Delay Contained: Scheduling Voice Over Multi-hop Multi-channel Wireless Mesh Networks

Vijay Gabale, Ashish Chiplunkar, Bhaskaran Raman, Partha Dutta

Abstract: In this work, we consider the goal of scheduling the *maximum number of voice calls* in a TDMA-based multi-radio, multi-channel mesh network. One of main challenges to achieve this goal is the difficulty in providing strict (packet-level) delay guarantees for voice traffic in capacity limited multi-hop wireless networks.

In this respect, we propose DelayCheck, an online centralized scheduling and call-admission-control (CAC) algorithm which effectively schedules constant-bit-rate voice traffic in TDMA-based mesh networks. DelayCheck solves the joint routing, channel assignment and link scheduling problem *along with delay constraint*. We formulate a relaxed version of this scheduling problem as an Integer Linear Program (ILP), the LP version of which gives us an optimality upper bound. We compare the output of DelayCheck with the LP-based upper bound as well as with two state-of-the-art prior scheduling algorithms. DelayCheck performs remarkably well, accepting about 93% of voice calls as compared to LP-based upper bound. As compared to state-of-the-art algorithms, DelayCheck improves scheduler efficiency by more than 34% and reduces call rejections by 2 fold. We also demonstrate that DelayCheck efficiently exploits the number of channels available for scheduling. With implementation optimizations, we show that DelayCheck has low memory and CPU requirements, thus making it practical.

1 Introduction

Wireless mesh networks have become a popular choice for providing data, voice and video applications in the context of enterprise networking [5], community or metro-scale networking [6] and public emergency-control systems [1]. Infrastructure-based wireless mesh networks consist of a network of statically positioned mesh routers. Such back-haul network architecture is reliable, scalable, cost-effective, and easy to deploy [8]. However, due to broadcasting in multi-hop wireless architecture, capacity limitation is one of the fundamental issues in wireless mesh networks [8]. In this context, it is well known that CSMA-based multi-hop MAC gives poor throughput [19] and results in high delay and jitter, which is unsuitable for real-time applications. Hence there is significant literature which has considered a TDMA-based approach for a multi-hop wireless mesh network, including TDMA-based WiMAX mesh [3].

The problem of scheduling transmissions in TDMA-based mesh networks is an active and stimulating area of research (see [17], [23] and references thereof). In this regard, several problems, like finding minimum length transmission schedule or channel assignment using minimum number of channels are proven NP-hard problems [14]. Consequently, there is a large body of literature which proposes heuristic-based channel assignment algorithms and/or transmission scheduling algorithms based on a fixed set of inputs. However, the problem of scheduling while considering strict (packet-level) delay constraint in multi-radio, multi-channel mesh network has not been studied extensively. In particular, we are not aware of any *online*, practical and effective scheduling algorithm which schedules a dynamic set of voice calls in multi-radio, multi-channel wireless mesh networks. It is this gap that our work fills in.

We consider constant-bit-rate (CBR) traffic to support voice calls in standalone multi-radio, multi-channel TDMA-based wireless mesh networks. Our goal is to support the maximum number of voice calls in a given period of time (e.g. over a day). The first and most important challenge to achieve this goal is the difficulty to provide strict (packet-level) delay guarantee for voice traffic in capacity limited multi-hop wireless networks. The voice delay constraint especially becomes challenging to handle in low data rate mesh networks, like a mesh network envisioned in [15] using IEEE 802.15.4 technology.

Secondly, scheduling transmissions in multi-hop wireless networks is a highly complex problem with numerous inter-related-sub-problems like finding path of communication (optimal routing problem), efficient utilization of available channels (optimal channel assignment) and interference-free link activation (optimal link scheduling). Also the delay-constraint has to be tightly integrated with these sub-problems. Thus, integrating solutions to these sub-problems for an efficient and effective scheduling algorithm is another challenge. Thirdly, for practical utility, the algorithm must be able to handle dynamic demand of traffic flows on-the-fly, while being memory and CPU-efficient.

In this regard, we propose DelayCheck, an online centralized scheduling and call-admission-control (CAC) algorithm which effectively schedules constant-rate voice traffic in TDMA-based mesh networks. With respect to above mentioned challenges, our work makes following contributions.

- DelayCheck is an *online*, polynomial time algorithm for delay-constrained centralized scheduling in a multi-radio, multi-channel, TDMA based mesh network (§5).
- DelayCheck works by jointly finding a routing path and allocating [slot, channel] resource tuples to the links on the

path while taking into account the strict delay-constraint (§5).

• We also give the implementation details of DelayCheck. DelayCheck has low memory and CPU requirements, yet being effective (§6).

• We formulate a relaxed version of the *delay-constrained* scheduling as an Integer Linear Program (ILP) for multi-radio, multi-channel TDMA-based mesh networks. The LP relaxation of this problem gives us an upper bound on the optimal solution (§7).

• We simulate DelayCheck on a 125-node 802.15.4-based mesh network for different traffic loads by varying the mean inter-call duration (§8). DelayCheck performs remarkably well accepting 93% of the calls with respect to the LP-based upper bound. This evaluation for 802.15.4 setting, which has impoverished data rate of 250Kbps, acts as a stringent test case for the performance of the scheduling algorithm.

• We further demonstrate that DelayCheck efficiently exploits the available number of channels in the network (§8).

• We compare the performance of DelayCheck with two state-of-the-art scheduling algorithms. DelayCheck performs about 34% better in terms of accepting the number of voice calls, even under strict delay constraint (§8).

To our knowledge, DelayCheck is the first scheduling algorithm which jointly considers packet-level delay constraint with a multi-radio, multi-channel TDMA-based mesh setting to support a dynamic set of voice calls. The rest of the paper is organized as follows. Next section (Sec. 2) motivates the need for delay-constrained scheduling. Sec. 3 compares prior work with DelayCheck for delay-constrained scheduling in TDMA-based mesh networks. Sec. 4 describes the network model and formulates the scheduling problem. In Sec. 5, we present DelayCheck algorithm along with a sample run on an input network topology. In Sec. 6, we briefly describe implementation details of the algorithm. Next, Sec. 7 describes the linear programming formulation to find the upper bound on optimal solution. In Sec. 8, we present simulation based evaluation and comparison of DelayCheck. We conclude the paper in Sec. 9.

2 Motivation

Our interest in delay-constrained scheduling stems from the goal of supporting a number of simultaneous voice calls, each of which has strict (packet-level) delay constraints, in TDMA-based mesh networks like 802.11 mesh (Fractel [10]), 802.16 mesh ([3]) and 802.15.4 mesh (Lo^3 [20], [15]). In this work, we assume that certain percentage of capacity is reserved to support the voice traffic in the network. The data and video applications can then be served in the rest of the available capacity; the scheduling of non-voice traffic itself is orthogonal to our work.

The scheduling algorithms designed so far (e.g. [14], [27], [18]) assume availability of high data rate backbone links; link speeds of order of few tens of Mbps. This implies that packets take much less time (order of few μs) to travel over a link and this results in the assumption that the end-to-end delay requirement for a voice call is *not a concern* of the scheduler. However, as the load of voice calls increases, although some voice calls may get accepted, they may fail to meet the end-to-end delay requirement. To justify this claim, we present a preview from our simulations of DelayCheck and two state-of-the-art scheduling algorithms, described in Sec. 8. The first algorithm, interference-aware CAC [26], schedules the links in interference-free manner in multi-channel networks. The second algorithm, even-odd scheduling [18], schedules pairs of nodes alternatively in multi-channel networks. Here, the scheduler gives a theoretical bound on packet-level delay, however does not admit or reject calls based on per-flow delay constraint. We apply these scheduler schemes to a WiFi TDMA mesh setting [12]; the parameter details are explained in Sec. 8. We increase the load on the scheduler by increasing the number of simultaneous calls active in the network. We assume 250ms ([24], [4]) as the tolerable value of delay for a voice packet.

Fig. 1 shows the behaviour of these scheduling schemes in terms of worst case end-to-end delay of an accepted voice call. For the case where the entire capacity of the network is used to support voice calls, and as the number of calls exceed 42, the prior algorithms accept the voice calls but fail to satisfy the packet-level delay constraint. Similar behavior can be observed when only 40% of the capacity is used to schedule the voice in the network. For both the cases, DelayCheck algorithm accepts the voice calls and at the same time, follows the strict delay constraint. Also when capacity is limited to 40%, the graph for DelayCheck shows that DelayCheck rejects the calls if the delay constraint is not satisfied. How DelayCheck meets such a strict delay constraint is part of the algorithm, explained in Sec. 5. Thus, although designed for interference-free multi-channel scheduling, prior algorithms do not guarantee packet-level delay constraints.

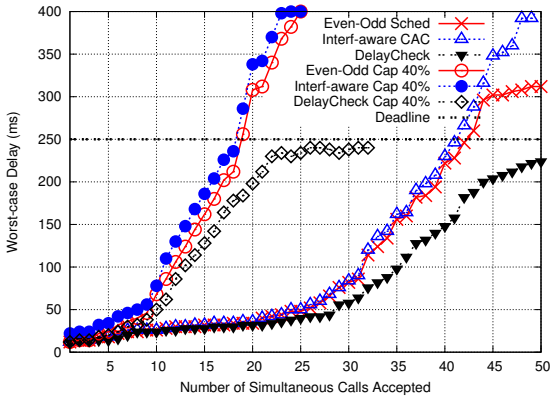

Figure 1. Prior work: voice calls fail to meet deadline

In Lo^3 [20], authors envision an 802.15.4-based multi-hop, multi-channel TDMA mesh network to support voice-based applications like two-way interactive voice using low-cost, low-power platforms. Here, the delay constraint becomes important because 802.15.4 radio has impoverished data rate of 250kbps. This entails a larger slot size (order of ms) to accommodate a voice packet and hence scheduling with larger frame size (order of few tens of ms). This implies that, now, meeting the voice packet deadline (of 250ms) over a multi-hop network becomes even more challenging.

As can be observed from Fig. 1, the end-to-end delay becomes critical even in 802.11 or 802.16 networks under heavy load of voice traffic and/or when the capacity reserved for voice is limited.

3 Related Work

There is large body of literature which has considered scheduling for multi-hop wireless networks. We describe the main related work here and explain how our algorithm differes from these.

The work in [21] proposes and evaluates a Load-Aware Channel Assignment (LCA) algorithm in a network built using standard 802.11 hardware. However, it is unclear how admission control can be implemented in such a CSMA-based multi-hop setting. Also in the proposed scheme, the channel assignment to radios is fixed. As opposed to CSMA-based channel access, [25] proposes a dynamic channel assignment and link scheduling mechanism for multi-radio, multi-channel TDMA-based wireless mesh networks. However, the subproblems of slot scheduling and channel assignment are not solved jointly which results in suboptimal solution (approximation ratio $= log|$total flows$|$).

The main distinction of DelayCheck from above algorithms is its consideration of *delay-constrained* scheduling. Also DelayCheck differs from these algorithms in its consideration of joint routing, channel assignment and scheduling in a TDMA-based multi-radio, multi-channel wireless mesh network. The work in [9] too, formulates the joint channel assignment and routing problem taking into account the interference constraints. It develops an algorithm that optimizes the overall network throughput subject to fairness constraints. But the assumption in [9] that traffic between a node and the gateway nodes is routed on multiple paths may not suit real-time traffic due to possible out of order packet delivery. Also importantly, unlike [9], DelayCheck considers strict delay constraints.

Other prior work has considered path delay aspect in scheduling. For instance, considering delay-sensitive traffic, [14] attempts to find the minimum length TDMA schedule that also minimizes end-to-end scheduling delay. It rightly identifies that *scheduling delay* depends on the order in which consecutive links are scheduled, as we do in DelayCheck too. However, the proposed algorithm does not consider strict delay guarantee and it is for the single channel case; modeling the scheduling delay in the multi-channel case is not discussed. Exploiting multiple channels in WiMAX mesh, [27] formulates the problem of packet transmission scheduling for real-time CBR (constant-bit-rate) traffic as a binary linear programming problem, and solves it using a heuristic based solution. However, this formulation does not consider out of order transmission on the adjacent links as in [14] and DelayCheck. It also restricts the scheduling of links to a single frame (unlike in [14] and DelayCheck).

Taking the approach of integrated routing, channel assignment and slot scheduling, [18] proposes a generalized link activation framework for scheduling packets over wireless backhaul (TDMA based WiMAX or WiFi). However, due to even-odd labeling, there may not be a feasible route in this scheme even if two nodes are able to communicate with each other. Also, each link is scheduled only half of the time and the bandwidth requirement of each link is constrained not to exceed half of the total capacity of the link.

As compared to these attempts (e.g. [14], [18] where delay is minimized but not strictly constrained), DelayCheck actively considers delay as constraint to schedule voice calls. Our goal is to schedule maximum number of voice calls. To achieve this goal, we jointly solve the routing, multi-radio, channel assignment and delay-constrained link scheduling problem. Our scheduling is not restricted to a single TDMA frame. DelayCheck also has a CAC module which rejects the calls if the delay-constraint is not met.

The work in [22] also considers packet-level delay guarantees, like DelayCheck; it proposes a routing and CAC (Call Admission Control) heuristic such that every packet of admitted flows strictly meets its delay and jitter requirements. This is especially useful for voice and video applications. One main difference in DelayCheck, compared to [22] is that DelayCheck is *online*. That is, it does not disturb ongoing flows while scheduling new flows. Also, [22] considers only a single channel and a single radio network, while DelayCheck considers a multi-radio, multi-channel setting.

4 Problem Formulation

We now describe the network model and explain the set of inputs, the set of constraints and the optimization goal of our delay-constrained scheduling problem.

Network Model: We represent wireless mesh network as a graph $G(\mathbf{V},\mathbf{E})$ having $|\mathbf{V}|$ nodes and $|\mathbf{E}|$ links. We assume that one of the nodes acts as the central controller and schedules all the calls. We refer to this node as the root node. Our algorithm takes as input a network connectivity graph and an interference graph. The interference graph indicates whether two links in the connectivity graph are intefering or not[1]. Both these graphs can be *arbitrary* graphs and our algorithm is not resticted in any way (e.g. to a unit disk graph [11]). We assume that all the links in the network have the same PHY layer data rate and that the channel conditions do not change for the period of scheduling. Each node is equipped with one or more radios and each radio has a fixed set of orthogonal channels (or frequencies). We assume a fixed size frame T_f as the part of TDMA MAC protocol. The frame has a fixed number of slots S, of fixed size T_s ($T_f = S * T_s$). For example, WiMAX networks could have $T_f = 10$ms with $T_s = 150\mu$s [7]. Considering low overhead of channel-switching time in practical implementations (40μ for 802.11a [13], 330μ for 802.15.4 [20]), we assume slot-level channel-switching for multi-channel allocation.

Input: Fig. 2 shows the set of inputs for the centralized scheduler.

• A scheduling interval in number of slots (the time period after which the schedule repeats itself): typically this interval

[1]Each node in the interference graph corresponds to a link in the connectivity graph, and two nodes are adjacent iff the corresponding links are interfering.

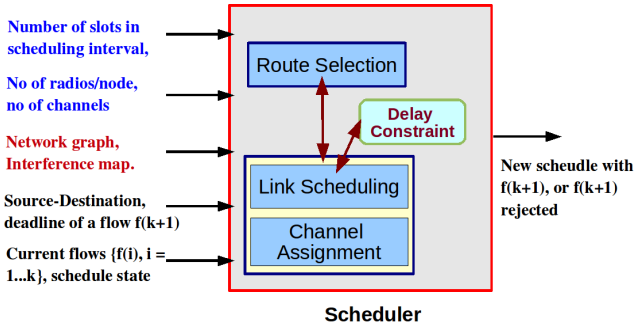

Figure 2. DelayCheck scheduler inputs

is either the frame size or an integer multiple of frame size.

• The number of radios at each node, and the number of channels per radio.

• A network connectivity graph and an interference graph. Note that, we assume connectivity between two nodes using bi-directional links.

• A source and destination pair of a voice call and a predetermined tolerable end-to-end (packet-level) delay value i.e. deadline (in number of slots).

• In addition, the online scheduler has to maintain the state (allocation of [slot, channel] tuple to the links) of already established calls.

Constraints: We now state the constraints of the problem.

• *Flow-schedulability and routing path:* There should be a path in the connectivity graph between source and destination if the flow is to be scheduled.

• *Primary interference:* For a node, no two of its radios should operate on the same channel in the same time slot.

• *Secondary interference:* A radio can only transmit or only receive at a time. For collision-free scheduling, transmission on a link in a slot and in a channel should not interfere with other interfering receptions (as per the interference graph).

• *Delay Constraint:* The total end-to-end time required to deliver each packet from source to destination should be less than the tolerable delay limit.

Optimization goal: Our goal is to maximize the number of voice calls successfully scheduled over a period of time (e.g. over a day).

Output: The scheduling algorithm assigns time slots and channels to the links of the network so that all of the above mentioned constraints are satisfied and the goal of maximizing the number of voice calls is met.

5 DelayCheck

In general, the delay-constrained scheduling in multi-hop networks with a set of flows as input is an NP-hard problem [16]. In this section, we describe *DelayCheck*, an online, heuristic-based, polynomial time CAC-algorithm which jointly finds a route, allocates the channels, schedules the links (and the radios) of an input flow in delay constrained manner.

The algorithm works in three phases. In the first phase of *construction*, for a new call request between a source and a destination, the algorithm constructs an auxiliary graph G' from network connectivity graph G considering the state of already established calls. Every path between source and destination on such an auxiliary graph gives a feasible schedule. To find such a schedule, in *allocation* phase, DelayCheck uses Dijkstra's shortest path subroutine to find the shortest path in auxiliary graph. This shortest path not only gives a feasible routing path in original input network graph but also finds a feasible schedule in delay-constrained manner.

The first two phases of the algorithm only consider a limited interference model (up to 2-hops). To incorporate an arbitrary interference graph, we have a third phase, the *post-processing phase*, where the path goes through a filtering phase to ensure that the schedule is conflict-free for the given interference graph. The final schedule thus assigns [slot, channel] tuple to each link on the path such that the flow-schedulability and routing constraint, interference constraint and the delay-constraint are satisfied.

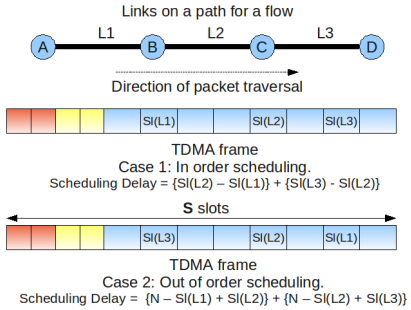

Figure 3. Out of order scheduling of links

We now describe each phase of the algorithm. We use the following notation. Let $\mathbf{V} = \{1, \ldots, V\}$ be the vertices in network graph, $\mathbf{S} = \{1, \ldots, S\}$ be the slots in the scheduling interval, $\mathbf{C} = \{1, \ldots, C\}$ be the channels available at a radio and $\mathbf{D}$ be the deadline (integer value, in terms of number of slots). Now, the delay involved in a path depends on the time-slots assigned for the links in the path. If links l_1, ..., l_m on a path are assigned slots n_1, ... n_m, then the total delay is $\sum_i((n_{i+1} - n_i) \bmod \mathrm{S})$ where $(n_{i+1} - n_i) \bmod \mathrm{S}$ is $n_{i+1} - n_i$ if $n_{i+1} > n_i$ and is $S - n_i + n_{i+1}$ otherwise, as shown in Fig. 3.

5.1 Phase I: Constructing the auxiliary graph

As the part of the algorithm, to schedule every new flow, we construct an auxiliary graph G' on-the-fly. A vertex in this graph is a tuple *(v,s,c,d)*. That is for each vertex v ($\in \mathbf{V}$) in original network connectivity graph, for each slot s ($\in \mathbf{S}$), for each channel c ($\in \mathbf{C}$) and for each integer delay value d ($\leq \mathbf{D}$), we have a vertex in the auxiliary graph. We now define 4 rules, as shown in Fig. 4, to add a directional edge in the auxiliary graph from vertex (v, s, c, d) to vertex (v', s', c', d'). These rules construct the requisite edges and complete the construction of the auxiliary graph.

Now, what does a link in G' mean? The rules in Fig. 4 are intended to signify the following meaning. Let us represent tuple (v, s, c, d) by vertex w and (v', s', c', d') by vertex w'. If edge (w, w') in G' is selected in phase II, this signifies that the edge between (v,v') is scheduled in slot s', in channel c' and the delay of the path so far (i.e. from the source to v')

Rule 1: In the original network connectivity graph G, v and v' must be connected to each other.
Rule 2: If there is only one radio available at node v, then s should not be equal to s'.
Rule 3: In slot s', there are some channels which are disallowed at node v (if node v transmits on any of these channels, it may disrupt the already admitted calls) and there are some channels which are disallowed at node v' (if node v' receives on any of these channels, it may result in collision due to already admitted calls). We term the set of channels allowed at a node v as **AllowChann(v)**. Now, if s is equal to s', c should not be equal to c'. Moreover, $c' \in AllowChann(v)$ and $c' \in AllowChann(v')$ and there should be at least one radio available at each node.
Rule 4: $d' = d + (s' - s) \bmod S$

Figure 4. Rules for constructing auxiliary graph

is d'. Note that the attachment of the above significance to a link (w, w') ignores s & c (s & c have meaning for any link of the form $(*, w)$). This effectively means that, if the source vertex is v'', all nodes of the form (v'', s, c, d) can be condensed into a single vertex v'' in G'. With this modification to G' (not mentioned in Fig. 4 for clarity), we are ready to move on to the next phase, where the above-mentioned rules of G' construction find significance.

5.2 Phase II: Allocating resources

Given the above meaning for a *link* in G', we can now give the **interpretation** of any *path* in the auxiliary graph with respect to rules in Fig. 4. If we map any path in G' to G (using the first element of the 4-tuple), then because of rule 1, it gives a routing path in G between source and destination. Due to rule 2, we avoid primary interference where a node does not transmit and receive at the same time. Now, rule 3 is intended to capture secondary interference. However, in phase I and II, only a restricted form of interference is accounted for, leaving generic interference handling to phase III. In Fig. 4, we in fact capture only 1-hop interference: this is for ease of exposition; Sec. 5.4 elaborates how we can extend this to account for 2-hop interference.

Rule 3 in Fig. 4 ensures that the channels allocated to the links of the new flow avoid 1-hop (secondary) interference with the already admitted flows as well as with itself. And finally because of rule 4, among the possible paths between source and destination, only those paths which follow the strict delay deadline get selected.

Now in G', every path from source to a node of the form (destination, *, *, *) has following properties:

• **Property 1:** All the links on the path have an interference-free slot and channel assignment (ensured by rules 1, 2 and 3).

• **Property 2:** A packet originated from the source reaches the destination within delay constraint (ensured by rule 4).

We now run the dynamic programming based Dijkstra's shortest path algorithm to find the shortest path from source to any (destination,*,*,*) in G'. Among all the feasible paths with respect to above mentioned properties, the algorithm finds a shortest path from source to (destination, *, *, *) and

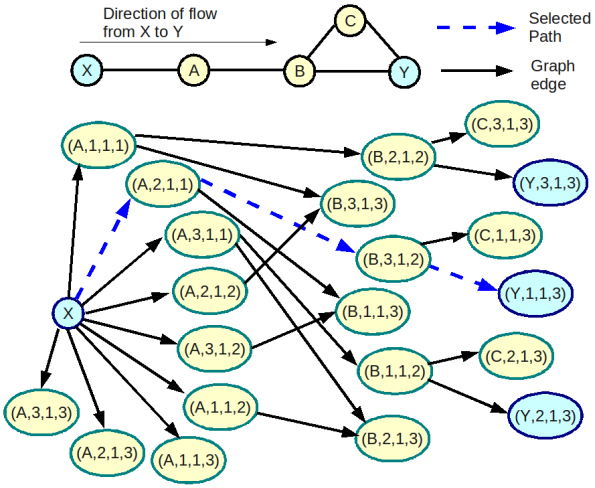

Figure 5. Constructing Auxiliary graph: example topology

outputs the sequence of nodes on the path. These nodes are actually the tuples of the form (v, s, c, d). Thus the sequence of nodes output by the algorithm, gives the joint routing, channel assignment and link scheduling of the new call request, meeting the delay-constraint in the process.

We justify the choice of *shortest* path in this phase as follows. Considering future call requests, a new call should be admitted in such a way so that the minimum amount of resources are blocked. The amount of resources in this case are the links in the path. This then transforms to a goal of scheduling the new flow on a shortest hop path which satisfies the problem constraints. Note that, this way, DelayCheck need not be optimal but schedules the flows on shortest path from available paths, to heuristically maximize the number of calls admitted in a given period.

Complexity of Algorithm: For e the number of edges and n the number of vertices in the graph, the running time of Dijkstra's shortest path algorithm is $O(n^2)$ with adjacency matrix representation and is $O(e(logn))$ with a partially ordered tree data structure. For DelayCheck, n is equal to the size of (v, s, c, d) tuple which is $O(|\mathbf{V}||\mathbf{S}||\mathbf{C}||\mathbf{D}|)$ while e can be $O((|\mathbf{V}||\mathbf{S}||\mathbf{C}||\mathbf{D}|)^2)$ at most.

Example: We now explain phase I and phase II using a simple 5 node network topology as shown in Fig. 5. Suppose we have to schedule a call from node X to node Y. Assume that the TDMA frame has only 3 slots and there is only 1 channel available for scheduling. Further assume that, the deadline for the call is 3 slots. Fig. 5 also shows the auxiliary graph constructed for this set of assumptions and for the given input network graph. The path selected at the end of phase II is also shown in the figure. The schedule for call from X to Y in this case is as follows: Link X-A is scheduled in slot 2 (of frame 1), using channel 1 and delay at A is 1 slot. Similarly, link A-B is scheduled in slot 3 (of frame 1), in channel 1 and delay at B is 2 slots. Finally, link B-Y is scheduled in slot 1 (of frame 2), in channel 1 and delay at Y is 3 slots. Note that, in auxiliary graph, the slot, channel and delay variables are absent from the tuple for the *source vertex* X. Also, for the simplicity of the diagram, the nodes which have no incoming edges are not shown in example auxiliary graph.

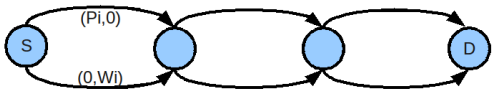

Figure 6. Constrained-fuel problem is NP-Hard

However complexity of phase II is exponential if D is given as input as it requires log D bits to represent D. And we expect that for any delay-constrained algorithm, like DelayCheck, D would be given as input. Hence, we defer the explanation of phase III and first explain our modification of the above algorithm to make it polynomial.

5.3 Running Phase II in polynomial time

We now explain how we cut down the complexity of phase II to polynomial by solving an analogous but simple real-life problem. Consider a graph of roads (corresponding to links in G') where each road has two labels: toll (corresponding to hops) and amount of fuel required (corresponding to delay) . Now we are given a source and destination and a fixed amount of fuel to start with. The goal is to choose a path which can take the car from source to destination with the given quantity of fuel (within a delay limit) which minimizes the amount of toll required to pay (number of hops). We call this problem (which is very analogous to delay-constrained scheduling) the **constrained-fuel** problem.

We first show that, constrained-fuel problem is NP-hard by reducing Knapsack problem to it. Then, we show that if the toll for each road is restricted to 1 unit, the problem is no-more NP-hard and can be solved in polynomial time. Since the toll corresponds to hop in auxiliary graph G', this is exactly what we require to make phase II run in polynomial time.

The reduction works as follows. Consider an instance of Knapsack problem where object i out of $1, \ldots N$ has weight w_i and profit p_i; and the capacity of the knapsack is W. We construct the corresponding instance of constrained-fuel problem as follows. The graph has $N+1$ vertices $0, \ldots, N$ where the source is 0 and the destination is N. For each $i \in \{1, \ldots, N\}$ there are two paths from vertex $i-1$ to vertex i, as shown in figure 6. The "upper" path consumes toll p_i and no fuel, while the "lower" path consumes fuel w_i and no toll. The total amount of fuel available is W. Now suppose the optimal path for this instance of constrained-fuel problem has been found out. We construct the solution to the given instance of Knapsack problem as follows. The optimal path must pass through vertices $1, \ldots, N$ in that order. For every object i, we put the object in the knapsack if and only if the optimal path takes the lower edge from vertex $i-1$ to vertex i. Due to this construction, there is a bijection between the paths from vertex 0 to vertex N, and the subsets of the objects. The time taken to traverse a path P is related with the profit achieved by the corresponding set of objects as follows. Let $S = \{i \in \{1, \ldots, N \mid P$ uses the lower edge to go from vertex $i-1$ to vertex $i\}\}$. Hence S is the set of objects put into the knapsack. The toll to be paid on the path P is $\sum_{i \notin S} p_i = \sum_i p_i - \sum_{i \in S} p_i$. Since $\sum_i p_i$ is independent of P (and S), the feasible path which minimizes the time taken corresponds to the feasible set of objects which maximizes the profit.

However, in Knapsack problem, if every object has profit of 1 unit, the problem is not NP-hard. Similarly, if we constrain the toll for a road such that each road requires exactly 1 unit to pay, then the problem is *no more* NP-hard. We now discuss a polynomial time algorithm to this constrained problem. As shown in Fig. 7, we reduce this optimization problem in polynomial time to a decision problem and further to a simpler optimization problem. For the first reduction, in the decision version, we ask the question as to whether we can find a path which consumes at most W fuel and at most T toll. The value of T is iterated from 1 to $|V|$ (number of vertices). For the second reduction, once we select a value for T, we can find a path which consumes minimum amount of fuel. To reduce back to original optimization problem, if we can find a path which takes at most T toll and consumes W' fuel, we can compare W' to W. If $W' \leq W$, we are done, otherwise we increase T and find new value of W'. If $W' \leq W$, we get the solution to initial optimization problem with T being the minimum toll and W' being the fuel consumed.

Now to solve the reduced optimization problem, we construct a graph G^1 (as shown in Fig. 8) as follows. We create T levels in G^1, denoted as $G_1, G_2, \ldots, G_T$. Each G_i contains all the vertices as per the original graph G. For each edge (u, v) in G, we will have an edge between u and v where u is in G_i and v is in G_{i+1}. The weight on these edges corresponds to the amount of fuel required to cover the corresponding road in the original graph. We now apply Dijkstra's algorithm to this graph G^1. This finds a minimum weight path, with path length $= i$, i varying from 1 to T, from source s to target t. That is, for a given path length, we find a path which consumes least amount of fuel. This solves the reduced problem in polynomial time and since each reduction takes polynomial time, the original problem is also solved in polynomial time.

Modification to phase I: Now, from above discussion, to make DelayCheck polynomial, we change the tuple $(v, s, c, \mathbf{d})$ to $(v, s, c, \mathbf{m})$ and have weights on the edges of the auxiliary graph. The first 3 rules to form an edge between (v, s, c, m) and (v', s', c', m') remain the same. We redefine the rule 4 as follows. We will have an edge between (v, s, c, m) and $(v', s', c', m+1)$ with weight $= (s' - s) \bmod S$. With the changed rule, as in phase II, we apply Dijkstra's algorithm on the auxiliary graph to find the shortest hop path (among feasible paths) which satisfies the delay constraint D. But, the complexity of the phase II is now $O(V^2S^2C^2V^2)$ (assuming adjacency matrix representation). Since V (number of vertices), S (number of slots), C (number of channels) are fixed and number of hops can be at most V, this results in a polynomial time algorithm for delay-constrained scheduling.

5.4 Extension of phase I to practical 2-hop interference

Rule 3 of the *construction* phase works under the assumption of 1-hop interference model. To incorporate a 2-hop

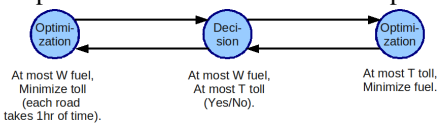

Figure 7. Reductions in constrained-fuel problem

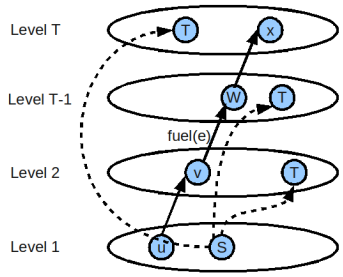

Figure 8. Solving the reduced constrained-fuel problem

interference model into the phase I, we now use 6-tuples in G', $[v, s, c, s1, c1, m]$. This increases the complexity to $O(V^2S^4C^4V^2)$. With this 6-tuple, we redefine rule 3 as follows. While constructing the edges, if s is equal to s', c should not be equal to c' (1-hop), moreover c should not be equal to c1 (2-hop)[2]. This way 2-hop interference can be eliminated by spatially reusing the channel only beyond 2 hops.

5.5 Phase III: Post processing

This is the phase which incoporates any arbitrary interference graph. We now explain phase 3 of the algorithm: post processing. To understand the need for post processing consider a hypothetical situation of a **weird** circular pond in a real-life deployment. Pond is just one case of arbitrary interference map. Assume that there are few nodes deployed which surround this pond. The distance between two nodes at the diametrically opposite side of the pond is just more than the transmission range. Hence the call has to go through the nodes surrounding the pond over a circular arc. This may potentially lead to an interference situation as the links on the opposite side of the pond may interfere [3] but could be actually separated by more than two hops.

Thus the algorithm requires a post-processing step where the path output by algorithm is scanned for possible interfering edges 'along the path'. Then the edges are either scheduled on different channels or on different slots. However, our evaluation on random topologies showed that such cases occured very rarely. Hence, in the current version of the algorithm, we simply reject the call if we find that the admitted path has interfering links beyond two hops. We give following reasoning for this rejection. If edges on the path are inteferring beyond 2-hops, this means that the selected path is really **bad**, it is unnecessarily consuming more resources than required, and this is contradictory to our original reasoning that the path should consume minimum resources.

5.6 Optimization

For Dijkstra's algorithm, the value of shortest path metric is unique but there could be many shortest paths in the graph having same metric value. Dijkstra's algorithm can be easily tweaked to find out all possible shortest paths without increasing the complexity. Now, if the allocation on one shortest path fails in phase III, we can select another shortest path to allocate the resources.

[2]In the earlier construction, s & c were used to "remember" the slot & channel allocation in the previous link; we now need to "remember" slot & channel allocation across two links.

[3]becasue of absence of any node in between in the the pond, otherwise the shortest path would have gone through this node.

6 DelayCheck: Implementing In Practice

To evaluate performance of DelayCheck, we implemented DelayCheck in a custom-built discrete event simulator with two implementation tricks.

Because of the four variables (v, s, c, m) in a tuple (which corresponds to a node in auxiliary graph), the first and foremost concern is about the storage require for the implementation. For example, for 25 nodes, 100 slots, 10 channels and 10 hops, the number of nodes in graph is $25 * 10^4$. If the graph is to be stored as cost adjacency matrix, the storage required is huge, of order of 10^{10}. The first trick that we use to get rid of this huge storage is that, we don't store the graph at all. In Dijkstra's algorithm, all we need to check is whether two vertices are connected or not (and if connected, what is the cost of the edge), and this we check on-the-fly based the 4 rules defined earlier. Note that such a checking takes $O(1)$ time. With n as the number of vertices for a graph, the only storage we require is the $O(n)$ cost matrix for Dijkstra's algorithm. The cost matrix has two columns of $O(n)$ rows, first column is the vertex number whereas second column is the cost of the shortest path.

The second concern is the time required to search the state space over the auxiliary graph to find the shortest path. On 2.2GHz processor with 2GB RAM, the algorithm (for 25 nodes, 100 slots, 10 channels and 10 hops) in naive form takes 8 seconds of user time and 10 ms of system time to execute a request. Although, this much delay is amenable for real-time call set up, any further improvement is better for low call set up latency. The second trick that we use to reduce this time is to partition the state space. The partition is based on the highest numbered slot and highest numbered channel being used in the already existing schedule. During search, we consider the most likely partition. For example, for very first call, we do not search over all the channels as the call would get accepted using one or two channels. For subsequent calls, we maintain the highest numbered channel being used and start our channel search from this number. If the search fails in the partitioned space, we go to the next step of exploring a bigger partition. However, in our evaluation, almost always, the call was scheduled in the first partition of the search space. With this trick, the user and system time required for algorithm to execute new request reduced to 2 seconds and 1 ms respectively.

7 Integer Linear Program (ILP) formulation

In this section, we give an Integer Linear Programming (ILP) formulation for the delay-constrained, multi-hop, multi-radio, multi-channel TDMA scheduling problem.

The ILP is shown in Fig. 9. This ILP is an *offline* algorithm in that it aims to schedule not only a given flow, but also may reschedule other existing flows. That is, its input is a collection of flows. The variables $X_{s,c,f,e}$ capture the joint routing, slot, and channel assignment for each flow. The ILP's objective is to maximize the number of calls accepted, modeled by

Variables:
$X_{s,c,f,e}$ is 1 if link e (e is directional) for flow f is scheduled in slot s and on channel c.
Y_f is 1 if flow f is entirely scheduled.
$O_{f,v}$ is 1 if, for flow f, the transmission slot of the incoming edge at intermediate node v comes later than the outgoing edge. (out of order case mentioned in [14]).
Objective:
$max \sum_f Y_f$
Flow-schedulability:
$Y_f \leq \sum_s \sum_c \sum_{e \in (\text{OutEdges(src)})} X_{s,c,f,e}. \ldots (1)$
$Y_f \leq \sum_s \sum_c \sum_{e \in (\text{InEdges(dst)})} X_{s,c,f,e}. \ldots (2)$
Routing path:
For source vertex, *src* of flow f:
$\sum_s \sum_c \sum_{e \in (\text{OutEdges(src)})} X_{s,c,f,e} \leq 1. \ldots (3)$
$\sum_s \sum_c \sum_{e' \in (\text{InEdges(src)})} X_{s,c,f,e'} \leq 0. \ldots (4)$
For destination vertex, *dst* of flow f:
$\sum_s \sum_c \sum_{e \in (\text{InEdges(dst)})} X_{s,c,f,e} \leq 1. \ldots (5)$
$\sum_s \sum_c \sum_{e' \in (\text{OutEdges(dst)})} X_{s,c,f,e'} \leq 0. \ldots (6)$
For every v $\notin (src, dst) \forall f$
$\sum_s \sum_c \sum_{e \in (\text{OutEdges(v)})} X_{s,c,f,e} - \sum_s \sum_c$
$\sum_{e' \in (\text{OutEdges(v)})} X_{s,c,f,e'} = 0. \ldots (7)$
Primary interference:
$\sum_c \sum_f \sum_{e \in \text{InEdges(v) U OutEdges(v)}} X_{s,c,f,e} \leq R, \forall s, \forall v.$
$\ldots (8)$
Secondary interference:
$\forall$ e,e' $\in$ Interferece graph,
$\sum_f (X_{s,c,f,e} + X_{s,c,f,e'}) \leq 1, \forall s, \forall c,. \ldots (9)$
Delay-constraint:
$\forall$ f and $\forall$ v $\notin (src(f), dst(f))$,
$\sum_{e \in \text{InEdges(v)}} \sum_s \sum_c (s+1) X_{s,c,f,e} \leq \sum_{e' \in \text{OutEdges(v)}}$
$\sum_{s'} \sum_c s' * X_{s',c,f,e'} + (S-1) O_{f,v}. \ldots (10)$
$1 + (\sum O_{f,v} - 1) * S \leq D. \ldots (11)$
$X_{s,c,f,e} \in \{0,1\}, Y_f \in \{0,1\}, O_{f,v} \in \{0,1\}. \ldots (12)$

Figure 9. Integer Linear Program for integrated scheduling

Y_f variable. Constraints (1), (2) ensure that, for a flow, at least one outgoing edge from the source and at least one incoming edge to the destination is scheduled. In addition, constraints (3)-(6) ensure that the if a flow is scheduled, exactly one outgoing edge from the source and exactly one incoming edge to the destination is scheduled. Constraint (7) ensures that the flow is conserved at every intermediate node. Thus, these constraints together ensure that the variable Y_f is set iff the flow f is scheduled end-to-end from source to destination. Constraint (8) ensures that in a slot, for a node, the total number of channels used are less than the number of radios available (i.e. primary interference). Among the set of interfering links for a slot, constraint (9) allows only one link to be active (i.e. secondary interference). This also covers the case that a radio can not be scheduled for transmission and reception on two links in the same slot.

Constraints (10) and (11) model (a necessary condition for) the limit on end-to-end delay. The end-to-end delay is determined by the scheduling delays of the consecutive links [14]. There are two ways in which links can be scheduled: in order and *out of order*, as shown in Fig. 3. Let us denote the slot in which link L_i is scheduled by $Sl(L_i)$. For two consecutive links L_1 and L_2, if they are scheduled in order, the link scheduling delay is $Sl(L_2) - Sl(L_1)$. Otherwise in out of order case, this delay is $S - Sl(L_1) + Sl(L_2)$. Now, for in order case, if L_1 is scheduled in slot 1 and L_2 in slot S, the worst case delay is $S - 1$. For the out of order link scheduling, the worst case scheduling delay is $S - 1$ if $Sl(L_1) = Sl(L_2) + 1$. For the delay constraint, we want the sum of the individual link scheduling delays to be less than the given deadline. Now to model this constraint, we define a variable $O_{f,v}$ for every flow f and every node v. This variable will be set to 1 if flow f passes through v and incoming-outgoing links are scheduled out of order. Constraint (10) asserts that the slot in which e (incoming edge) is scheduled must be at most the slot of e' (outgoing edge), or the slot of e' + $(S-1)$, in which case $O_{f,v}$ will be set to 1. Constraint (11) asserts the lower bound on the deadline by constraining the end-to-end delay to be less than $1 + (\text{allowed number of out of order cases} - 1) * S$. Here, the first term is the lower bound on in-order scheduling while second term is the lower bound delay for out-of-order scheduling. Note that, the constraint (10) ensures that $O_{f,v} = 1$ if e and e' are scheduled out of order. However, it does not ensure the converse. But this is alright, the reason being as follows. Suppose e and e' are in order or f does not pass through v, but $O_{f,v} = 1$ at an optimal point p in the ILP's solution space. Consider the point p' obtained by setting $O_{f,v} = 0$ in p. Then p' is still feasible and has the same $\sum Y_f$ cost as p. If the optimal number of flows scheduled is F at the point p then the same number of flows get scheduled at the point p'. Furthermore, p and p' represent the same schedule. Conversely, if there is no feasible point for the given input set, the ILP with constraints (10) and (11) will also not find such an optimal point.

Note that the above ILP itself is a relaxed version of our delay constrained scheduling, since ILP constraints (10) & (11) capture only a *necessary* condition for the delay constraint, but not the *sufficient* condition. This is however alright, since we anyway use the ILP to get an upper bound on our optimization metric. To obtain the upper bound, we remove the integer variable constraint and solve the Linear Program (LP) version of the above formulation. The upper bound gives the number of calls accepted by an *offline* scheduler for the given input set. Although, the ILP maximizes the number of calls accepted over a period of time, even solving the LP-relaxed version of the formulation is very inefficient in practice in terms of CPU-time and memory requirements. In our evaluation, the GLPK [2] required several minutes on a 2.2GHz desktop processor with 2GB RAM, to solve the LP formulation to schedule an input flow. This is not acceptable for real-time application support where flows need to be accepted/rejected with a low latency of at most a few seconds.

8 Evaluation

In this section, we evaluate DelayCheck for (1) 802.15.4-based mesh network as envisioned in [20] and (2) 802.11-based mesh network as envisioned in [10].

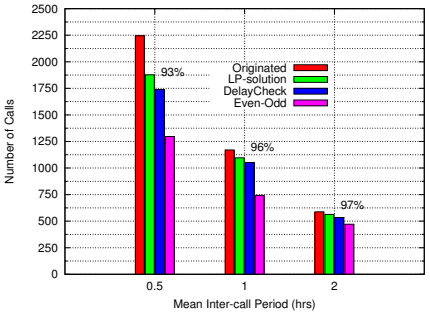

Figure 10. Voice calls accepted compard to LP-based upper bound

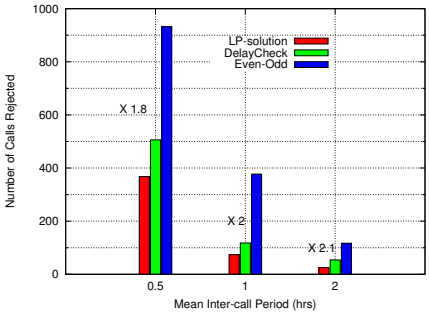

Figure 11. Voice calls rejected compard to LP-based upper bound

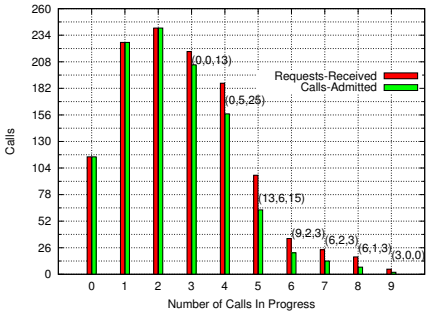

Figure 12. Admission probability of DelayCheck scheduler

802.15.4-based radio has impoverished 250Kbps of data rate and hence voice delay constraint becomes challenging to meet in such low data rate mesh networks. The 802.15.4-based mesh thus acts as stringent test case for the performance of DelayCheck. We simulate voice based wireless mesh setting envisioned in [20] over an area of 2 km x 2 km, with 125 randomly placed nodes: 25 infrastructure nodes (static backbone nodes) and 100 client nodes (handsets). The evaluation parameters considered are: 1 radio per node, maximum 16 channels per radio, exponential inter-call duration with mean varied from 30 min to 2 hrs, exponential call duration with mean of 2 min. The calls are generated from every client node and we choose source and destination pairs randomly. We consider a TDMA frame of 60ms with 8 data slots of 6ms each [20]. We assume scheduling interval to be 240ms which is 4 times the frame length. We run the simulations for 12 hrs (a half-days period).

8.1 Scheduler Performance

To evaluate performance of the scheduler, we define a metric, *scheduler efficiency*, as the number of calls accepted by DelayCheck compared to the number of calls accepted by LP-based solution. To solve the LP and find the upper bound on the number of calls accepted, we use the GLPK [2]. We model the LP GNU MathProg language. We implement even-odd slot scheduling algorithm of [18] and compare its performance with DelayCheck. Although, 802.15.4 has 16 non-overlapping channels, for scheduler testing, we assume that the scheduler has 4 channels available for scheduling.

Fig. 10 shows number of calls originated, number of calls accepted by LP-based solution, DelayCheck and even-odd scheduling for mean inter-call duration of 0.5, 1 and 2 hrs. For even-odd scheduling, we discard the accepted calls which fail to meet the deadline. As the bars show, DelayCheck has scheduler efficiency of 93% (close to upper bound) where as even-odd scheduling has scheduler efficiency of 69%. The close to upper bound output of DelayCheck is due to the consideration of joint routing, channel assignment and link scheduling in DelayCheck. Also, DelayCheck accepts 34% more calls than even-odd scheduling. This is mainly due to the basic limitation of even-odd scheduling where a link can be active in only one of the two slots. Also, DelayCheck allows scheduling across frames without limiting the scheduling (of all the links) to a single frame and thus accepts those calls which can't be scheduled in a single frame but across multiple frames within the delay budget.

Fig. 11 shows number of calls rejected by DelayCheck, in comparison to even-odd scheduling algorithm for mean inter-call duration of 0.5, 1 and 2 hrs. As the bars show, Delay check reduces the call rejections by 2 fold as compared to even-odd scheduling algorithm, even under strict delay constraint. Thus the joint routing, channel assignment and link scheduling along with delay-constraint outperforms the even-odd scheduling.

We take a more closer look at the scheduler behavior and observe the admission probability of the scheduler for a new call request. We consider the data from mean inter-call duration of 1 hr. Fig. 12 plots number of simultaneous calls in progress on x-axis and the number of calls accepted along with the number of call requests received; on y-axis. The bar corresponding to calls admitted also shows the reasons for rejection of the call requests. The numbers in the bracket indicate call rejection due to (1) no slot available (2) no channel available (3) delay budget exceeded. For example, when 3 calls are in progress, DelayCheck admits 94% of the calls. The CAC module of DelayCheck rejects the other 6% calls due to delay constraint being exceeded. This implies that although the slot and channel resources were available at the network nodes, DelayCheck did not find any feasible path through these resources which meets the delay constraint. Also as the number of simultaneous calls increases, the resource consumption in the network also increases. For example, when DelayCheck has 6 calls in progress, it admits only 64% of calls. 9 calls are rejected due to non-availability of slots, 2 due to non-availability of channels and 3 due to delay budget getting exceeded.

8.2 Effect of channels

Fig. 13 shows percentage of calls accepted as the number of channels available for scheduling are increased from 1 to 8. From the graph, for 0.5 hr inter-call duration, as the number of channels available increase to 4, the percentage of calls accepted to jump to 78% from 44%. This justifies the need for multi-channel capability of the scheduler in a multi-hop mesh setting like Lo^3 [20].

8.3 DelayCheck for 802.11 mesh

As in Sec. 2, we apply DelayCheck to 802.11-based mesh setting in Fractel [12]. Fractel has a TDMA frame length

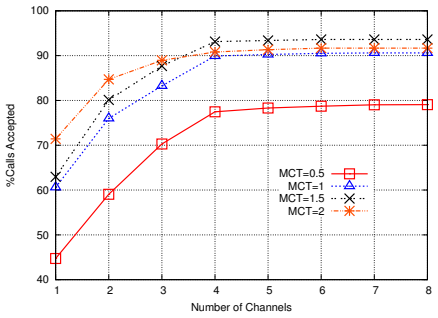

Figure 13. Effect of number of channels on scheduler

of 200ms with 100 data slots. We assume that 3 orthogonal channels are available for scheduling. We consider two algorithms as mentioned in Sec. 2, interference-aware CAC [26] and even-odd scheduling [18]. We increase the load in the system by increasing the number of simultaneous calls handled by the scheduler, by generating calls through 500 client nodes. Fig. 1, in Sec. 2, shows the behaviour of DelayCheck, interference-aware CAC, and even-odd scheduling in terms of worst case end-to-end delay of an accepted voice call. As can be observed, DelayCheck always schedules voice calls within the deadline of 250ms. Moreover, the number of voice calls admitted (not shown in figure) by DelayCheck are 31% more than interference-aware CAC and 26% more than even-odd scheduling. This shows that DelayCheck clearly outperforms these two algorithms in terms of scheduler efficiency as well as delay guarantee.

9 Conclusion

In this work, we considered the goal of supporting the *maximum number of voice calls* in a TDMA-based multi-radio, multi-channel mesh network. We presented DelayCheck, an online centralized scheduling and call-admission-control (CAC) algorithm, which effectively schedules constant-rate voice traffic in TDMA-based multi-channel mesh networks. To our knowledge, DelayCheck is the first delay-constrained scheduler which has following features as compared to past work: (1) scheduling slots across multiple frames within the delay budget (2) out of order link scheduling in multi-channel network (3) integrated routing along with channel assignment and link scheduling (4) implementable in practice for real-time applications. Through simulation based evaluations, we showed that DelayCheck meets strict (packet-level) delay guarantee for voice traffic in capacity limited multi-hop wireless networks. We compared DelayCheck with the LP-based upper bound and two state-of-the-art scheduling algorithms. DelayCheck performed remarkably well, accepting 93% of voice calls as compared to upper bound. In comparison to state-of-the-art algorithms, DelayCheck accepted 34% more voice calls. We also described implementation tricks for memory and CPU efficient operation of DelayCheck.

References

[1] Firetide, reliable connectivity anywhere. http://www.firetide.com/caseStudies.aspx.

[2] Gnu linear programming kit. http://www.gnu.org/software/glpk/.

[3] IEEE 802.16 WirelessMAN. `http://www.ieee802.org/16/`.

[4] Mean Opinion Score Calculator. `http://www.davidwall.com/MOSCalc.htm`.

[5] Nortel mesh networks. nortelnetworks.com/solutions/wrlsmesh/.

[6] Roofnet: An experimental 802.11b/g mesh network. http://pdos.csail.mit.edu/roofnet/.

[7] Ieee standard for local and metropolitan area networks part 16: Air interface for fixed broadband wireless access systems. *IEEE standard 802.16-2004*, October 2004.

[8] I. F. Akyildiz and X. Wang. A survey on wireless mesh networks. *IEEE Communications Magazine, vol. 43, no. 9, s23-s30*, Sept. 2005.

[9] M. Alicherry, R. Bhatia, and L. r. Li. Joint channel assignment and routing for throughput optimization in multi-radio wireless mesh networks. In *MOBICOM*, 2005.

[10] K. Chebrolu and B. Raman. FRACTEL: A Fresh Perspective on (Rural) Mesh Networks. In *NSDR*, Sep 2007. A Workshop in SIGCOMM 2007.

[11] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graph. In *Discrete Mathematics, Volume 86 , Issue 1-3*, December 1990.

[12] A. Dhekne, N. Uchat, and B. Raman. Implementation and Evaluation of a TDMA MAC for WiFi-based Rural Mesh Networks. In *NSDR'09*, Oct 2009.

[13] P. Djukic and P. Mohapatra. Soft-TDMAC: A Software TDMA-based MAC over Commodity 802.11 hardware. In *INFOCOM'09*, Apr 2009.

[14] P. Djukic and S. Valaee. Delay aware link scheduling for multi-hop tdma wireless networks. *IEEE/ACM Trans. Netw.*, 17(3):870–883, 2009.

[15] V. Gabale, B. Raman, K. Chebrolu, and P. Kulkarni. Lit mac: Addressing the challenges of effective voice communication in a low cost, low power wireless mesh network. In *ACM DEV, Accepted for publication*, 2010.

[16] V. Gabale, B. Raman, and A. Chiplunkar. On delay-constrained scheduling in multi-radio, multi-channel wireless mesh. *CSE Tech Report*, 2010.

[17] D. Ghosh, A. Gupta, and P. Mohapatra. Scheduling in multihop wimax networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 12(2):1–11, 2008.

[18] G. Narlikar, G. Wilfong, and L. Zhang. Designing multihop wireless backhaul networks with delay guarantees. In *INFOCOM*, 2006.

[19] B. Raman and K. Chebrolu. Design and Evaluation of a new MAC Protocol for Long-Distance 802.11 Mesh Networks. In *11th Annual International Conference on Mobile Computing and Networking paper (MOBICOM)*, Aug/Sep 2005.

[20] B. Raman and K. Chebrolu. Lo^3: **Lo**w-cost, **Lo**w-power, **Lo**cal Voice and Messaging for Developing Regions. In *NSDR'09*, Oct 2009.

[21] A. Raniwala, K. Gopalan, and T.-c. Chiueh. Centralized channel assignment and routing algorithms for multi-channel wireless mesh networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 8(2):50–65, 2004.

[22] A. Sahoo and P. Goyal. A scheduling and call admission control algorithm for wimax mesh network with strict qos guarantee. In *Comsnet*, 2010.

[23] A. Sen and M. L. Huson. A New Model for Scheduling Packet Radio Networks. In *INFOCOM*, 1996.

[24] W.C.Hardy. Voip service quality: Measuring and evaluating packet switched voice. In *McGraw-Hill*, New York, Dec. 2002.

[25] H. Yu, P. Mohapatra, and X. Liu. Dynamic channel assignment and link scheduling in multi-radio multi-channel wireless mesh networks. In *MobiQuitous*, 2007.

[26] H. Yu, P. Mohapatra, and X. Liu. Channel assignment and link scheduling in multi-radio multi-channel wireless mesh networks. *Mob. Netw. Appl.*, 13(1-2):169–185, 2008.

[27] J. Zou and D. Zhao. Real-time cbr traffic scheduling in ieee 802.16-based wireless mesh networks. *Wirel. Netw.*, 15(1):65–72, 2009.