single RSU capital cost factor*. We will use four different multiplicative single RSU capital cost factors ($1, 2, 3$, and 4) in our simulations; obviously, the effect of CAPEX on the total cost increases as the single RSU capital cost factor increases. The goal of these experiments will be to assess the effectiveness of algorithm CA when the influence of CAPEX to the total cost ranges from lighter to heavier.

To summarize, these are the generic steps we follow in each experiment:

1) Using the first traffic trace generated as explained above, solve the ILP [3] to calculate the RSUs placement and initial capacities.
2) Run the CA algorithm to calculate the adjusted capacities of the RSUs.
3) Run the optimal offline and the greedy online schedulers using the initial RSU configuration on the design traffic trace used in step 1.
4) Run the greedy online scheduler using the initial RSU configuration on the remaining 9 traffic traces (and average the results).
5) Run the greedy online scheduler using the CA placement on the traffic traces used in step 4 (and average the results).

Each experiment was run for each of the 4 possible single RSU capital cost factors. The rationale behind step 3 (i.e., running both schedulers on the trace used to calculate the initial RSU configuration), is the following: Using the initial placement with the optimal offline scheduler and with the design trace gives a lower bound on the performance of the initial configuration. Using the initial placement with the greedy online scheduler and with the design trace gives us an idea of how detrimental to the initial configuration is the online nature of job scheduling (although we use the design trace itself).

Figure 1 shows the performance of using algorithm CA to adjust the initial capacities. The total cost CAPEX+OPEX is shown in Figure 1a, and the drop ratio achieved is shown in Figure 1b. Both the total cost and the drop rate are shown for each of the 4 single RSU capital cost factors ($1, 2, 3$, and 4 on the x-axis). The number of opened RSUs for each one of these factors are 25, 24, 24, and 24, respectively.

Running the initial RSU configuration with the optimal offline scheduler and the design trace (i.e., the lower bound shown as "Initial + Offline Sch.") is shown with a solid black line and diamond markers. Obviously, it achieves the minimum possible total cost, and the minimum drop rate. Note that the minimum drop rate is not always 0%; this is due to the fact that the requests and their deadlines generated by our random processes cannot always be serviced (i.e., there may be requests that cannot be accommodated in any schedule). The results of using the online scheduler and the design trace on the initial configuration output are denoted by "Initial + Online Sch.". Switching to the online scheduler that has to schedule the design trace requests as they come, increases significantly the OPEX (and, hence, the total cost) and the drop ratio, as can be seen in the figure.

We compare these results with the performance of applying algorithm CA on the initial configuration, using the greedy online scheduler on the design trace, i.e., "CA + Online Sch.". Observe that, while the total cost is similar to the cost incurred by the initial configuration, the drop ratio achieved is very close to the lower bound. This means that by investing more in CAPEX by buying more RSU capacity, algorithm CA compensates for this cost increase by reducing OPEX by a similar amount, while almost completely achieving its main goal, i.e., the reduction of drop ratio as much as possible.

While the previous results are encouraging for the practicality of the CA algorithm, the more important test is clearly running CA on the 9 traces that were not used in the design phase. These 9 traces were generated with the same statistics as the design trace. The averaged results of running the online scheduler with the initial configuration, and with the CA configuration are shown in Figure 1 denoted by "Initial + Online Sch. (New Traces)" and "CA + Online Sch. (New Traces)", respectively. For comparison purposes, we have added results when we maximize the capacities of all opened RSUs, i.e., the "Max. Cap. + Online Sch. (New Traces)" in Figure 1). These results correspond to the case of using as much capacity as possible in order to achieve the best possible drop ratio, while being oblivious to any cost increases.

As expected, running the three different configurations (Initial, CA, and Max. Cap.) on the 9 new traces incurs larger total costs than running Initial and CA on the design trace (Figure 1a). Obviously, the increase for the Initial and CA configurations is due to increased OPEX costs, while the high total cost of max capacity reflects its high CAPEX cost. Note though, that the discrepancy between the latter and the costs of the Initial and CA configurations decreases as the CAPEX cost becomes more dominant in the total cost (i.e., the single RSU capital cost factor increases). This is due to the fact that as the CAPEX contribution to the total cost increases, the initial configuration opens fewer RSUs and equips them with more (up to max) capacity. Nevertheless, the cost incurred by maximizing all capacities is always significantly greater than using the Initial or CA configurations. On the other hand, its drop ratio is very close to the lower bound (Figure 1b). Therefore, if one is oblivious to costs, maximizing all capacities will give the best drop ratio. If the total cost is a consideration, then Figure 1b shows that the CA configuration has a much smaller drop ratio than the Initial configuration, while incurring almost the same (actually smaller) cost as the Initial configuration, as seen in Figure 1a.

To better understand the performance of the CA algorithm, we show the performance of the configuration resulting after each one of its iterations in Figure 2, for the case of the single RSU capital cost factor of one (the leftmost points in Figure 1). The total cost and its components, i.e., OPEX and CAPEX,

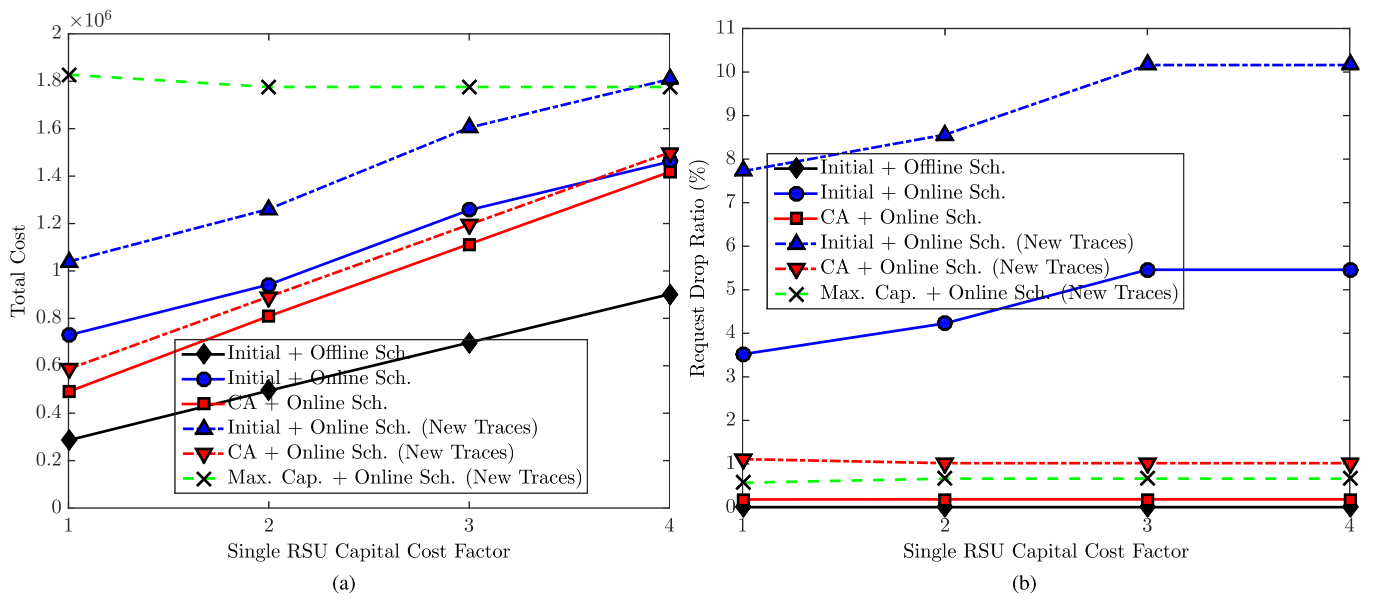


Fig. 1. The Effect of Capacity Augmentation Algorithm on RSU Placement Scheme.

are shown in Figures 2a and 2b, respectively. Figure 2c shows the request drop ratio for each iteration. The results for the other single RSU capital cost factors is similar, and, therefore, we concentrate on the case of single RSU capital cost factor of one.

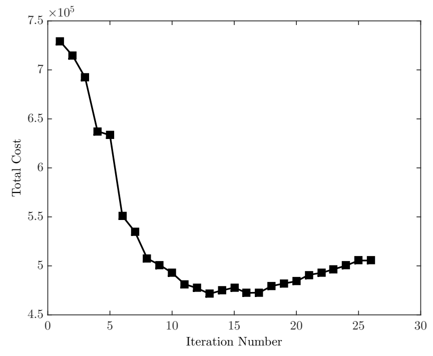
As can be seen in Figure 2, there are two phases in the graphs. The first phase (up to iteration 9) corresponds to a sharp decrease in the drop ratio. During this phase, by adding capacity to those RSUs with the highest impact on the drop ratio, algorithm CA creates opportunities to serve more vehicles at their favourable positions relative to RSUs (from an energy point of view), while, at the same time, it decreases the contention between requests for service. This causes both the OPEX and the drop ratio to decrease. On the other hand, increasing RSU capacities increases CAPEX. Hence, after a certain point (iteration 9), increasing capacities doesn't affect OPEX by much, while CAPEX continues to increase, and as a result, the total cost is increasing. During this second phase, the improvement of the drop ratio is slow, especially when compared to its rapid drop during the first phase. This is to be expected, since after a certain point the capacities of the RSUs are no longer an issue, and increasing them does not improve significantly the OPEX or the drop ratio.

In all previous experiments, the RSUs were chosen from three types, with a maximum capacity of 6. Given that vehicular networks are built with an operational horizon measured in decades, in the future it may be possible to increase RSU capacities much beyond the upper bound of 6. In order to assess the performance of algorithm CA in this case, we run it with the same CAPEX model as before, but without an upper bound on the available capacities. The results are shown in Figure 3. The curve "CA (Unlimited Cap.) + Online Sch. (New Traces)" shows the results of running algorithm CA without RSU capacity upper bounds. The other curves are the result of the Initial and CA configurations from Figure 1, where RSUs have a maximum capacity of 6. In these experiments, the highest capacities reached by algorithm CA are 8, 9, 9, and 9 for single RSU capital cost factors $1, 2, 3, 4$ respectively. Note that these maximum capacities are not much higher from the upper bound of 6 used before.

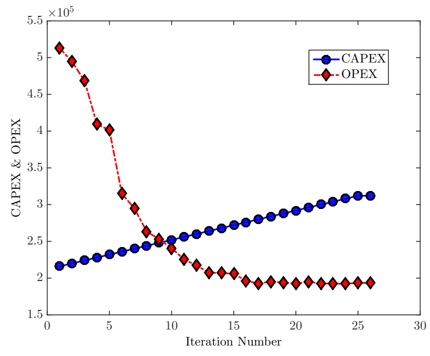
We note that algorithm CA has a higher drop ratio when run with unlimited capacities than with limited capacities (although the total costs have the reverse relationship). This can be explained as follows: First, reaching the capacity upper bound on an RSU forces the algorithm to distribute its capacity increases to other RSUs in order to decrease the overall drop ratio, and as a result, this distribution of extra capacity eventually helps to service more requests within their deadlines. Second, there is the pathological situation of a vehicle generating more requests than can be serviced before the vehicle leaves the servicing RSU coverage area; therefore, an RSU can have the highest impact on the request drop ratio and, at the same time, increasing its capacity does not reduce the number of dropped requests. This causes algorithm CA to focus on the wrong RSU and to continuously increase its capacity, until it detects that the drop ratio has not improved (the length of window M in Algorithm 1) and terminates.

VI. Conclusion

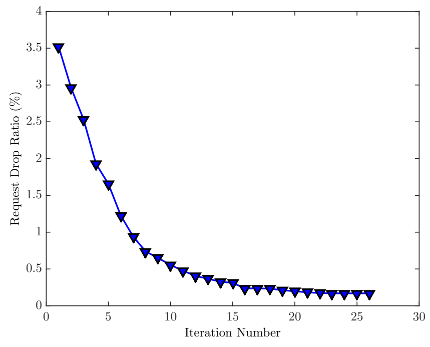
This paper has addressed the issue of capacity augmentation in energy efficient road-side unit (RSU) deployments. The objective is to find RSU radio capacity augmentation assignments that minimize the total capital expenditure and long-term operating expenditure costs. This is subject to meeting packet deadline constraints with a given packet loss rate target. An algorithm, referred to as the capacity augmentation (CA) algorithm, was proposed that iterates over the RSUs, selecting candidates for capacity augmentation based on their packet



(a)

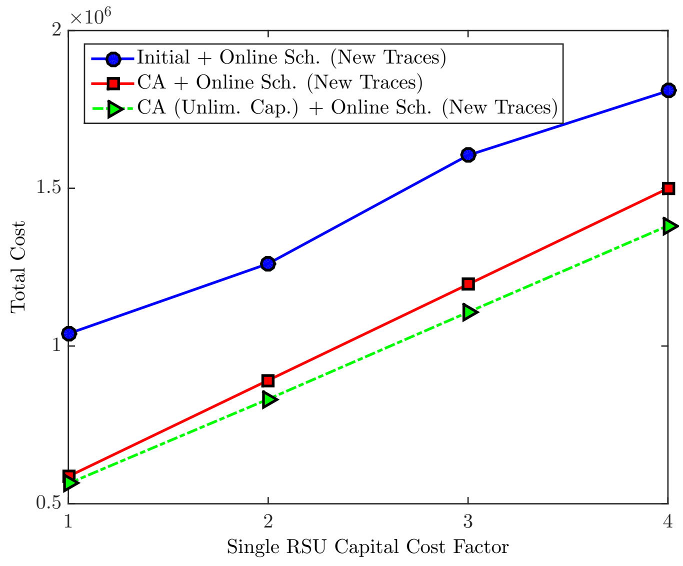


(b)

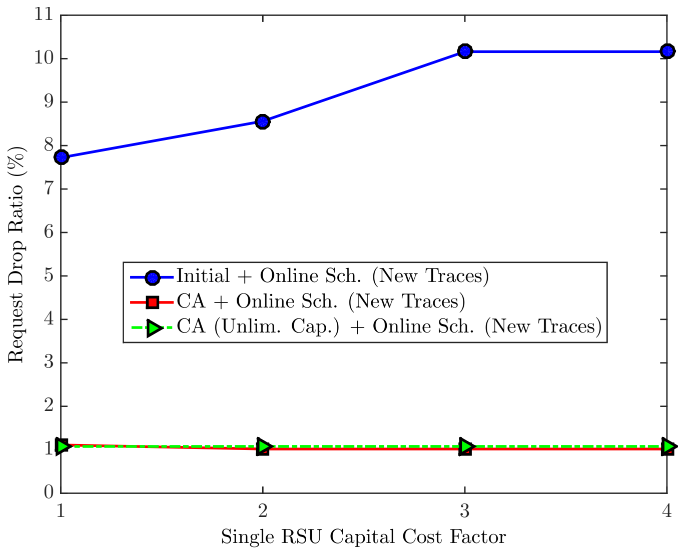


(c)

Fig. 2. The Capacity Augmentation Algorithm Progress in each of Iteration.



(a)



(b)

Fig. 3. The Effect of Unlimited Capacity on the Capacity Augmentation Algorithm Performance.

loss rate sensitivities. Results were presented that characterize and compare the performance of the CA Algorithm using a greedy online packet scheduler. It was shown that the CA Algorithm is an efficient way to assign RSU radio capacity that can achieve the desired packet loss rate target while reducing the sum of operating and capital expenditure costs.

REFERENCES

[1] A. Farbod and T. Todd, "Resource Allocation and Outage Control for Solar-Powered WLAN Mesh Networks," *IEEE Trans. on Mobile Computing*, vol. 6, no. 8, pp. 960–970, Aug 2007.

[2] G. Badawy, A. Sayegh, and T. Todd, "Energy Provisioning in Solar-Powered Wireless Mesh Networks," *IEEE Trans. on Vehicular Technology*, vol. 59, no. 8, pp. 3859–3871, Oct 2010.

[3] N. Nikookaran, T. D. Todd, and G. Karakostas, "Combining capital and operating expenditure costs in vehicular roadside unit placement," *Submitted*, 2016.

[4] U. Ashraf, "Capacity Augmentation in Wireless Mesh Networks," *IEEE Trans. on Mobile Computing*, vol. 14, no. 7, pp. 1344–1354, July 2015.

[5] F. Ahdi and S. Subramaniam, "Capacity Enhancement of RF Wireless Mesh Networks Through FSO Links," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 8, no. 7, pp. 495–506, July 2016.

[6] F. Y. S. Lin, "Link Set Capacity Augmentation Algorithms for Networks Supporting SMDS," in *IEEE ICC '94*, May 1994, pp. 624–629 vol.1.

[7] Y. Liang, H. Liu, and D. Rajan, "Optimal Placement and Configuration of Roadside Units in Vehicular Networks," in *IEEE VTC*, 2012, pp. 1–6.

[8] T. S. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed. Prentice Hall, 2001.

[9] J. Song, Y. Wu, Z. Xu, and X. Lin, "Research on Car-Following Model Based on SUMO," in *IEEE 7th International Conference on Advanced Infocomm Technology (ICAIT)*, Nov 2014, pp. 47–55.

[10] M. Mahdian, Y. Ye, and J. Zhang, "Approximation algorithms for metric facility location problems," *SIAM Journal on Computing*, vol. 36, no. 2, pp. 411–432, 2006.

[11] K. Holmberg, "Solving the Staircase Cost Facility Location Problem with Decomposition and Piecewise Linearization," *European Journal of Operational Research*, vol. 75, no. 1, pp. 41 – 61, 1994.

[12] J. A. Volpe, "Vehicle-Infrastructure Integration (VII) Initiative Benefit-Cost Analysis, Version 2.3," United States Department of Transportation, Washington, DC, Tech. Rep., May 2008.

[13] T. Kumrai, K. Ota, M. Dong, and P. Champrasert, "RSU Placement Optimization in Vehicular Participatory Sensing Networks," in *IEEE INFOCOM Workshop*, April 2014, pp. 207–208.