Ring Optimal Assignment of Slot Reservations for Cyclic TDMA Protocols

Marc Mosko and Ignacio Solis
Palo Alto Research Center
Palo Alto, CA 94304, USA
{mmosko,isolis}@parc.com

J.J. Garcia-Luna-Aceves
Baskin School of Engineering
UC Santa Cruz
Santa Cruz, CA 95061, USA
jj@soe.ucsc.edu

***Abstract*—We present an algorithm to assign time slots to nodes in a TDMA network that minimizes the jitter in time slot assignments. By reducing the jitter in time slots around the TDMA frame, we can provide more consistent network access and reduce the overall delay seen by an application with time-varying traffic patterns, such as normal web traffic. Our algorithm can reduce the average delay seen by all nodes in a network by up to 51%, based on our numerical analysis. The algorithm is designed for TDMA MAC layers where a node has the potential to reserve some number of time slots out of a larger selection of available slots, and one wishes to choose the slots that most evenly distribute them around the TDMA ring. The algorithm solves the Minimum Variance Placement problem for the special case of a directed ring. After describing an exact solution using dynamic programming, we present a much faster run time heuristic that closely approximates the exact solution.**

I. Introduction

This paper presents methods to reduce the jitter in wireless TDMA networks. Wireless TDMA networks are used in cellular systems, military systems, and in wireless backhaul networks, and in some peer-to-peer systems. We are concerned primarily with distributed peer-to-peer systems that do not use a central controller and present a distributed algorithm to smooth out time slot assignments over a TDMA frame. By smoothing out the time slot assignments over a frame and not letting them bunch up, we can reduce the delay for variable bit rate traffic, such as normal TCP/IP web traffic that arrives at seemingly random times in a TDMA frame. When traffic arrives at unpredictable times, the best one can do is to evenly spread out network access over the frame.

In wireless computer data networks, Medium Access Control (MAC) protocols can be divided in to contention-based and schedule-based. Contention protocol, such as IEEE 802.11 CSMA/CA [1], are widely used but have poor support for real-time data traffic, especially without an access point. They also suffer poor performance under high load. Prior work on scheduled MAC protocols can be classified as topology independent and topology dependent. Topology-independent scheduled protocols are typically pre-assigned Time Division Multiple Access (TDMA) schemes where each node is assigned one or more unique time slots in a global schedule. These types of protocols suffer from channel under-utilization an also require global knowledge of the network. Topology dependent scheduled protocols, also called dynamic TDMA protocols, construct local TDMA schedules for small groups of nodes using neighborhood topology information, thus solving the under-utilization problem of TDMA protocols. Previous dynamic TDMA protocols, such as NAMA [2], offer good performance at high load, but still have problems with real-time data traffic because their randomized slot assignment algorithms do not have bounds on inter-slot time intervals. This paper presents a reservation algorithm to compliment dynamic election TDMA MAC layers.

Once a node knows from a dynamic TDMA slot assignment scheme in which time slots it may transmit, a node may reserve zero or more of those time slots. A reservation means that a node may continually use the same time slot within a frame without contending for an election each frame. Because reservations repeat over multiple frames in the same time slot, the reservation problem may be visualized as an assignment problem around a directed ring. Our reservation algorithm, known as Ring Optimal Assignment Reservations (ROAR), finds optimal allocations of reservations to optimize some characteristic of a traffic flow. In this paper, we look at the problem of finding the reservations that minimize the inter-slot jitter, which is the same as minimizing the variance of the slot distances around the ring. The desire of a node to reserve a time slot, and a reservation conflict resolution mechanism, are protocol-dependent signaling mechanisms not discussed here. The Context Aware Scheduling Algorithm (CASA) MAC protocol [3], is an example of a MAC protocol that implements topology and reservation signaling sufficient to realize the ROAR algorithm.

The CASA protocols use a different slot allocation mechanism than CASA. It uses a permutation of time slots per node. The permutation ordering means that some slots get a high likelihood of being assigned to the node because they appear towards the front of the permutation and other slots are unlikely to be assigned because they appear towards the end of the permutation. This spreads the likelihood of assignment more evenly over nodes and reduces the high-variance of the NAMA method.

The paper is organized in the following sections. Section II describes the scheduling and reservation problems and introduces our notation. Section III describes a reservation scheme to minimize overall variance of slot reservations around a ring. Section IV presents the results of numerical analysis of the

ROAR algorithm. Section V concludes the paper.

II. Scheduling Overview

In our TDMA model, time is divided in to frames and frames in to slots. Let there be S slots per frame. Because TDMA frames are cyclic, we may view time slots as vertices on a ring each a unit distance apart. Our convention is that on a ring of integer circumference S, there are nodes labeled in $[1, S]$ with directed arcs in clockwise rotation. The distance metric is the arc distance between nodes. Using notation similar to Manku [5, p.14], we have a distance metric δ_{clk} around the directed arcs of a ring and δ_{abs} as the shortest-path distance around the surface of a circle. The metric δ_{clk} measures distance like a clock, where one may only travel one direction around the ring. The metric δ_{abs} measures the distance going either way around the circle.

$$\delta_{clk}(u,v,n) = \begin{cases} v-u & v \geq u \\ n+v-u & \text{otherwise} \end{cases} \qquad (1)$$

$$\delta_{abs}(u,v,n) = \begin{cases} \min\{v-u, n+u-v\} & v \geq u \\ \min\{u-v, n+v-u\} & \text{otherwise} \end{cases} \qquad (2)$$

The triangle inequality property of a distance metric affects the complexity of algorithms. The inequality, for our purposes, states that given any three points forming a triangle, the sum of the lengths of any two sides is no less than the the length of the remaining side. This property holds for δ_{abs} but does not hold for δ_{clk} . To see that it does not hold for δ_{clk} , imagine a clock face with hour marks. Take the three hour points $1, 2, 3$. The time-distance from 1 o'clock to 2 o'clock to 3 o'clock is 2 hours. The time-distance from 3 o'clock to 1 o'clock is 10 hours. Therefore, $\delta_{clk}(1,2,12)+\delta_{clk}(2,3,12) < \delta_{clk}(3,1,12)$ in violation of the triangle inequality.

III. Reservation Algorithm for Minimum Variance

The Ring Optimal Assignment Reservations by Variance (ROAR-V) algorithm allocates reservations over a frame to smooth out slot assignment by minimizing the variance of distances between reservation slots. Our problem may be formulated as follows: For a given frame, the node has an existing reservation set R and newly assigned set of slots W, with $V = R \cup W$. Each slot in V has a maximum capacity of C, begin the number of load units transmittable during a single slot. Find the subset V' of V, where $|V'| \leq K$, where K is the maximum reservable slots by the node and V' optimizes the objective. We wish to find V' such that inter-reservation slot variance is minimized. That is, find the K slots in V that most evenly divide the frame, assuming cyclic reservations. The algorithms in ROAR-V address this problem. ROAR-V is particularly suited for random, unpredictable traffic flows. In a network with some predictable traffic and some unpredictable traffic, a hybrid algorithm would be called for.

The ROAR-V problem is similar to several previously studied problems. The k-variance problem is to find a subset of k points in an n-node graph with minimum variance. The variance is the sum of squared Euclidean distances between all pairs of n nodes divided by k. There exists an $O(k^2 n + n \log n)$ algorithm [6], where n is the total number of nodes from which one must choose k. The Minimum Variance Placement (MVP) problem for facility location, which is shown to be NP-hard [7] for general distance metrics, is also similar to ROAR-V except it minimizes the all-pairs variance. MVP does not require that distances satisfy the triangle inequality. The authors show that not only is MVP NP-hard, even obtaining a relative time approximation is NP-hard if the distances do not satisfy the triangle inequality [7, Prop. 1.2].

A. ROAR-V

This section presents an exact solution to the ROAR-V problem using dynamic programming and a heuristic based on rotating rings that has much faster run time. Given a set of $\mathbf{V}$ integer positions around a ring of integer circumference S, find the set $\mathbf{V}' \subseteq \mathbf{V}$ of size K that most evenly divides the ring. This problem may be visulized as in Fig. 1, where each S positions are as hour marks on a clock, and the set $\mathbf{V}$ are markers on some hours. Let the set $\mathbf{D}_S$ be the minimum distances of adjacent elements of set $\mathbf{S}$. We wish to find a set $\mathbf{V}'$ of cardinality K such that the variance $\text{var}(\mathbf{D}_{V'})$ is minimum. Note that all solutions will have the same mean $\mu = S/K$ because of the circular structure.

If one considers δ_{clk} , the ROAR-V problem is very hard because ring distances do not satisfy the triangle inequality. However, minimizing the variance of nearest neighbor distances around a circle using δ_{abs} yields an equivalent solution in this special case of minimizing variance on a ring. Solving ROAR-V using δ_{abs} between nearest neighbors may be solved exactly in time $O(n^2 k)$ using dynamic programming. The algorithm $PickBest$ in Alg. 1 first assumes that some element i in $\mathbf{V}$ is in the solution. With this assumption, it calls the algorithm $FillTable$ in Alg. 2 to determine the set of K choices that minimize the sum of squared distances around a circle, ending again at element i. $PickBest$ then iterates over all n starting positions. $FillTable$ works on the interval distances δ_{abs} between two elements. In the algorithm $FillTable$, each row represents the best solution assuming row elements have been chosen in addition to the implicit starting element determined by the rotation in $PickBest$. Each column represents the best solution assuming that element col is in the last element in the solution. Because we must close the circle, we calculate $FillTable$ to K rows, which is actually a solution with $K+1$ elements because the starting element is counted twice.

In $FillTable$, row 1 is initialized to the sum of squared distances from the chosen starting element to the end of interval col. Because in row 1, we assume exactly 1 element chosen in the solution in addition to the implicit starting element, this row may be calculated as the initial condition. Rows $2 \ldots K$ are filled in by picking the element from $row-1$ where the sum of squared distances between the $row-1$ solution and the current column is minimized. Once we have filled in the table up to element $m[K, n]$, we may read back the solution from element $m[K, n]$. Each element in the matrix m is an ordered pair (ss, p), where ss is the sum of squared

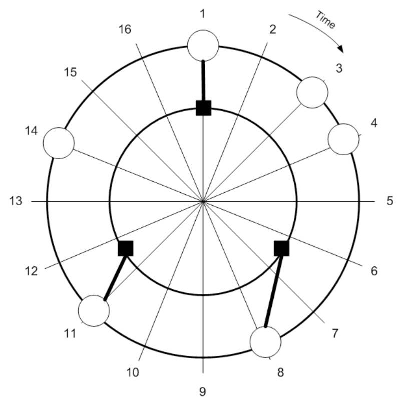


Fig. 1. Ring Fitting (suboptimal)

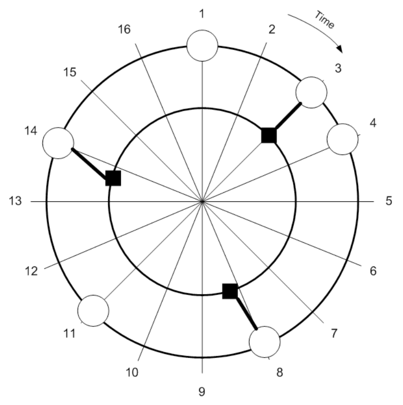


Fig. 2. Ring Fitting (optimal)

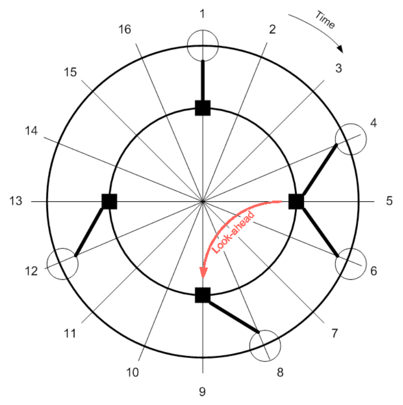


Fig. 3. 1-step look ahead

distances and p is the traceback path to allow reading the solution. The element p will be an ordered set $(c_1, c_2, \dots, c_K)$, where c_i is the column index. Reading back the solution in $PickBest$ converts from column index to vertex number, accounting for the rotation of **IV**.

Algorithm $PickBest$, while polynomial in time, is not practical for a scheduling algorithm because of the $O(n^2 k)$ time bound. A network node, for example, could expect to have $n >> 100$ time slots per frame and $k >> 10$, which would be on the order of 10's of millions of calculations every few hundred milli seconds, which would require several GHz of processing speed. Therefore, we present a heuristic algorithm, called ROAR-VR, that approximates the solutions found by $PickBest$ by rotating a minimum variance ring around the possible slots. The heuristic runs in time $O(n\ k)$. The above example would be on the order of 10's of thousands of calculations every few hundred milli seconds.

B. ROAR-VR Heuristic

The ROAR-VR heuristic operates by fitting a minimum variance ring to the possible time slots. Fig. 1 shows an example where $n = 6$ on a 16-slot frame and $K = 3$. The algorithm begins by exactly lining up the ring with one time slot, such as slot 1. For a minimum variance ring, the mean inter-slot distance is $\mu = 5\frac{1}{3}$, so the ring locations are at $u = (1, 6\frac{1}{3}, 11\frac{2}{3})$. for each u_i, we pick the nearest slot in V. In this example, the chosen slots would be $v = (1, 8, 11)$. The variance from the mean would be $\sigma^2 = 3.\bar{2}$. In the next step of the algorithm, the ring slots would be at $u = (3, 8\frac{1}{3}, 13\frac{2}{3})$, which would result in the chosen slots of $v = (3, 8, 14)$. The variance from the mean would be $\sigma^2 = 0.\bar{2}$, which in this case is the minimum possible variance so this is an optimal choice.

In some cases, there is ambiguity as to which slot to pick. It may be that there are exactly 1 or exactly 2 nearest neighbors to a fitted ring slot. For example, in Fig. 3, the match for position 5 is equidistant from slots at positions 4 and 6. The algorithm then looks to what would be the next match, at position 9, and picks the slot that minimizes the sum of squared distance. In this case, picking slot 4 would have a sum of squared distance of $9 + 16 = 25$ and picking slot 6 would have a sum of squared distance of $25 + 4 = 29$. Therefore, the algorithm would pick slot 4 in this case. It may be that the one-step look ahead is also ambiguous. In this case, to avoid combinatorial time complexity, the algorithm picks one look ahead site at random. If in this example it were to pick slot 8, it would proceed as above. If it were to pick slot 10, then then the sum of squared distances would pick slot 6, a sub-optimal choice in the long-run. The look-ahead slot choice does not determine the actual slot choice in the next iteration of the algorithm. One may, of course, extend the one-step look ahead to multiple steps, each improving the decision made at the expense of running time.

Alg. 3 presents our heuristic for solving the ROAR-V problem. It returns the subset of slots in $\mathbf{V}$ of size K that minimize the inter-slot variance. The rotating ring method selects one slot in $\mathbf{V}$ as the start position, then adds N/K to it to find the target position with zero error from the mean. This is a rational number, not an integer slot number. Alg. 3 then finds the one or two slots in $\mathbf{V}$ that are closest to that target, as those are the slots that minimize the error from the mean. The algorithm repeats for all $K - 1$ target locations equally spaced N/K apart, and calculates the variance of the solution. It then picks the next slot in $\mathbf{V}$ as the starting position, and repeats, keeping track of the minimum variance solution

The rotating ring has a starting position i and matching positions $i + kd$, where $k = 1..K - 1$. For a given starting position, it calls Alg. 5 to determine the best vertex to pair with the next matching position. $FindNext$ returns a set of $\{\emptyset\}$ or $\{i_1\}$ or $\{i_1, i_2\}$, where i_j is an index position in $\mathbf{V}$. If the return set is empty, the configuration under examination is invalid; this typically happens when there are not enough vertices remaining to find K of them. If it returns one element, then that is the unique best match for the target. If it returns two elements, then there are two equidistant matches from the target. In the case of two equidistant matches for the target, if the current target is the last of K, then we pick the choice with minimum variance around the whole ring. If the current target is somewhere in the middle of the solution, we pick the choice that would minimize the sum of squared distances of the intervals around the current choice. Whenever there are equidistant choices for the one-step lookahead, we pick one at random.

Algorithm 1:

```
PICKBEST($\mathbf{V}, K, S$)
(1)  Convert $\mathbf{V}$ to intervals $\mathbf{IV}$ on $S$
(2)  $IV[i] \leftarrow \delta_{abs}(V[i], V[(i \mod d) + 1], S)$
(3)  $n \leftarrow |\mathbf{V}|$
(4)  $best \leftarrow (\infty, (\emptyset))$
(5)  for $i = 0$ to $n - 1$
(6)      $rot \leftarrow$ Rotate $V$ by $i$
(7)      $temp \leftarrow$ FILLTABLE($rot, K$)
(8)      $last$ will be the final element $(ss, p)$ that closes circle
(9)      $last \leftarrow temp[K, n]$
(10)     if $last.ss < best.ss$
(11)         Dereference path $last.p$ to actual values in $\mathbf{V}$
(12)         $Vout : Vector[1 \dots K]$
(13)         $Vout[j] \leftarrow V[((last.p[j] + i) \mod n) + 1]$
(14)         $best \leftarrow (last.ss, Vout)$
(15) return $best.p$ return $best$
```

Algorithm 2:

```
FILLTABLE($\mathbf{IV}, K$)
(1)  $n \leftarrow |IV|$
(2)  $m$ is matrix of ordered pairs $(ss, p)$, $p$ is a path
(3)  $m : Matrix[1 \dots K, 1..n]$
(4)  Initialize first row to sum of square distances
(5)  $m[1, j] \leftarrow \left( \left[ \sum_{i=1}^{j} IV[i] \right]^2, (\emptyset) \right)$
(6)  for $row = 2$ to $K$
(7)      for $col = row$ to $n$
(8)          $e$ is ordered pair $(ss, last)$ element in $m$
(9)          $e \leftarrow (\infty, \{0\})$
(10)         for $k = 1$ to $col - 1$
(11)             $dist \leftarrow \sum_{i=1}^{col-1} IV[i]$
(12)             $x \leftarrow m[row - 1, k].ss + dist^2$
(13)             if $x < e.ss$
(14)                 $e \leftarrow (x, m[row - 1, k].last \cup k)$
(15)         $m[row, col] \leftarrow e$
(16) return $m$
```

IV. Numerical Analysis

We analyze ROAR-V via monte carlo simulation, and compare it against not making any reservations ("NoRes") and a greedy reservation scheme ("Greedy"). The NoRes system uses only the randomly assigned time slots picked by the slot allocator. The Greedy scheme will reserve a time slot if the node has a traffic backlog at an assigned slot and will release a reservation if there is no longer a traffic backlog. The simulator is a custom C++ program implementing the slot assignment algorithm, the reservation algorithm, and a FCFS packet scheduler. The simulator uses a "unit load", where up to 5 units may be transmitted per time slot. The modeled network is a fully connected mesh and all traffic is only single hop.

We use two time slot allocators: one similar to NAMA and one similar to CASA [3]. The NAMA allocator uses a uniformly random choice of node per time slot, so the inter-slot allocations for a node are geometrically distributed. The CASA allocator randomly permutes a time slot vector for each node and assigns time slots based on the permutation order, so it has a much tighter inter-slot distribution than geometric.

There are two traffic loads. The Clustered traffic load picks random nodes and random time slots to allocate load, and those choices are fixed for all frames in a simulation run. This models CBR traffic. The Bursty traffic load is like Clustered, but the allocation changes from frame to frame, which results in an unpredictable pattern. It models random bulk traffic. We run the network at 70% utilization, so there are always $0.7 \cdot slots * 5$ units of load in the network. Our hypothesis going in to the experiment was that Greedy would do best with the Clustered traffic and ROAR-V would do best with the Bursty traffic.

The reservation algorithms work as follows. The NoRes algorithm does not make any reservations. The Greedy algorithm operates slot-by-slot reserving or releasing slots based on the instantaneous backlog at that slot time. The ROAR-V algorithm operates frame-by-frame reserving or releasing slots based on backlog from the previous frame and the expected traffic arrivals from the previous frame. The Greedy and ROAR-V algorithms allows a node to have up to $maxResPerNode = \lfloor slots/2 \rfloor + 1$ maximum reservations, allocating no more than $maxResPerFrame = \lfloor maxResPerNode/5 \rfloor + 1$ reservations per frame. For example, with 100 nodes and 300 slots, $maxResPerNode = 151$ and $maxResPerFrame = 31$. A given node may thus have, at most 31 reservations the first frame, then an additional 31 in the second frame. If the node only reserved, say 10 slots in the first frame, it may have up to 41 in the second frame. To reserve a slot, a node must both have the capacity via $maxResPerFrame$ and $maxResPerNode$ and be assigned unreserved slots by the slot allocator. A node may only choose reservations from the slots allocated in a frame. The idea behind $maxResPerFrame$ is to prevent a few nodes from grabbing all the reservable time slots at once. For the Greedy algorithm, if a node has a backlog and an available reservation, it will reserve a timeslot. If there is no backlog at a reserved time slot, it will release the reservation. For the ROAR-V algorithm, the same maximums apply, $maxResPerNode$ and $maxResPerFrame$, but the way slots are allocated and released is different than Greedy. If there is no backlog at the start of a frame, the maximum reservations does not increase. If there is no backlog and no new work to do (new arrivals), the maximum number of reservations decreases by $maxResPerFrame$, down to zero. Reservations are allocated in a block, via the ROAR-V algorithm, up to the maximum permissible in the frame. Once the ROAR-V algorithm is run, if the variance of the allocation is less than the current allocation, a node will modify its reservations by releasing previously reserved slots not in the result set and reserving new slots in the result set.

Simulations are run with 50, 100, and 200 fully-connected nodes with 300 time slots/frame. The experiment is run for 100 frames, and repeated for 10 trials with different random number seeds and traffic loads. The figures present the average delay over all 10 trials. The 95% confidence interval, while not shown, is typically under between 3% - 5% of the mean for the Bursty traffic and 12% - 20% of the mean for Clustered traffic.

In Figs 7 – 8, all but one result are statistically significant (there are no overlapping 95% confidence intervals) within each group (i.e. 50-node, 100-node, or 200-node). The one result with overlapping intervals is Fig 8, 200 nodes between ResNone and Greedy. In Figs 9 – 10, all but one result are statistically equivalent (the 95% confidence intervals all overlap) within each group. The one exception is Fig 10, 50-node between Greedy and ROAR-V, where Greedy is significantly better than ROAR-V.

Looking at the results, and considering the statistical significance of the 95% confidence intervals, we see there is only a significant difference between reservation algorithms for Bursty traffic. For Clustered traffic, the means are all fairly close and the 95% confidence intervals overlap. At times, Greedy performs better, on average, than no reservations (e.g. Fig 10), but sometimes no reservations works better than Greedy (e.g. Fig 9, 100 nodes), when Greedy makes a sub-optimal allocation. Sometimes ROAR-V performs better on clustered traffic, even though its reservation scheme does not depend on the traffic arrival times. We would conclude form this that none of the schemes sufficiently optimize for clustered traffic beyond the randomization of the slot allocator. For Bursty traffic, there is a significant difference between reservation algorithms. The Greedy algorithm reserves slots when there is a burst, so a node with a backlog gets more transmission opportunities and clears its backlog sooner than without reservations (NoRes). However, these reservation may be bunched up in one part of the frame, so overall the delay is not optimized. The ROAR-V reservation scheme, because it evenly spreads reservation slots over a frame, greatly reduces delay compared to Greedy.

Numerically, Table I shows the percentage improvement of ROAR-V over Greeding and NoRes for the 6 cases of Bursty traffic using the data presented in Figs 7 – 10. Compared to not using reservations, ROAR-V has between 44% and 51% lower delay and compared to a Greedy reservation scheme, ROAR-V has between 29% and 51% lower delay.

TABLE I
PERFORMANCE IMPROVEMENT OF ROAR-V

Scenario	vs. Greedy	vs. NoRes
CASA Bursty 50-node	29.7%	45.4%
NAMA Bursty 50-node	30.6%	44.8&
CASA Bursty 100-node	46.0%	50.1%
NAMA Bursty 100-node	46.4%	50.8%
CASA Bursty 200-node	51.3%	44.3%
NAMA Bursty 200-node	51.3%	51.8%

Fig. 5 shows the standard deviation of the inter-slot times for NAMA and CASA in executions with 300 slots, averaged over 200 frames with 10 independent trails. The results shown are for NoRes, but this aspect of the system is independent of the reservation scheme. As described in [3], the CASA scheduler has a significantly tighter schedule generation than NAMA, with the difference becoming more pronounced as the node to slot ratio grows (as the number of nodes to the number of slots becomes larger). All the differences in the figure are significant, the 95% confidence interval is under 1.0 in all cases. The improvement in inter-slot jitter ranges between 8% for 50-nodes to 18% for 200 nodes.

In terms of average traffic delay, there are a few specific cases where the CASA scheduler is significantly better than the NAMA scheduler, then all other cases are statistically equivalent. Fig. 6 shows all 6 comparisons between NAMA and CASA for Clustered and Bursty traffic, with Greedy, NoRes, and ROAR-V reservation techniques. The X-axis is a category label, such as "G-50" meaning "Greedy 50-node". The abbreviation "NR" is for NoRes and "V" for ROAR-V. For Bursty traffic with ROAR-V, the NAMA and CASA lines overlap, so there is only one line visible. CASA has a statistically significant lower total delay in the 100-node and 200-node runs with Bursty traffic NoRes (the top middle pair of lines in the figure). In all other direct comparisons, CASA and NAMA have similar delays, due to large confidence intervals for the Clustered data and very close results for the Bursty data. The standard deviation of the average delay is generally slightly lower for CASA, the largest difference being for NoRes and Bursty traffic (about 2x to 3x less).

Fig. 4 compares the performance of the heuristic ROAR-VR to the dynamic programming solution ROAR-V in terms of running time. The tests were run on a 3.2 GHz Quad-Core MacPro using the same C++ code as the simulator. We ran experiments with 5,10, 20, 40, 80, and 160 nodes with 1000 time slots for 20 frames. This resulted in, on average, 6.25, 12.5, 25, 50, 100, and 200 time slots being assigned to each node. The figure shows the total execution time divided by the number of slots divided by the number of nodes. The ROAR-VR line grows at a rate between 2.6x and 4.2x for each doubling of the number of slots. On average it grows around $n^{1.56}$, where n is the number of slots (the size of the $\mathbf{V}$ vector). The ROAR-V series grows between 6.5x and 12.7x for each doubling of n, scaling on average like $n^{4.46}$.

V. CONCLUSION

We have presented the ROAR-V algorithm solved via dynamic programming and as a fast run time heuristic. The ROAR-V algorithm solves the Minimum Variance Placement problem for facilities constrained to a ring topology, and is applicable to dynamic TDMA networks where nodes may reserve some subset of their assigned slots. The ROAR-V algorithm has between 29% - 51% lower delay than a Greedy reservation scheme and between 44% - 51% lower delay than not using reservations at all. We also compared the NAMA and CASA slot allocators (election schemes), and found that in terms of average delay, CASA has a slight improvement in a few cases where reservations are not used. In terms of variation of time-slot assignments, CASA has a significant 8% to 18% improvement in inter-slot jitter over NAMA.

Future work should look at adapting ROAR-V to be aware of the incurred delay, and pick different rotations of the minimum variance schedule based on recurrent traffic loads. We have also worked on a minimum latency algorithm, known as ROAR-L, which we will present at a later time.

Algorithm 3:

```
ROTATERING($V, K, N$)
(1)  $d \leftarrow |V|$
(2)  $u \leftarrow N/K$
(3)  These track the best solution (lowest variance)
(4)  $bestv \leftarrow \infty$
(5)  $bestsol : Vector[1 \ldots K]$
(6)  for $i = 1$ to $d$
(7)     This is the solution being tested and its starting slot
(8)     $sol : Vector[1 \ldots K]$
(9)     $start \leftarrow V[i]$
(10)    $target$ is the next ring position at min variance
(11)    $target \leftarrow start + u$
(12)    $idx$ is the current index in $sol$
(13)    $idx \leftarrow 1$
(14)    $sol[idx] \leftarrow i$
(15)    $bestidx$ is the best next index into $V$
(16)    $bestidx \leftarrow i$
(17)    while $idx < K$
(18)       $pos$ is the index in to $V$ to test for a fit to $target$
(19)       $pos \leftarrow (bestidx \mod d) + 1$
(20)       $bestidx \leftarrow 0$
(21)       $x \leftarrow$ FINDNEXT($V, d, sol, pos, idx, target, K, N$)
(22)       if $x = \emptyset$
(23)          This is invalid configuration, stop checking.
(24)          break
(25)       if $|x| = 1$
(26)          Exactly 1 best match.
(27)          $bestidx \leftarrow x[1]$
(28)       else
(29)          There are 2 best matches.
(30)          $bestidx \leftarrow$ PICKFROMTWO($V, idx, target, u, K, x$)
(31)          $idx \leftarrow idx + 1$
(32)          $sol[idx] \leftarrow bestidx$
(33)          $target \leftarrow target + u$
(34)       if $bestidx = 0$
(35)          $v \leftarrow \infty$
(36)       else
(37)          Compute the variance of the current solution
(38)          $v \leftarrow \mathrm{var}(sol)$
(39)       if $v < bestv$
(40)          $bestv \leftarrow v$
(41)          $bestsol \leftarrow sol$
(42)    return $sol$
```

Algorithm 4:

```
FINDNEXT($V, d, sol, pos, idx, target, K, N$)
(1)  $besterr \leftarrow 2N$
(2)  $bestidx \leftarrow \{\emptyset\}$
(3)  $j \leftarrow idx$
(4)  $maxidx$ is the maximum value we may try assigning to
     this position and leave room for $K - startpos$ additional
     assignments.
(5)  $maxidx \leftarrow ((sol[1] + d - (K - pos)) \mod d) + 1$
(6)  while $j \neq maxidx$
(7)     $err \leftarrow \delta_{abs}(target, V[j], N)$
(8)     if $err < besterr$
(9)        $besterr \leftarrow err$
(10)       $bestidx \leftarrow \{j\}$
(11)    else if $err = besterr$
(12)       $bestidx \leftarrow bestidx \cup \{j\}$
(13)    $j \leftarrow (j \mod d) + 1$
(14) return $bestidx$
```

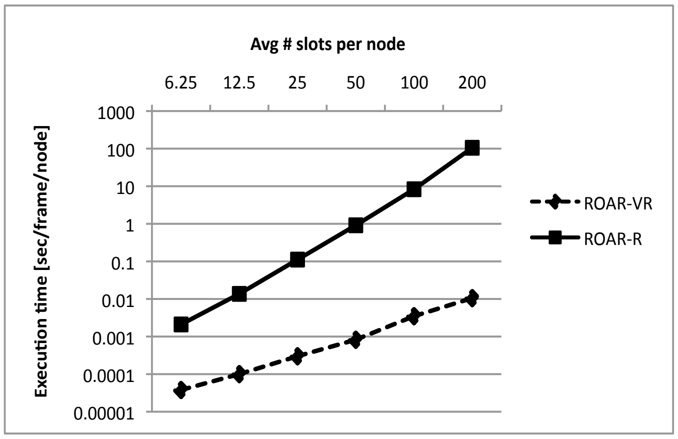


Fig. 4. Comparison of heuristic ROAR-VR to ROAR-V run time

Algorithm 5:

```
PICKFROMTWO($V, idx, target, u, K, x$)
(1)  if $idx = K - 1$
(2)     We can close the ring
(3)     Try $sol[idx + 1]$ as $x[1]$ and $x[2]$, the one that minimizes
        the total variance of the solution over the whole ring.
(4)     $bestidx \leftarrow$ best choice
(5)  else
(6)     We cannot close the ring
(7)     Try $sol[idx + 1]$ as $x[1], x[2]$, and run FindNext with a
        target of $target + u$, then pick the answer from FindNext
        with minimum $\delta_{clk}$ from $sol[idx + 1]$ to $target + u$ with
        random tie break. Call this $nextidx$.
(8)     Pick $x[1]$ or $x[2]$ that minimizes the sum-squared $\delta_{clk}$ from
        $V[x[1]]$ or $V[x[2]]$ to $V[nextidx]$, with random tie break.
(9)     $bestidx \leftarrow$ best choice
(10) return $bestidx$
```

References

[1] ANSI/IEEE, *Std 802.11 1999 Edition [ISO/IEC 8802-11:1999(E)], Part 11: Wireless LAN Medium Access Control (MAC) and Physical layer (PHY) Specifications.* New York: IEEE, 1999.

[2] L. Bao and J. J. Garcia-Luna-Aceves, "A new approach to channel access scheduling for ad hoc networks," in *Proc. MobiCom '01*. New York, NY, USA: ACM Press, 2001, pp. 210–221.

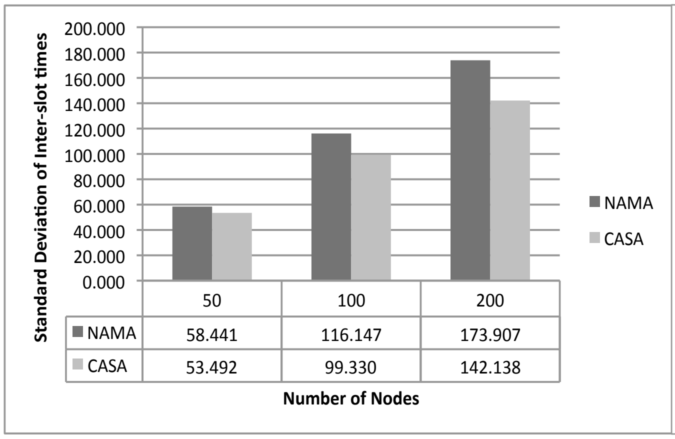
[3] J. Garcia-Luna-Aceves, M. Mosko, I. Solis, R. Braynard, and R. Ghosh, "Context-aware packet switching in ad hoc networks," in *Proc. PIMRC 2008*, 2008.

[4] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM 2001*, 2001, pp. 149 – 160.

[5] G. S. Manku, "DIPSEA: A modular distributed hash table," Ph.D. dissertation, Stanford University, 2004.

[6] A. Aggarwal, H. Imai, N. Katoh, and S. Suri, "Finding k points with minimum diameter and related problems," *J. Algorithms*, vol. 12, no. 1, pp. 38–56, 1991.

[7] V. Radhakrishnan, S. O. Krumke, M. V. Marathe, and D. J. Rosenkrantz, "Compact location problems," in *Lecture Notes in Computer Science*. Springer Verlag, 1993, vol. 761.



	50	100	200
NAMA	58.441	116.147	173.907
CASA	53.492	99.330	142.138

Fig. 5. Slot Allocator Comparison (inter-slot jitter)

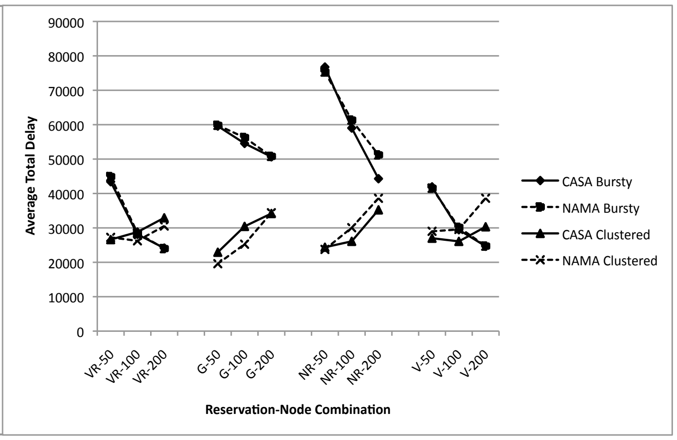
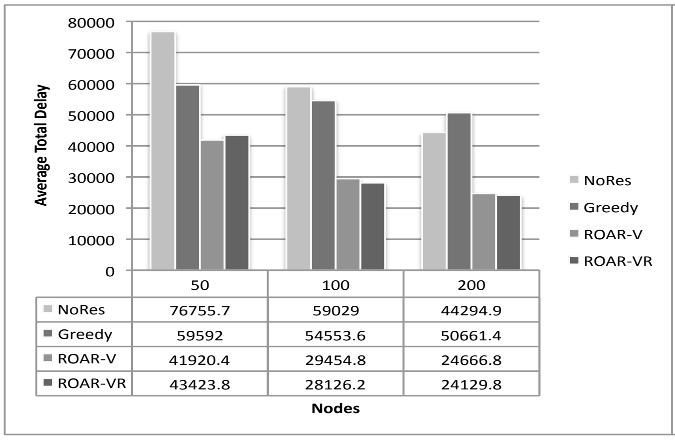
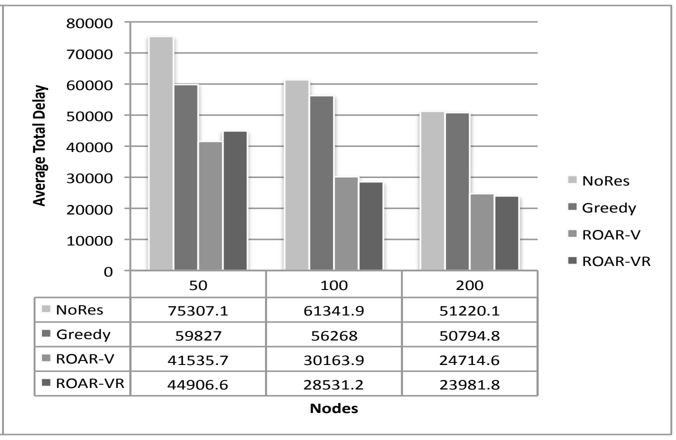


Fig. 6. Slot Allocator Comparison (average delay)



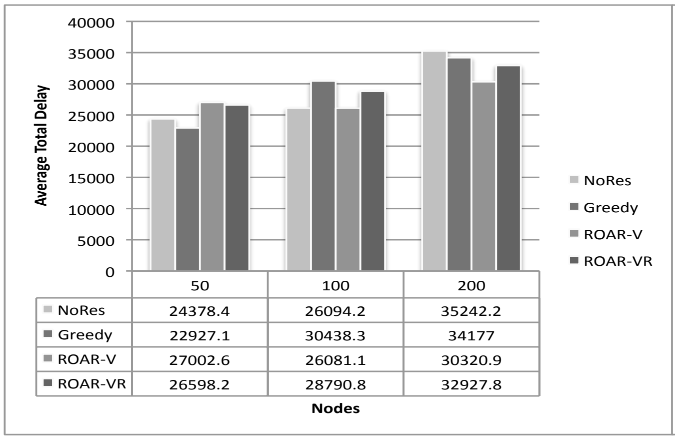
	50	100	200
NoRes	76755.7	59029	44294.9
Greedy	59592	54553.6	50661.4
ROAR-V	41920.4	29454.8	24666.8
ROAR-VR	43423.8	28126.2	24129.8

Fig. 7. CASA Scheduler, Bursty Traffic



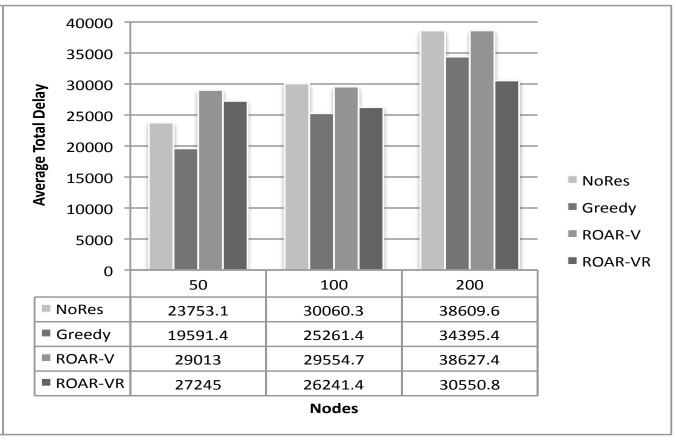
	50	100	200
NoRes	75307.1	61341.9	51220.1
Greedy	59827	56268	50794.8
ROAR-V	41535.7	30163.9	24714.6
ROAR-VR	44906.6	28531.2	23981.8

Fig. 8. NAMA Scheduler, Bursty Traffic



	50	100	200
NoRes	24378.4	26094.2	35242.2
Greedy	22927.1	30438.3	34177
ROAR-V	27002.6	26081.1	30320.9
ROAR-VR	26598.2	28790.8	32927.8

Fig. 9. CASA Scheduler, Clustered Traffic



	50	100	200
NoRes	23753.1	30060.3	38609.6
Greedy	19591.4	25261.4	34395.4
ROAR-V	29013	29554.7	38627.4
ROAR-VR	27245	26241.4	30550.8

Fig. 10. NAMA Scheduler, Clustered Traffic