On the schedulability of deadline-constrained traffic in TDMA Wireless Mesh Networks

P. Cappanera[1], A. Lori[2], G. Stea[3], G. Vaglini[3]

[1] Dip. di Sistemi e Informatica, University of Florence, Via S. Marta 3, 50139 Firenze – Italy, paola.cappanera@unifi.it

[2] KKT, Via Madonna del Piano, 6, 50019 Sesto Fiorentino (FI) – Italy, alessandro.lori@kkt.it

[3] Dip. di Ingegneria dell'Informazione, University of Pisa, Largo L. Lazzarino 1, 56122 Pisa – Italy, {g.stea, g.vaglini}@iet.unipi.it

***Abstract*— In this paper we evaluate the schedulability of traffic with arbitrary end-to-end deadline constraints in Wireless Mesh Networks (WMNs). We formulate the problem as a mixed integer-linear optimization problem, and show that, depending on the flow aggregation policy used in the network, the problem can be either convex or non-convex. We optimally solve the problem in both cases, and prove that the schedulability does depend on the aggregation policy. This allows us to derive rules of thumb to identify which policy improves the schedulability with a given traffic. Furthermore, we propose heuristic solution strategy that allows good suboptimal solutions to the scheduling problem to be computed in relatively small times, comparable to those required for online admission control in relatively large WMNs.**

Keywords—Link Scheduling; Wireless Mesh Networks; Real-time Traffic; Network Calculus

I. Introduction

Wireless Mesh Networks (WMNs) [1] are used for providing broadband access to mobile clients located at the edge of wireline networks, or in remote, rural, or non-cost-effective areas, e.g. offices and home environments. In WMNs, end-users are served by stationary *mesh routers*, connected through wireless links. Moreover, some mesh routers are generally connected to the Internet through wires, and thus act as gateways for the entire WMN. Communication between mesh routers is multi-hop, with intermediate routers acting as relays for endpoints not in the transmission range of each other. Many of the radio resource management issues in a WMN are common to multi-hop wireless networks. However, problems such as energy consumption (typical of ad hoc networks) are no longer an issue and a centralized network management can be used (as opposed to the distributed approaches of ad hoc networks), in which nodes are coordinated by a network entity exploiting global knowledge of the topology and additional conditions.

Each mesh router transmits its packets in broadcast. All the mesh routers tuned on the same frequency and within the transmission range receive such packets. To avoid signal interference, *link scheduling* is used to guarantee conflict-free operation in the context of Time Division Multiple Access (TDMA, [2]), where time is slotted and synchronized. Through link scheduling, only sets of non-interfering links are activated simultaneously in each slot. WMNs are already and will be supporting real-time traffics, such as voice, video, or traffic control, whose bit rate is often highly variable, and which require firm guarantees on their maximum *end-to-end delay*. Cross-layer approaches where link scheduling and routing are jointly addressed have been extensively studied in the past few years due to their application to TDMA MAC protocols [3]-[21]. However, few works have considered *arbitrary end-to-end delay bounds* as *constraints* on link scheduling before. Instead, most of the available works ([3]-[11]) compute schedules constrained by the flows' *rates*. While this approach has the obvious benefit of utilizing links efficiently, it is certainly not enough to guarantee that arbitrary pre-specified delays are met. Moreover, among the works that compute link schedules based on delays (e.g. [12]-[21]), most only take into account the *sum of the waiting times* due to TDMA scheduling, whereas this is only one component - and not necessarily the largest one - of the end-to-end delay, which also includes *queuing*. Accordingly, those algorithms often compute delay-infeasible schedules (and largely so), even when delay-feasible solutions exist.

Following our previous work [18]-[21], we consider TDMA WMNs with flows constrained by *leaky bucket* regulators, whose delay constraints are *arbitrary*, i.e., not linked to their rate requirements. We assume that shortest-path routing to the Internet gateways is in place. In that setting, the link scheduling problem can be formulated as a mixed integer-nonlinear problem, which we can solve *optimally*: in other words, we can compute a link schedule that guarantees the required delay bounds whenever it is possible to do so. The objective to be minimized is the maximum *delay violation*, (i.e. the maximum difference between the worst-case delay and the requested deadline). As shown in [20], this leads to optimal schedules that are also *robust*, i.e. such that the parameters of some flows can be varied (even by large amounts) with limited impact on the actual delays.

In this paper, we show that the schedulability of a set of flows *depends on their aggregation policy within the network*.

Flows may/may not be able to meet their deadline depending on whether they are scheduled in isolation (i.e., buffered at different queues, as in the IntServ framework [22]), or aggregated (i.e., buffered in the same FIFO queue, as in the DiffServ framework [23]). In this last case, we distinguish the case when flows are aggregated *only* at their path ingress node (i.e., when the aggregate they belong to is the same on all nodes of their common path), and when aggregation follows the *shortest-path tree*, i.e. uplink flows are aggregated as they travel towards their destination gateway, whereas downlink flows coming from different gateways are aggregated as they progress toward the same destination mesh router. We show that there are indeed cases when the aggregation policy determines the schedulability, and we devise rules of thumb to suggest which aggregation scheme may best suit a given traffic scenario.

Again depending on the flow aggregation policy, the nature of the link scheduling problem may be different: more specifically, when flows are aggregated *progressively*, constraints are non-convex, whereas they are convex in the other cases. When we deal with convex constraints only, optimal schedules can be computed for networks of up to several tens of nodes (i.e., more than 40) in minutes or hours, i.e. times that are affordable in a resource provisioning timescale perspective. Otherwise, the computation is limited to a few nodes (i.e., up to 15), and rapidly explodes beyond that figure. Therefore, we consider trading optimality for computation time, so as to make computations fast enough for online admission control in a dynamic environment. We describe a heuristic which allows suboptimal – but still practically good – schedules to be computed in short times, given an estimate of the flows' rate along the links. We also show that the heuristic devised for the case with convex constraints yields good performance also with non-convex constraints.

The rest of the paper is organized as follows: Section II explains the system model. In Section III we introduce link scheduling and delay constraints and formulate the optimization problems, distinguishing the various aggregation frameworks. We compare the optimal solutions in Section IV. Section V tackles the problem of computing a link schedule fast enough for online admission control. The related work is addressed in more detail in Section VI. We report conclusions and highlight directions for future work in Section VII.

II. System Model

We now describe the assumptions for our analysis. A table of notation is reported in the Appendix for ease of reference. We assume that each mesh router is equipped with a *single* time-slotted channel for data transmission (control transmissions are assumed to be out of band). *Transmission slots* of a fixed duration T_s are grouped into *frames* of N slots, periodically repeated every $N \cdot T_S$ time units. This happens, for instance, in 802.16 networks, where the frame length is usually set to 5 ms. Each slot is assigned to a set of non-interfering links through *conflict-free link scheduling*, which will be described in detail later on: at each slot, a subset of links may be activated for transmission only if no conflicts occur at the intended receivers, i.e., if the receiver can correctly decode the transmissions. The WMN is modeled through a *connectivity graph*, $G=(V,E)$, whose nodes $V=\{v_1,\cdots,v_n\}$ are mesh routers and whose edges $E=\{e_1,\cdots,e_m\}$ are *directed* links connecting a transmitter to the nodes within transmission range from it. The connectivity graph is a *logical* representation of the WMN, which can be derived from the physical WMN topology once the transmit powers, antenna gains, node distances and path loss are known. For instance, Fig. 1, left pictures a situation where the transmission range of node 6 is such that 7 and 4 do not hear it, whereas 3 does[1]. If node 6's transmission range is increased (e.g., by boosting its transmission power), the connectivity graph may eventually include either or both the links from 6 to 7 and 4.

The fact that some sets of links are not allowed to transmit simultaneously is modeled through *conflicts*. For each edge of the network $e \in E$ we define a *conflicting set* of edges $\mathcal{I}(e)$ which includes all the edges that will not transmit simultaneously with e *and* e itself; mutual exclusion in that set is straightforwardly defined as follows:

$$\sum_{i\in\mathcal{I}(e)} x_i(t) \le 1 \text{, if } e \text{ is active in slot } t=1,2,...,N \text{ ,}$$

where $x_e(t)$ is a binary variable, such that $x_e(t)=1$ if link $e \in E$ is active in slot t, and 0 otherwise. The condition requires that, if edge e is active in slot t, $\mathcal{I}(e)$ contains one active edge only (i.e., edge e itself). We translate the above condition to a *conflict graph* $G_c=(E,C)$, shown in Fig. 1, right, whose nodes represent *links* of the connectivity graph and whose edges $C=\{c_1,\ldots,c_r\}$ model the conflicts between links. In the latter, two types of conflicts are modeled:

- *Hard* conflicts: if link (i, j) is active, all the links having either i or j as an endpoint must not transmit, since otherwise communication would be impossible. This models the fact that links are half-duplex and that each node can transmit to/receive from at most one neighbor in a slot. For instance, in Fig. 1, right, link (0,1) conflicts with (3,0) and (1,4) (half-duplex), and with (0,3) (one recipient at a time). Hard conflicts can be readily inferred from the connectivity graph.

- *Soft* conflicts: two links - not having endpoints in common - may be in each other's immediate vicinity, so that scheduling them simultaneously would cause too much

[1] The connectivity graph of Fig. 1 is not meant to represent a real-life case. It includes fewer links than it would be reasonable to assume in order to keep the resulting conflict graph simple enough.

reciprocal interference and decrease their achievable rates. For instance, in the conflict graph of Fig. 1, right, edge (0,1)-(3,4) implies that these two links should not be activated simultaneously. Soft conflicts cannot be inferred *sic et simpliciter* from the connectivity graph, since they depend on physical characteristics that are not incorporated in the latter. Two such conflicts are reported in dashed lines in Fig. 1, right, as an example.

Set $\mathcal{I}(e)$, including both hard and soft conflicts, can be easily obtained by retrieving the one-hop neighborhood of e in the conflict graph.

We assume that the conflict graph is given, and that each link e has a constant transmission rate W_e, representing the maximum rate at which correct reception is guaranteed – based on whatever interference model, e.g., SINR thresholds [24] – when all the non-conflicting links (i.e., those in $E \setminus [\mathcal{I}(e) \setminus e]$) transmit simultaneously.

Determining link rates and the conflict graph (and, especially, "soft" conflicts) is outside the scope of this paper. In fact, this constitutes part of network planning, along with positioning nodes, etc., and the solution to this problem are likely to span long periods of times (weeks or more), whereas we deal with admission of flows, something which happens at much faster timescales (i.e., seconds to hours). For the sake of completeness, we observe that both the link rates and the conflict graph depend on the transmission power at *every* link and on the channel gains. The problem of computing transmission powers, a set of conflicts and the resulting link rates can be solved in several ways and according to several objectives (e.g., maximizing the minimum overall rate, or the minimum cardinality, of a non-conflicting set, etc.). Our schemes, presented in the next section, can work with any such solution.

The WMN contains one or more gateway nodes, which act as sources for downlink traffic and sinks for uplink traffic. We assume that destination-based, shortest-path routing is enforced, and only consider communications between the mesh nodes and the gateways.

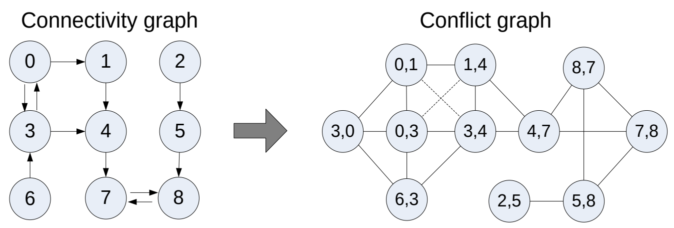


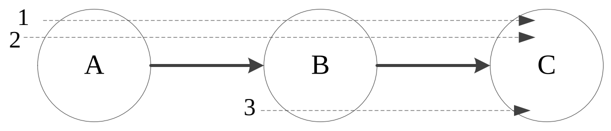
Fig. 1 - Logical connectivity graph (left) and conflict graph (right) of a WMN

The WMN is traversed by *flows*, i.e. distinguishable streams of traffic. Each flow has a delay constraint, specified as a *required end-to-end delay bound* δ. At the ingress node, its arrivals are constrained by a *leaky-bucket* shaper, with a *burst* σ and a *sustainable rate* ρ. The path of a flow q, i.e. the set of links it traverses from the source to the destination, is denoted as P_q. As far as buffering at output links is concerned, we consider three different options, shown in Fig. 2: a *per-flow queuing* framework, where packets of each flow are buffered separately at each link. Thus, a link handles as many queues as the flows traversing it. Alternatively, in a *per-path queuing* framework, packets of flows traversing the *same path*, i.e. the same set of links (e.g., flows 1 and 2 in Fig. 2), are buffered *in a single queue*. This way, a link handles as many queues as there are paths traversing it. As a third option, we consider *per-exit-point queuing*. By "exit point" we mean the last node in the WMN to be traversed, i.e. the gateway in the uplink or the egress mesh router in the downlink. In this last case, a link has as many queues as the number of exit points that it is connected to. For instance, uplink flows are progressively aggregated as they travel towards their gateway. On the other hand, downlink flows from several gateways, destined to the same mesh router, will be aggregated as they travel the shortest-path tree in the downlink direction. In all three cases, we assume that buffers are FIFO.

The three above mentioned queuing frameworks can be implemented by using standard components/protocols of IP-based networks. More specifically, we need mesh routers to be able to classify and queue packets at an output link as follows:

- based on their 5-tuple (i.e., source and destination IP addresses, protocol, and source and destination ports), for per-flow queuing; the above information can be read in the IP header.
- Based on the exit point in the WMN for per-exit-point queuing. The latter can be inferred from the IP destination address of the packet *and* the mesh node IP routing table. In fact, if a mesh router can route a packet with an IP destination address *x*, then it also knows the WMN egress point for that packet.
- Based on both the IP addresses of the entry and exit point for per-path queuing. This could be achieved via IP-over-IP tunneling, using the source/destination address of the outer IP header to classify packets. Alternatively, if IPv6 is used, the entry and exit points could be easily encoded in the 20-bit *flowID* field of IPv6. Finally, MPLS could be used for the same purpose.

Note that MPLS, the IPv6 *flowID* field, and IP-over-IP tunneling could be used to speed up classification of packets under per-flow and per-exit-point queuing as well.



Queuing	Link	A-B	B-C

Per-flow	2	3
Per-path	1	2
Per-exit-point	1	1

Fig. 2 - Queuing frameworks and related number of queues at each link

III. Problem Formulation

The purpose of this paper is to compute a conflict-free schedule which does not violate the required delay bounds whenever it is possible to do so. First, we identify the constraints that ensure the conflict-free property, which are common to all three formulations. Delay feasibility constraints, instead, depend on the queuing framework, and they lead to different problem formulations.

Given a conflict graph C, only conflicts between *active links*, i.e. those with a non-null flow, have to be considered. We thus define $C_f \subseteq C$ as the subset of conflicts involving active links:

$$C_f := \{(i,j) \in C : f_i > 0 \text{ and } f_j > 0\},$$

where f_i denotes the flow going through link i. For instance, links that carry no traffic because of routing are not considered into C_f.

We define an *activation offset* π_e for link e, $0 \le \pi_e \le N$, i.e., the time at which link e's transmission opportunity starts, and a *transmission duration* Δ_e the link is allowed to transmit for. Fig. 3 shows the relevant quantities, plus others that will be defined in the following. Since time is slotted, π_e and Δ_e variables are non-negative integers. The fact that a link has *one* transmission opportunity within a frame, though limiting, ensures that link scheduling maps can be kept compact.

The schedule must ensure the *conflict-free* condition: while a link is transmitting, all conflicting links must refrain from transmitting. For any pair of links i and j which are neighboring nodes in C_f we have:

- if j transmits after i, it must wait for i to complete the transmission, i.e. $\pi_i - \pi_j + \Delta_i \le 0$.
- Otherwise, the symmetric inequality holds, i.e. $\pi_j - \pi_i + \Delta_j \le 0$

In order to linearize the combination of the above constraints, we introduce a binary variable o_{ij}, $(i,j) \in C_f$, called *conflict orientation*, which is 1 if i transmits after j and 0 otherwise. The left-hand side of both the previous inequalities can thus be upper bounded by N regardless of the relative transmission order, as π_i and Δ_i belong to $[0,N]$. This completes the formulation of the *conflict-free constraints*, which are necessary and sufficient conditions:

$$\begin{array}{ll} \pi_i - \pi_j + \Delta_i \le N \cdot o_{ij} & \forall (i,j) \in C_f \\ \pi_j - \pi_i + \Delta_j \le N \cdot (1 - o_{ij}) & \forall (i,j) \in C_f \end{array} \tag{1}$$

For a schedule to be valid, each link must also complete its transmission within the frame duration, i.e.:

$$\pi_e + \Delta_e \le N \quad \forall e \in E. \tag{2}$$

From now on, we will denote with $\overline{S}$ the feasible region defined by inequalities (1) and (2), and summarize the above two sets of inequalities by writing $\{\pi_e, \Delta_e\} \in \overline{S}$.

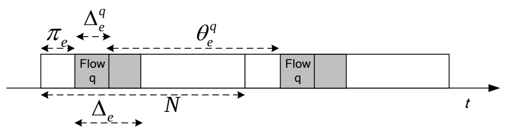


Fig. 3 - Relevant quantities in link scheduling

Beside those arising from interference, additional constraints are needed to keep into account the end-to-end delay requirements. In this section we expose them and formulate the problem of delay-constrained link scheduling. We first deal with per-flow and per-path queuing frameworks, between which many similarities exist, and then describe the problem under per-exit-point queuing in a separate sub-section. The framework developed in this paper relies on basic Network Calculus concepts, i.e. *arrival curve*, *service curve* and *delay bound*. Interested readers can find background in [25], from which we also borrow notation.

A. Per-flow and per-path queuing

In per-flow queuing, each link e transmits traffic of several flows on each activation. We can therefore partition the link's Δ_e among them, i.e. $\Delta_e = \sum_{q:e \in P_q} \Delta_e^q$. Δ_e^q is the link activation quota reserved for flow q, which need *not* be an integer, since when a link e is activated it can switch among backlogged queues regardless of slot boundaries. We assume that backlogged flows traversing e are always served in the same (arbitrary) local order, and we call I_e the ordered set of the flow indexes. We assume that each backlogged flow q is served for *no less* than Δ_e^q. If a flow is idle, its service time can be exploited by other backlogged flows at e, as long as the transmission from any flow z starts within at most $\sum_{x \in I_e : x < z} \Delta_e^x$ from the activation of link e.

Therefore, flow q has a *guaranteed rate* equal to $R_e^q = W_e \cdot \Delta_e^q / N$ at link e, W_e being the transmission rate of that link. Since each flow has a single transmission opportunity in a frame, then the maximum *inter-service time* for that flow is $\theta_e^q = \left(N - \Delta_e^q\right) \cdot T_S$, irrespective of the local ordering at each link. Thus, each link of a mesh router is a *rate-latency* server [25] for the flows traversing it, with a rate R_e^q and a latency θ_e^q. Accordingly, each flow has an end-to-end delay bound equal to (see [25]):

$$D_q = \begin{cases} \sum_{e \in P_q} \theta_e^q + \sigma_q / R_{\min}^q & \text{if } \rho_q \le R_{\min}^q \\ \infty & \text{otherwise} \end{cases}, \tag{3}$$

where $R_{\min}^q = \min_{e\in P_q}\left\{R_e^q\right\}$.

The above bound is *tight*, i.e. D_q is actually the maximum delay under the hypotheses [25]. The first addendum in (3) is called *latency delay*, and it is due to link scheduling and arbitration of the flows at the links. The second is called *burst delay*, and it is the time it takes for the flow's burst to be cleared at the minimum guaranteed rate.

Given a traffic characterization, our aim is to find a conflict-free schedule which is also feasible from a delay point of view. This is indeed a *feasibility* problem, i.e. it is solved if *a* feasible solution is found. However, in order to have a measure of the quality of the computed solution, we transform it into a min-max *optimization* problem, where the objective function (to be minimized) is the maximum *delay violation* $V_{\max} \triangleq \max_{q\in Q}\left\{D_q - \delta_q\right\}$. We will call this problem the *Minimum Max Violation Problem* (MinMVP)

$$\begin{array}{llll}
\min & V_{\max} & & \\
\text{s.t.:} & & & \\
& D_q - \delta_q \le V_{\max} & \forall q\in Q & (i) \\
& R_{\min}^q \le W_e \cdot \Delta_e^q / N & \forall e\in P_q, \forall q\in Q & (ii) \\
& \Delta_e^q \ge N\cdot \rho_q / W_e & \forall e\in P_q, \forall q\in Q & (iii) \\
& \Delta_e \ge \sum_{q:e\in P_q} \Delta_e^q & \forall e\in E & (iv) \\
& \left\{\pi_e, \Delta_e\right\} \in \overline{S} & & (v) \\
& o_{ij} \in \left\{0,1\right\} & \forall (i,j)\in C_f & (vi) \\
& \pi_e, \Delta_e \in \mathbb{Z}_0^+ & \forall e\in E & (vii) \\
& R_{\min}^q \in \mathbb{R}_0^+ & \forall q\in Q & (viii) \\
& \Delta_e^q \in \mathbb{Z}_0^+ & \forall e\in P_q, \forall q\in Q & (ix)
\end{array} \qquad (4)$$

In the MinMVP we linearize the min operator in (3) by means of an additional continuous variable $R_{\min}^q$ for each flow. Constraint (*i*) defines the maximum delay violation, which is minimized in the objective. Constraint *(iii)* guarantees that all delays are finite, since enough rate is reserved for each flow at each link. Furthermore, constraint *(ii)* will be active for the flow q with the maximum violation, i.e. $R_{\min}^q = \min_{e\in P_q}\left\{W_e\cdot\Delta_e^q/N\right\}$, as $V_{\max}$ inversely depends on $R_{\min}^q$. Constraint (*iv*) defines the link activation Δ_e to be at least as large as the sum of the flow activations Δ_e^q. The link activation is also used in the *feasible region* constraints (*v*).

Clearly, the solutions to the feasibility problem correspond to points where the objective function in (4) is non positive. The MinMVP formulation leads to a *Mixed Integer Non-Linear (MINLP)* problem, with convex non-linear constraints, that can be solved optimally using a general purpose MINLP solver [26]. An equivalent formulation as a *Quadratically Constrained Problem* (QCP) is also possible, which allows it to be solved using mixed integer QCP solvers [27].

Re-writing the feasibility problem as an optimization problem brings considerable advantages. First of all, we will show that it allows us to explore the schedulability region, assessing the relationships between the schedulability and the various parameters involved (i.e., flow deadlines, burst, rates). To this end, $V_{\max}$ is a good indicator of how much the WMN is *loaded*, i.e. whether it might support more traffic or tighter deadlines (or, if $V_{\max}$ is positive, which flow is the most critical). Second, as shown in [20], it yields robust schedules, such that relatively large variations in the flow bursts and rates can often be accommodated without having to compute a schedule anew.

If *per-path* queuing is used, instead, a set of flows $q_1,...,q_k$ traversing the same path can be modeled as a *single* flow. In fact, a well-known result regarding leaky buckets is the following:

Property 1: *if two leaky-bucket shaped flows* 1 *and* 2 *are aggregated at a node, then their aggregate is still a leaky bucket shaped flow, with parameters* $\sigma = \sigma_1 + \sigma_2$ *and* $\rho = \rho_1 + \rho_2$. *Furthermore, the delay bound* (3) *computed for the aggregate flow is in fact the worst-case delay for each flow.*

This means that all the above modeling and the formulation of the MinMVP still hold, provided that the flow characteristics and requirements are composed as follows:

- the required delay bound for the aggregate is $\delta = \min_{1\le q\le k}\left\{\delta_q\right\}$;
- the leaky bucket parameters for the aggregate are $\sigma = \sum_{1\le q\le k}\sigma_q$, $\rho = \sum_{1\le q\le k}\rho_q$.

From the network management standpoint, under per-path queuing the number of queues managed at each link is reduced with respect to the per-flow case, due to the fact that several flows are aggregated.

Finally, we remark that (4) can be used to schedule either uplink or downlink flows, or both simultaneously. Moreover, the modeling works regardless of the number of gateways in the WMN.

B. Per-exit-point queuing

We now describe delay constraints under per-exit-point queuing. We initially assume that the WMN has *one* gateway, and focus on *uplink* traffic directed towards that single gateway. Such assumptions are only required to simplify the notation, and will be removed in the next sub-section.

Under per-exit-point queuing, all flows are buffered FIFO in the same queue at each link, i.e. they are aggregated as they progress towards the gateway node. This defines a *sink-tree* topology, where paths can be tagged using the source node as a label without any ambiguity (i.e., path P_i goes from node *i* to the gateway). Call N_i the number of nodes traversed by path P_i. In order to denote a node's position in a path, we define function $l_i(h)$ that returns the label of the *h^th^* node in path P_i, $1\le h\le N_i$, and function $p_i(z)$ that returns the position of node *z* along path P_i, $p_i(\cdot) = l_i^{-1}(\cdot)$. Given two paths P_i and P_j

, $i \neq j$, their traffic is aggregated at the first common node $M_{i,j}$. We say that the two paths *merge* at that node, i.e. $M_{i,j} = l_i(a) = l_j(b)$, for some a,b such that $1 \le a \le N_i$ and $1 \le b \le N_j$ and $l_i(a-1) \neq l_j(b-1)$. Without loss of generality, we assume the nodes are labeled so that each path is an increasing label sequence from the ingress node towards the egress one. It is worth noting that if two paths P_i and P_j merge at node $x = l_i(h) = l_j(k)$, they share all nodes from the node x up to the gateway. With reference to Fig. 4, we have $P_0 = \langle 0,5,9 \rangle$, $P_3 = \langle 3,5,9 \rangle$. Hence, $l_0(1) = 0$, $l_3(1) = 3$, $l_0(2) = l_3(2) = 5$, $l_0(3) = l_3(3) = 9$, and $M_{0,3} = 5$.

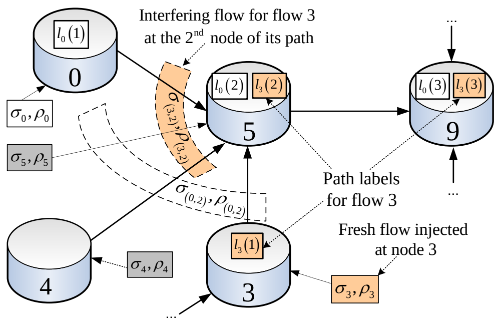


Fig. 4 - Relevant quantities for paths P_0 and P_3

Each link e is activated once in the frame for an activation of Δ_e time units. Therefore, each link is itself a *rate-latency server*, with a *guaranteed rate* equal to $R_e = W_e \cdot \Delta_e / N$ and a *latency* $\theta_e = (N - \Delta_e) \cdot T_S$. We now describe the formulas for computing the worst-case delay for a flow in a sink-tree network of rate-latency nodes. The complete derivation process is shown in [28], to which the interested reader is referred for the details. Let us first introduce two preliminary results:

Theorem 2 ([28], Theorem 4.15)**:** *Consider a node x and let Ξ be the set so that $x = l_i(h_i)$ for each node $i \in \Xi$ and some $1 \le h_i \le N_i$. Let σ_i, ρ_i be the leaky-bucket parameters for the fresh flow entering node i. Then, the aggregate flow at the output of node x is leaky-bucket shaped, with a burstiness s_x and a sustainable rate r_x as follows:*

$$s_x = \sum_{i \in \Xi} \left[\sigma_i + \rho_i \cdot \sum_{1 \le j \le h_i} \theta_{l_i(j)} \right], \; r_x = \sum_{i \in \Xi} \rho_i \, , \tag{5}$$

and the values in (5) *are tight output constraints at x.*

Let us now consider a sink tree as the one shown in Fig. 4, and let us focus on path P_i. By Property 1, we can assume that one fresh flow enters each path P_i without loss of generality. Accordingly, we denote with σ_i, ρ_i the leaky-bucket parameters of that flow and with δ_i its required delay bound. If no fresh flow is injected at node *i*, we can assume that a "null flow", with $\sigma_i = 0$, $\rho_i = 0$, $\delta_i = +\infty$, is injected in the network at that node.

Based on Property 1 and Theorem 2, we can also model the aggregate traffic that joins path P_i at node $l_i(h)$, composed of both the flow arriving from upstream nodes and the fresh flow injected at node $l_i(h)$ itself, as a *single flow*. We call it the *interfering flow* (i,h), and we denote its leaky-bucket parameters as $\sigma_{(i,h)}, \rho_{(i,h)}$. The following property shows how to compute the leaky-bucket parameters of an interfering flow from node parameters:

Property 3: *In a path P_i, for $2 \le h \le N_i$, it is:*

$$\sigma_{(i,h)} = s_{l_i(h)} - \left[s_{l_i(h-1)} + r_{l_i(h)} \cdot \theta_{l_i(h)} \right], \; \rho_{(i,h)} = r_{l_i(h)} - r_{l_i(h-1)} .$$

Note that, in general, although for two different paths P_i and P_j, $l_i(h) = l_j(k)$, interfering flows (i,h) and (j,k) may not be the same (hence we need a pair of subscripts for denoting them). In fact, from Property 3, given a node $x = l_i(h) = l_j(k)$, $(i,h) \equiv (j,k)$ if and only if there exist a node $y < x$ such that $y \in P_i, P_j$. In the network of Fig. 4, we can see that paths P_0 and P_3 merge at node $5 = l_0(2) = l_3(2)$ and $(0,2) \neq (3,2)$ (both being easily identifiable through color codes in the figure). Furthermore we define flow $(i,1)$ as the sum of the output flows at all children of node i (if there are any) and the fresh traffic entering node i. For instance, at a leaf node, $\sigma_{(i,1)} = \sigma_i$ and $\rho_{(i,1)} = \rho_i$.

Having said this, we now show how to compute the worst-case delay for a flow along a path. First of all, in order for queues not to build up indefinitely at a node x, the following *stability condition* must be ensured:

$$r_x^* = R_x - r_x \ge 0 \, , \tag{6}$$

where r_x^* is called the *residual rate* of node x, i.e. the rate which is not strictly necessary to sustain the admitted traffic. If (6) holds for all nodes along path P_i, the worst-case delay for the flow traversing that path is upper bounded by:

$$D_i = \sum_{h=1}^{N_i} \left[\theta_{l_i(h)} + \frac{\sigma_{(i,h)}}{CR_{l_i(h)}} \right], \tag{7}$$

where $CR_{l_i(h)}$ is the *clearing rate* at node $l_i(h)$. The latter is the rate at which a burst arriving at once at that node $l_i(h)$ leaves the gateway in a worst-case scenario.

In general, $CR_{l_i(h)}$ is a function of both the *service rate* $R_{l_i(k)}$ and the *sustainable rate of interfering flows* $\rho_{(i,k)}$ at nodes $h \le k \le N_i$. It can be computed once it is known which nodes act as *bottlenecks* for node $l_i(h)$, according to the following definition.

Definition 4: *Consider two nodes x and y, such that path P_i traverses them in that order, i.e. $p_i(x) \le p_i(y)$. Then, we say that y is a bottleneck for x if:*

$$r_y^* \le \min\left\{ r_j^* : p_i(x) \le p_i(j) < p_i(y) \right\} . \tag{8}$$

Intuitively, node y is a bottleneck for node x if its residual rate is the minimum among all nodes in the path from x to y. Note that, by definition, x is a bottleneck for itself. Call

$B_x = \left\langle b_1^x, b_2^x, \ldots b_{W_x}^x \right\rangle$ the sequence of $W_x \geq 1$ bottleneck nodes for node x, sorted in the same order as they appear in any path that traverses that node, so that $b_1^x = x$. Then, it is:

$$CR_x = R_{b_{W_x}^x} \cdot \prod_{y=1}^{W_x - 1} \frac{R_{b_y^x}}{R_{b_y^x} + \left(r_{b_{y+1}^x} - r_{b_y^x} \right)}. \qquad (9)$$

Note that, since the sequence of bottlenecks depends on rate allocation at the links, (9) (and, accordingly, (7)) are non-smooth. Non-smooth functions are very hard to treat, except at very small scales, within optimization problems. However, an alternative formulation of (9) can be given as a minimum of smooth functions, which allows us to formulate our problem in a more tractable way. The property is given by the following theorem, whose proof is in the Appendix:

Theorem 5: *Consider a sink-tree path P_x and define:*

$$\Sigma_x = \left\{ S : S \subseteq P_x, x \in S \right\}$$

i.e., the set of those subsets which also include x (note that $B_x \in \Sigma_x$). Denote with $n_S(h)$ the h^{th} node in S, with $n_S(1) = x$. Then it is:

$$CR_x = \min_{S \in \Sigma_x} \left\{ R_{n_S(|S|)} \cdot \prod_{h=1}^{|S|-1} \frac{R_{n_S(h)}}{R_{n_S(h)} + \left(r_{n_S(h+1)} - r_{n_S(h)} \right)} \right\} \qquad (10)$$

The set Σ_x grows exponentially with the length of path P_x, i.e. with the depth of the sink tree. However, since paths in a WMN are not expected to be longer than few hops, this is never a problem in practical cases. Similarly to what we have done in the per-flow and per-path queuing frameworks, we can formulate a feasibility problem and turn it in to a min-max optimization problem having the maximum delay violation as an objective, where the new delay formula is given by (7):

$$\begin{array}{lll}
\min \quad V_{\max} & & \\
\text{s.t.:} & & \\
\sum_{h=1}^{N_i} \left[\theta_{l_i(h)} + \sigma_{(i,h)} / CR_{l_i(h)} \right] - \delta_e \leq V_{\max} & \forall e \in E & (i) \\
CR_e \leq R_{n_S(|S|)} \cdot \prod_{h=1}^{|S|-1} \frac{R_{n_S(h)}}{R_{n_S(h)} + \left(r_{n_S(h+1)} - r_{n_S(h)} \right)} & \forall S \in \Sigma_e \; \forall e \in E & (ii) \\
\Delta_e \geq \sum_{i \in E : e \in P_i} N \cdot \rho_i / W_e & \forall e \in E & (iii) \\
\theta_e = (N - \Delta_e) \cdot T & \forall e \in E & (iv) \\
R_e = W_e \cdot \Delta_e / N & \forall e \in E & (v) \\
\left\{ \pi_e, \Delta_e \right\} \in \overline{S} & & (vi) \\
o_{ij} \in \{0,1\} & \forall (i,j) \in C_f & (vii) \\
\pi_e, \Delta_e \in \square_0^+ & \forall e \in E & (viii) \\
CR_e \in \square_0^+ & \forall e \in E & (ix)
\end{array} \qquad (11)$$

Note that, for this formulation, we can use the same subscript e to denote a flow (or set thereof) and a node, since all flows are aggregated at a node. Constraint (*i*) implements (7), whereas (*ii*) implements (10). The other constraints mirror those of problem (4), *mutatis mutandis*.

The above mixed integer-non-linear formulation contains *non-convex* non-linear constraints. This can be easily worked out by counterexample, i.e. by constructing instances where local optimization solvers yield different optima with different starting points, which cannot happen with convex problems. Therefore, in order to solve the above optimization problem, *global* optimization techniques are required, involving a combination of both branch-and-bound - to handle the integrality constraints on the π_e, Δ_e and o_{ij} variables - and linearization/bound reduction techniques - to deal with the non-linear non-convex constraints. We will come back to the problem of computation efficiency in Section V, after comparing the optimal schedules obtained under the different aggregation frameworks for several topologies and traffics. Hereafter, for ease of exposition, we will refer to both (4) and (11) with the name MinMVP, depending on the context. Possible ambiguities will be resolved by explicitly mentioning the related queuing framework.

C. Generalizations

We now show that the model for per-exit-point queuing only requires minor, straightforward changes to accommodate multiple gateways and downlink traffic.

If the WMN includes G gateways, $G \geq 1$, then a link e may belong to up to G shortest-path trees. For instance, with reference to Fig. 5, link 6-9 (among others) belongs to both the trees rooted at gateways G1 and G2. Nothing needs to be changed to account for multiple gateways under per-flow and per-path queuing. Under per-exit-point queuing, instead, traffic of different trees is buffered in separate queues at a link as discussed above, hence *two* queues will be provisioned at link 6-9. However, a link e is still activated *once* in a frame, for an integer duration Δ_e, regardless of the number of per-exit-point queues that it accommodates. All it takes to accommodate multiple gateways is to define P_g as the set of links belonging to gateway g's tree, and *per-exit-point durations* Δ_e^g at each link, such that:

$$\Delta_e = \sum_{g : e \in P_g} \Delta_e^g, \qquad (12)$$

thus sharing the link activation among the above queues. Therefore, under per-exit-point queuing, the multiple-gateway case can be taken into account by generalizing formulation (11) so that:

- delay constraints (*i*-*v*) are reformulated specifying a gateway superscript whenever required (e.g., Δ_e^g in place of Δ_e etc.);
- link scheduling constraints (*vi*) remain the same as in (11);

- a further constraint linking per-exit-point activations Δ_e^g to the overall link activation Δ_e is added, taken from (12), i.e.:

$$\Delta_e \geq \sum_{g:e\in P_g} \Delta_e^g ,$$

which is similar to a corresponding constraint in (4),where multiple per-flow queues are scheduled on the same link and per-flow activations are connected to the per-link activations. Reformulating the MinMVP with these constraint is thus a straightforward task, which is left to the alert reader.

Under shortest-path routing, downlink traffic injected at the gateways will follow its route to its mesh router exit point. Each of the latter will thus be the root of a (downlink) shortest-path tree, whose leaves are the gateway nodes, as shown in Fig. 6 (where symmetric link costs have been assumed, which is not a requirement, however). Therefore, the link scheduling problem is still the same as in the uplink direction, *mutatis mutandis*. From a practical point of view, it is worth noting that the number of gateways in a WMN is expectably small with respect to the number of nodes in it, and that each mesh router is unlikely to exchange traffic with them all simultaneously, in any case. Therefore, per-exit-point queuing in the downlink will lead to a link scheduling problem with possibly many, though *simpler* trees, whereas in the uplink we have fewer, but more leafy trees. Note that a link may also belong to *both* an uplink (i.e., gateway-rooted) *and* a downlink (i.e., mesh router-rooted) tree simultaneously, without this being a problem.

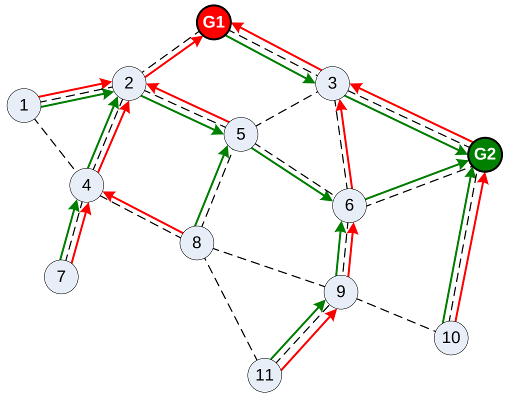


Fig. 5 – Links belonging to different shortest-path trees in a multi-gateway WMN.

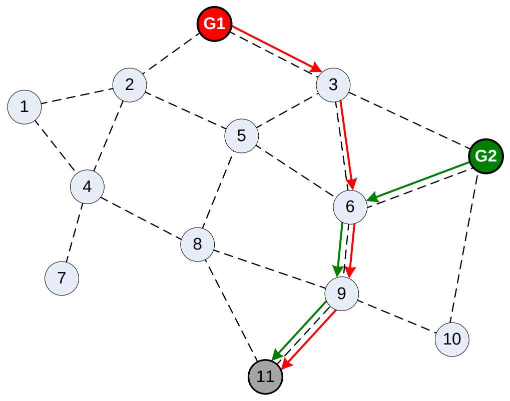


Fig. 6 – Shortest-path tree rooted at mesh router 11, downlink traffic

As a last generalization, we mention the fact that formulas (7)-(10) for computing the delay bound in a sink-tree tandem are *symmetric*, i.e., they apply also to a *source-tree* path [29]. The proof of this result is notationally heavy, and relies on the commutativity of the convolution operator. Due to the latter, given a source-tree path of rate-latency nodes, we can always build a sink-tree path that has the same delay bound. An example of two such equivalent paths is shown in Fig. 7. This symmetry could be exploited to devise a fourth queuing policy, i.e., *per-entry-point*, according to which traffic coming from the same entry point (possibly destined to different destinations) is buffered FIFO in the same queue. As soon as the routing leads two sub-flows onto different links towards their respective destinations, they are de-aggregated. With reference to Fig. 5, for instance, traffic originating from node 11 and destined to G1 and G2 would be buffered FIFO in the same queue at link 11-9 and 9-6, and then split into two flows at node 6. This queuing framework can be accommodated via straightforward modifications to the per-exit-point model, and used for both uplink and downlink traffic. We do not pursue this generalization further in the paper, since we think that its practical interest is limited.

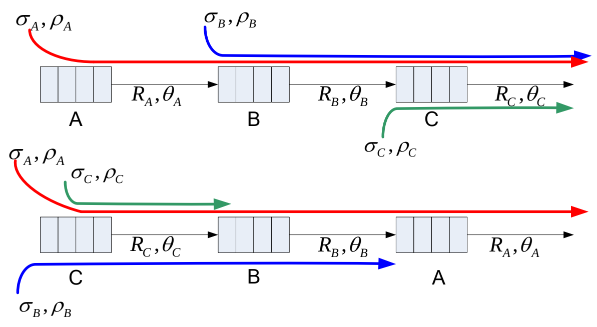


Fig. 7 – A sink-tree path (above) and a source-tree path (below), where the tagged red flow has the same delay bound

Summing up, the MinMVP problem can be formulated in a WMN with any number of gateways, traffic flowing in both the uplink and the downlink direction, and per-flow, per-path or per-exit-point aggregation, provided that shortest-path routing is in place. The MinMVP problem is mixed integer non-linear, and it is convex except under per-exit-point aggregation.

Finally, we observe that formulas (3) and (7) are such that the delay never increases when the rate of a link in the WMN is increased. More specifically, (3) is weakly monotonic with the rate of each link in the path of the flow and insensitive to the others, whereas (7) is monotonically decreasing with the rate of any link in the tree (as proved in [32]), and insensitive to the others. Thus, increasing a link e's rate with respect to W_e at some slots can only decrease $\delta_{\max}$, or have no effect at all. Link e's rate may be increased, for instance, if the set of e's interferers scheduled at slot t, call it $I_e(t)$, is smaller than its *worst-case* set of interferers (based upon which W_e had been computed before link scheduling): in this case the SINR at link e's receiver will be higher, hence the data rate could be increased accordingly. Sets $I_e(t)$ and the associated interference can be computed straightforwardly by post-processing schedules.

IV. Schedulability comparison

Being able to optimally solve the MinMVP allows us to explore the solution space, i.e. to assess how the flow and system parameters affect the schedulability. We performed several experiments, solving the MinMVP problem in a per-flow, per-path and per-exit-point queuing frameworks. For this evaluation, we consider a single-gateway WMN, whose shortest-path tree is the 15-node balanced binary tree shown in Fig. 8, with both *homogeneous* and *heterogeneous* flows. We only assume uplink traffic, since this is the most interesting case under per-exit-point queuing.

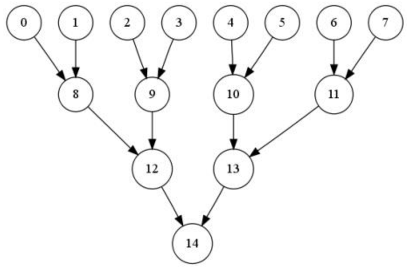


Fig. 8 – Sample binary tree

First of all, the criterion of comparison is the value of the objective function $V_{\max}$. The latter, of course, depends on the flow deadlines δ_q in turn. However, when deadlines are homogeneous (i.e. $\delta_q = \delta$), the actual value of $V_{\max}$ can be expressed as $D - \delta$, with $D = \max_q \{D_q\}$ depending on the instance of the problem. Hence, while whether a given traffic demand is schedulable ($V_{\max} \le 0$) or not ($V_{\max} > 0$) does depend on the actual value of δ, the fact that $V_{\max}^a \le V_{\max}^b$, with *a* and *b* being two different aggregation frameworks, does not. Thus, $V_{\max}^a \le V_{\max}^b$ actually implies that using *a* would allow for *smaller* values of δ (or a higher offered load) to be schedulable, and is therefore considered preferable here. Furthermore, there is no insight to be gained by varying δ as it only acts as an offset to $V_{\max}$, and for this reason we keep it constant, i.e. $\delta = 20$, until further notice.

We start with homogeneous traffic. In Fig. 9, we plot $V_{\max}$ against the rate in a scenario with 20 flows originating at each node, for two different values of the burst ($\sigma = 0$, $\sigma = 1000$), and with different aggregations. For all flows, ρ varies in $[50;650]$. Two main observations can be gathered. First of all, under per-flow and per-path aggregation, $V_{\max}$ depends minimally, if at all, on the flow rates, as long as the problem is solvable. This is because D_q does not depend on ρ_q in (3), as long as $\rho_q \le R_{\min}^q$. However, the minimum $V_{\max}$ is obtained when each link has the largest possible rate. Hence, modifying the *flow* rates does not change anything, at least until they grow so large that D_q becomes infinite. To prevent the reader from drawing hasty conclusions, we also observe that, if *routing* was put into the framework (i.e., in a case study where a flow has more than one path to a destination), the outcome would instead depend more heavily on the flow rates, as they would probably influence the path a flow takes. Furthermore, the performance under per-flow aggregation is always remarkably worse than under per-path aggregation. This is because, as flows are aggregated, the *latency* of the aggregate is generally smaller than the latency of the single flows at each node, since a larger transmission duration is given to the aggregate. This is further confirmed by the following experiment: we inject a *constant* load of traffic at each node (i.e., same overall σ, ρ), fractioned in 1 to 50 flows. Fig. 10 reports $V_{\max}$ against the number of flows per node, and confirms that the gain with a per-path framework increases with the number of flows that are aggregated. Thus, aggregating a large number of smaller flows (besides leading to more manageable implementations, due to fewer queues being required) improves the overall performance.

Second, when per-exit-point aggregation is used, Fig. 9 shows $V_{\max}$ being linearly increasing with the rate. The slope is almost constant, and the values of σ determines the offset. This is because rates intervene in the computation of the delay bound in (7) through (5) and (9). Moreover, it turns out that – for a given burst σ – there exists a value of ρ below which per-exit-point aggregation outperforms per-path aggregation, i.e. it yields a smaller $V_{\max}$. This boundary value occurs at smaller rates as the burst increases. In Fig. 11, we plot the curve where $V_{\max}$ has the same value in both per-path and per-exit-point frameworks in a (σ, ρ)-plane. The curve is a decreasing line, whose best fit is the following:

$$\sigma = [-4.1364 \cdot \rho + 2035.6]^{+} \tag{13}$$

Below the latter, per-exit-point aggregation yields better results. On the other hand, in the region above the curve per-path aggregation is more effective.

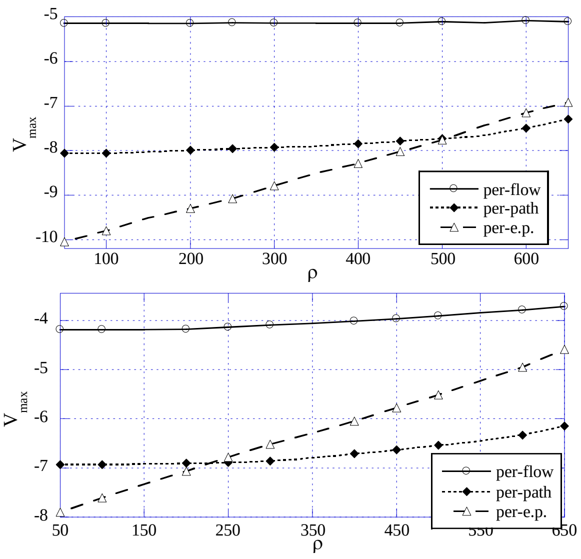


Fig. 9 - Homogeneous flows in a balanced binary tree for $\sigma = 0$ (above) and $\sigma = 1000$ (below)

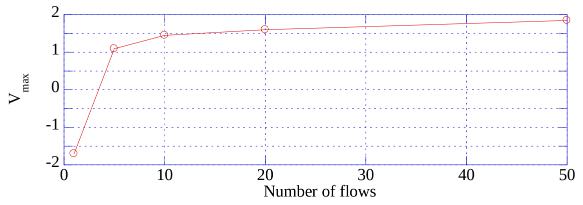


Fig. 10 - Constant overall load (homogeneous flows)

The same analysis was repeated for other topologies, i.e., unbalanced and ternary trees, with homogeneous traffic. In all cases qualitatively similar results were obtained, which are henceforth omitted for the sake of conciseness. More specifically, it always holds that per-flow queuing fares the worst, and that a decreasing line separates the regions of the (σ, ρ)-plane where per-exit-point aggregation fares better than per-path aggregation. Coefficients in (13), however, are topology-specific.

The above considerations are also true, at least up to some extent, if we relax the assumption of homogeneous flows. Fig. 12 reports the average $V_{\max}$ for 30 instances, on the balanced tree with heterogeneous random flows (confidence intervals are not visible): rates and bursts are generated uniformly between $[0.8 \cdot 300/K; 1.2 \cdot 300/K]$ and $[0.8 \cdot \sigma/K; 1.2 \cdot \sigma/K]$ respectively, with σ ranging from 1500 to 6000 and $K = 20$ being the number of fresh flows originating at each link. Although the lines represent averages, $V_{\max}^{per-p} \le V_{\max}^{per-f}$ holds for each instance of the problem. For instance, Fig. 12 suggests that the maximum aggregate burst schedulable in a per-flow framework is 4500, whereas in a per-path framework it is 6200, i.e. 38% larger. This corresponds to approximately 11 additional flows per node. More to the point, the qualitative behavior does not change if we allow for a larger spread for the bursts. We have expanded the above interval from $[0.8;1.2]$ to $[0.2;1.8]$, without experiencing any noticeable change in the outcome. Even considering non-uniform distributions for the flows, e.g. $[0.2;0.4]$ for one half of the flows and $[1.6;1.8]$ for the other, has no significant impact on $V_{\max}$. This seems to suggest that, as long as flows have the same deadline, aggregating heterogeneous flows improves the delay performance, and that the latter depends on the overall burst rather than on how it is distributed. On the other hand, the per-exit-point framework with the generated instances always fares worse. It has to be observed, however, that the point corresponding to the average values $(E[\sigma], E[\rho])$ in the (σ, ρ)-plane of Fig. 11 would be located in the region where per-path aggregation performs better. Although we do not show it here for the sake of conciseness, we can select flow parameters so as to obtain the opposite outcome.

So far, we have considered homogeneous deadlines. In fact, the behavior changes if flows with different deadlines are aggregated. In that case, in fact, per-flow scheduling comes back into play. Fig. 13 shows a case with 3 flows per node having the same σ, ρ, with $\rho = 300$ and σ ranging from 0 to 2000, but *different* deadlines (30, 60, and 100 respectively), on a balanced binary tree. For small bursts (i.e., below 600), per-path aggregation performs better, whereas per-flow is winning for larger bursts. This can be explained by considering that, depending on the burst size, either the *latency* or the *burst* delay may be predominant in (3). On one hand, as already noticed, aggregating flows always *reduces* their latency delay. On the other hand, tighter delay requirements are imposed on the aggregate, which instead increases the maximum violation. The first effect dominates for small bursts. Furthermore, note that in none of the above cases per-exit-point aggregation performs better. The same considerations again apply to different topologies, such as random and ternary trees.

From the above analysis, the following conclusive remarks can be obtained:

- aggregating flows on the same path is always beneficial as long as they have the same deadline. On the other hand, it matters little whether their traffic characteristics (σ, ρ) are similar or not. For instance, real-time traffic of different types (e.g., voice and video) can be aggregated, as long as the deadlines are the same.

- Aggregating flows progressively is beneficial only when flows have the same deadlines and limited bursts (e.g., voice traffic, which is known to be non-bursty). The limit beyond which it stops being beneficial decreases with the flows rate. For instance, high-rate, bursty flows (e.g., compressed video streams) should not be aggregated at each node if delay guarantees are a concern.

- The above considerations are fairly insensitive of the actual tree shape. Regular and irregular trees exhibit few differences.

It is also evident that there is no clear winner among the three aggregation frameworks, i.e. one that it is likely to warrant a higher schedulability in all the scenarios. A clearer picture can be obtained by putting computation overhead into the framework, which is what we do in the next section.

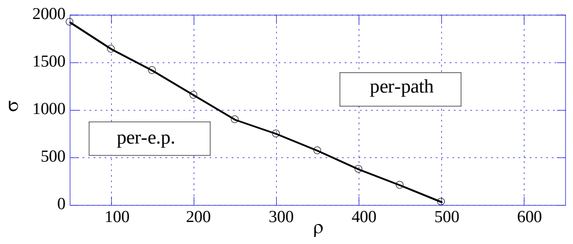


Fig. 11 – Regions of the (σ,ρ) plane where a given aggregation model leads to smaller $V_{\max}$ (homogeneous flows case)

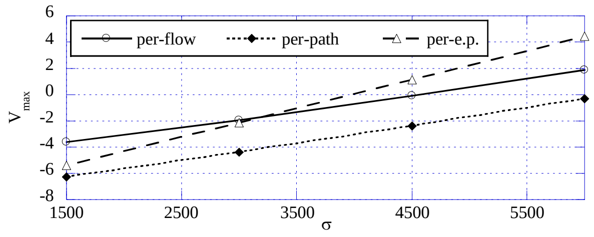


Fig. 12 - Heterogeneous randomly generated flows

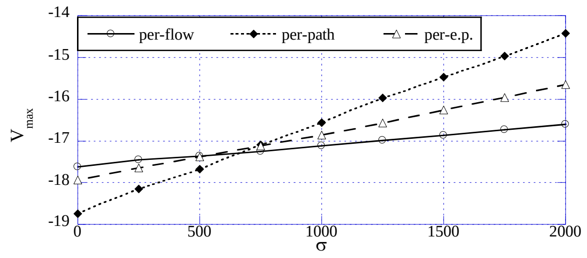


Fig. 13 - Flows with different deadlines in a balanced binary tree

V. Online admission control

The presence of integer (π_e, Δ_e) variables and, mostly, the structure of the conflict-free scheduling constraints $\overline{S}$ makes the MinMVP complex. Furthermore, as already stated, the problem is also non convex under per-exit-point aggregation. Under per-flow and per-path frameworks, the MinMVP can be solved optimally for WMNs of few tens of nodes, which is the expected scale for current and future WMNs. Note that, somewhat counterintuitively, using a per-flow or per-path framework makes almost no difference in the solving times, at least not until the number of flows per path grows very large. This is due to the fact that the above choice only influences continuous variables, whereas the number of integer and binary variables stays the same in both cases. Fig. 14, left, shows the distribution of the computation times for solving 100 instances of three different WMNs in a per-path framework, i.e. a balanced binary tree of 15 nodes, a ternary tree of 13 nodes, and a random tree of 12 nodes carrying uplink traffic, using CPLEX [27]. Computations are done on a PC equipped with an Intel Core2 Duo E6400 2.1 GHz, 2 GB RAM and a Linux kernel 2.6.18. Solving larger instances (20-30 nodes) requires instead minutes or hours on the same system. These are clearly affordable times when compared to the timescales of network (re)engineering, but not so when compared to the timescale of admission control decisions. When a per-exit-point framework is used and the problem is non convex, the maximum size that one can expect to solve using available techniques is at most 15-20 nodes, depending on both the topology and the number of flows. For larger networks, the computations may be altogether impossible due to memory constraints. Furthermore, computation times tend to be considerably higher for topologies of a similar number of nodes. Fig. 14, right, shows the distribution of the computation times using the BARON solver [30] on 100 instances of each of the above-mentioned WMN topologies. Note that we had to use a different solver than CPLEX because the latter only solves convex problems.

For this reason, we now tackle the problem of trading optimality for computation time through heuristic approaches. We first discuss heuristics for the convex case, and show that these can also be adapted to work in the per-exit-point case.

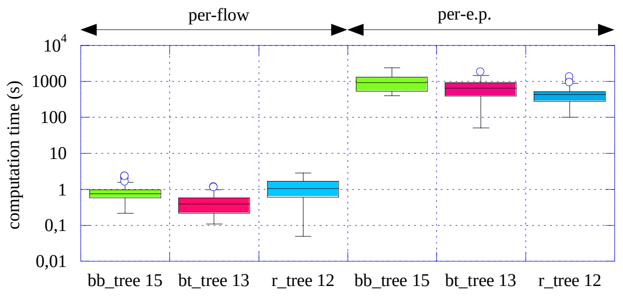


Fig. 14 - Box plot of the resolution times for different WMN topologies under per-flow and per-exit-point frameworks

A. *Heuristics for the per-flow and per-path cases*

Since the problem is convex, the greatest computational burden comes from the binary variables that define the transmission order, i.e. conflict orientations. Once the latter are set, near-optimal solutions to the remaining mixed integer-convex problem can be computed in few tens or hundreds of milliseconds for instances of tens of nodes. Hereafter, we refer to the problem of setting the conflict orientations as the *conflict* subproblem, and to the resulting, simplified MinMVP as the *re-*

duced MinMVP sub-problem. Our heuristic solution strategies rely on solving the two separately, in the above order.

Before delving deeper into each sub-problem, it is worth mentioning that the quality of the solution of the conflict sub-problem has a remarkable impact on the overall schedulability of a given traffic load: starting with a "bad" conflict orientation will thwart any attempt to compute a feasible schedule, even if the reduced MinMVP is solved optimally. Moreover, the conflict sub-problem is itself non trivial, so that trying to solve it optimally is out of question if speed is a concern.

In [20], a *blind* heuristic was presented, which can be used when no information about the underlying network is available. The latter relies on solving the conflict sub-problem using a genetic approach, where the fitness of each conflict orientation string is given by the optimum of the reduced MinMVP problem that it generates. Results shown in [20] prove that this allows computation time to be traded for optimality: allowing for a larger number of generations increases the likelihood of hitting a value of $V_{\max}$ close to the real optimum. The trade-off is configurable, so that a criterion based on the maximum response time for admission control can be used to stop the computations. In this paper a new *engineered* heuristic is proposed, where an estimate of the rates that the network is expected to support is used to infer an *offline* solution to the conflict sub-problem. It is often the case that a network administrator can estimate the *average rates* that its network links should support. In this case, some of the computations needed for an admission control test can be done offline. More specifically, the administrator can use that knowledge to solve the *conflict* problem, setting the transmission order variables, once and for all *offline*, and then solve the reduced MinMVP problem *online*, at the time of admission control. We will show that the reduced problem is accurate and only takes a few milliseconds, hence the online part is fast and effective.

Setting the conflict orientations transforms the unoriented conflict graph into a *dependency* graph. Define

$$f_e = \sum_{q:e\in P_q} \rho_q$$

the overall rate of flows traversing link e. Call:

$$\Delta_e^{LB} = \lceil N \cdot f_e / W_e \rceil \qquad (14)$$

the lower bound for the activation duration of link e, given an estimate of the average rate request f_e. If $\Delta_e < \Delta_e^{LB}$ at some link, then the problem is clearly infeasible because delays are unbounded according to (3). On the other hand, unless the following condition holds on every path P in the dependency graph:

$$\sum_{e\in P} \Delta_e^{LB} \le N, \qquad (15)$$

then the problem is infeasible. In fact, in this case there would be no way to partition the frame into activation durations while preserving bounded delays. Therefore, a good heuristic is one that allows you to set the conflict orientations so that i) (15) holds for all paths when it is possible to do so, and ii) the duration of the activation durations are maximized, so that delays are kept as small as possible. Given a rate estimate f_e, we formulate the conflict sub-problem as follows:

$$\begin{array}{lll} \max & \sum_{e\in E} f_e \cdot \Delta_e & \\ \text{s.t.:} & \Delta_e \ge N \cdot f_e / W_e & \forall e \in E \\ & \{\pi_e, \Delta_e\} \in \overline{S} & \\ & o_{ij} \in \{0,1\} & \forall (i,j) \in C_f \\ & \pi_e, \Delta_e \in \mathbb{N}_0^+ & \forall e \in E \end{array} \qquad (16)$$

In the above problem, the first constraint sets a lower bound on the activations as per (14), and the second one includes all the conflict inequalities. Note that, since the f_e play the role of weights in the objective, solving (16) gives preference to the links carrying the largest rates. Once an assignment for the o_{ij} variable is obtained as a solution of (16), the *online* part consists in solving the remaining reduced MinMVP.

A fast heuristic solution approach for the reduced MinMVP (shown in Fig. 15) relies on formulating a nonlinear convex problem starting from the MinMVP, relaxing the integrality constraints on variables π_e and Δ_e, and using the o_{ij} assignment from (16). Call S^* the set of conflict-free constraints that you get once the o_{ij} are assigned. The continuous relaxation is as follows:

$$\begin{array}{llll} \min & V_{\max} & & \\ \text{s.t.:} & D_q - \delta_q \le V_{\max} & \forall q \in Q & (i) \\ & R_{\min}^q \le W_e \cdot \Delta_e^q / N & \forall e \in P_q, \forall q \in Q & (ii) \\ & \Delta_e^q \ge N \cdot \rho_q / W_e & \forall e \in P_q, \forall q \in Q & (iii) \\ & \Delta_e \ge 1 + \sum_{q:e\in P_q} \Delta_e^q & \forall e \in E & (iv) \\ & \{\pi_e, \Delta_e\} \in S^* & & (v) \\ & \pi_e, \Delta_e \in \mathbb{R}_0^+ & \forall e \in E & (vi) \\ & R_{\min}^q \in \mathbb{R}_0^+ & \forall q \in Q & (vii) \\ & \Delta_e^q \in \mathbb{R}_0^+ & \forall e \in P_q, \forall q \in Q & (viii) \end{array}, \quad (17)$$

Model (17) can be easily interpreted by comparison with (4): constraints (*i-iii*) and (*vii-viii*) are in fact the same; (*vi*) is the continuous relaxation of the same constraint in (4); (*iv*) is homologous to the one in (4): however, as the non-integer solutions of (17), call them π_e^*, Δ_e^*, must then then *rounded* to their integer part, a "+1" is required to ensure that $\lfloor \Delta_e^* \rfloor \ge \sum_{q\in I_e} \Delta_e^q$, i.e., to prevent the rounding from reducing the minimum guaranteed rate to below the required one.. Constraint (*v*) summarizes conflict-free conditions having already solved the conflict orientation subproblem.

Variables π_e^*, Δ_e^* obtained in the solution to (17) are then *rounded* to their floor integer. Note that rounding preserves the conflict-free condition, since:

$$\pi_i^* + \Delta_i^* \le \pi_j^* \Rightarrow \lfloor \pi_i^* \rfloor + \lfloor \Delta_i^* \rfloor \le \lfloor \pi_j^* \rfloor \qquad (18)$$

Finally, the rounded solution is given as an input to the following *auxiliary problem*:

$$\begin{array}{lllr} \min & V_{\max} & & \\ \text{s.t.:} & D_q - \delta_q \le V_{\max} & \forall q \in Q & (i) \\ & R_{\min}^q \le W_e \cdot \Delta_e^q / N & \forall e \in P_q, \forall q \in Q & (ii) \\ & \lfloor \Delta_e^* \rfloor \ge \sum_{q \in I_e} \Delta_e^q & \forall e \in E & (iii) \\ & R_{\min}^q \in \mathbb{R}_0^+ & \forall q \in Q & (iv) \\ & \Delta_e^q \in \mathbb{R}_0^+ & \forall e \in P_q, \forall q \in Q & (v) \end{array} \qquad (19)$$

The purpose of solving (19) is to compute the *actual* $V_{\max}$: in fact, some Δ_e^q computed as a solution of (17) might still be increased (thus further reducing $V_{\max}$) before constraint (*iii*) in (19) becomes active. Constraints (*i*-ii) and (*iv*-*v*) are homologous to those of the previous problems. Note that integer scheduling variables π_e, Δ_e are not part of (19), since the schedule is not modified. Both (17) and (19) are convex, hence solvable in polynomial time using interior point methods. The rounding procedure runs in $\Theta(2 \cdot |E|)$ time. On the same system mentioned before, the heuristic solves 30-node instances within few tens of ms in a per-path framework, whereas optimally solving the reduced MinMVP problem takes seconds or tens thereof.

```
solve continuous relaxation (17)
   input: $o_{i,j}$
   output: $\pi_e^*, \Delta_e^*$

for each e in E {        //rounding
   $\pi_e$=floor($\pi_e^*$)
   $\Delta_e$=floor($\Delta_e^*$)
}

solve auxiliary problem (19)
   input: $\pi_e, \Delta_e$
   output: $\Delta_e^q, R_{\min}^q, V_{\max}$
```

Fig. 15 – Pseudocode of the solution scheme for the reduced MinMVP

In order to evaluate the accuracy of the heuristic, we compare i) the minimum of the reduced MinMVP and the one of the MinMVP, and ii) the number of times when the former fails to compute a feasible solution, which is instead found by the latter. We do so on two sample topologies.

- a 31-node balanced binary tree with uplink traffic only, i.e. with 30 flows, each one generated by one node.
- the WMN whose connectivity graph is shown in Fig. 16, which is loosely based on the one of the TFA project [31]. We deploy the 21 nodes in the same positions as in [31], and assume that each node is equipped with one omnidirectional antenna[2]. We set the transmission range of each node so that the WMN is fully connected. Nodes 0 and 17 are gateways, and eight *bidirectional* flows per gateway (i.e., 32 overall) are transmitted to/from the gateways.

In both topologies, all links have a capacity equal to 9600. We show that, if the rate estimate is accurate, the proposed approach is effective. We perform the *offline* conflict resolution assuming that each flow generates a given rate as an estimate ρ_{est}. Then, we solve the reduced MinMVP. In doing so, we use homogeneous bursts and deadlines for all flows, however using *different* rates than ρ_{est}. The employed rates are a random variable, with a mean value of ρ_{est}. For each topology, we solve 30 instances. The parameters are shown in Table 1.

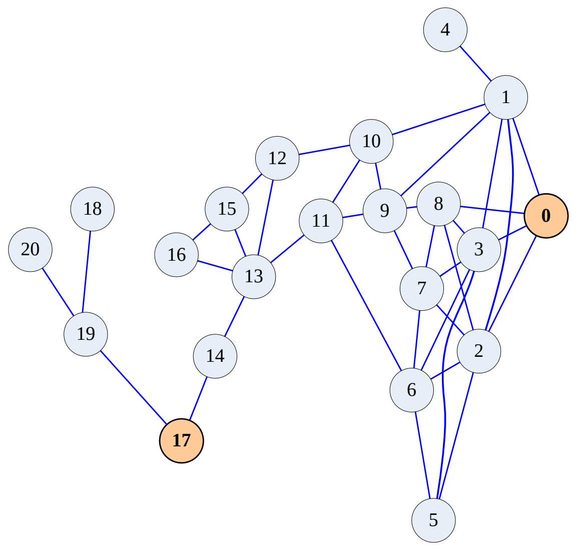


Fig. 16. The case-study WMN.

Table 1 – Parameters for the performance evaluation

Topology	Tree	TFA
ρ_{est}	300	200
σ	500	500
δ	40	40
W	9600	9600

Fig. 17 shows that the reduced MinMVP is always within few percentage points to the real optimum, regardless of the topology and of the rate distribution. As Fig. 18 shows, the percentage of instances declared unfeasible grows with the standard deviations, which is expectable. However, in the balanced tree topology, which fares the worst of the two, it is be-

[2] In [31], some nodes are also equipped with directional antennas to gateway nodes. We do not include these links, which are less interesting from a link scheduling perspective.

low 10% as long as the standard deviation is below 16% of the average.

As far as computation times are concerned, the online part completes in tens to hundreds of milliseconds in both topologies, as shown in Fig. 19. The optimum solution instead requires tens to thousands of seconds. Interestingly enough, the *offline* part of the heuristic (not shown in the graph) is also reasonably fast, although efficiency is a minor concern in this case. For the two topologies, we obtained an upper bound of 200ms for the offline part.

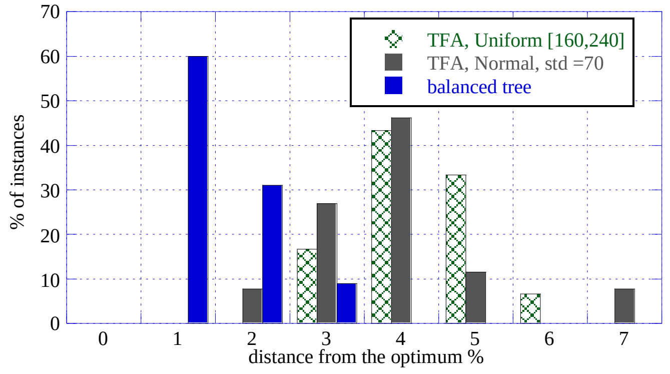


Fig. 17 - Accuracy of the reduced MinMVP problem

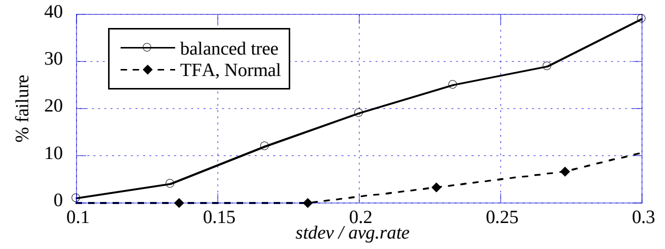


Fig. 18 – Percentage of solved instances against the standard deviation of the rate distribution

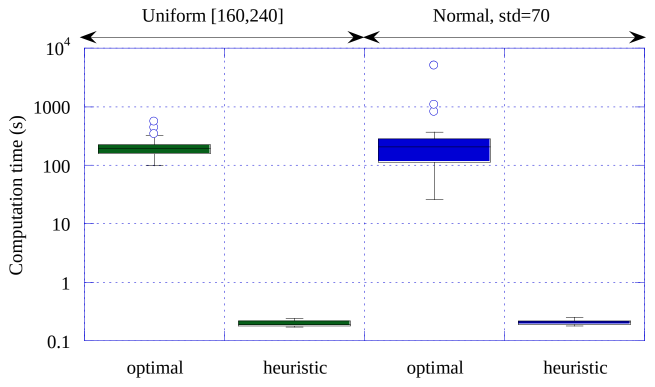


Fig. 19 – Box plot of the solving times of the MinMVP and the Reduced MinMVP, TFA topology

B. *The per-exit-point case*

Under per-exit-point aggregation, separating the conflict and reduced MinMVP sub-problems does not pay off, as the latter is non convex, and therefore still hard to solve. While general-purpose meta-heuristics for solving non-convex problems have been around for a while (e.g., multi-start methods), they all share some common features: they are hardly predictable, very dependent on parameter tuning, and – being general purpose – they seldom exploit possible underlying structures of the problem to be solved. Instead of trying to adapt the above meta-heuristics to finding suboptimal solutions in the per-exit-point case, we take a different approach: we show that solutions of the per-path problem (whether optimal or computed through the heuristic approach) can be exploited to find good suboptimal solutions in the per-exit-point case. Our claim is that given a solution of the per-path case in the following form:

$$\begin{cases} \overline{\pi_e} & e \in E \\ \overline{\Delta_e^q} & e \in E, q \in I(e) \end{cases}$$

the following solution, used in a per-exit-point framework:

$$\begin{cases} \pi_e = \overline{\pi_e} & e \in E \\ \Delta_e = \sum_{q \in I(e)} \overline{\Delta_e^q} & e \in E \end{cases}, \tag{20}$$

yields a value of $V_{\max}$ which, though obviously suboptimal, is often reasonably close to the optimal value of the per-exit-point case. Therefore, we can compute a good suboptimal solution for the per-exit-point case without having to solve any non-convex problem. The above claim is supported by the results shown in Fig. 20, where we plot the relative distance to the optimum of the heuristic obtained by using (20). The box plot is obtained by running 30 random instances of a balanced binary tree network, with rates uniformly selected in $[240,360]$ and burst selected in $[0.8 \cdot K, 1.2 \cdot K]$, $K = 1500, 3000, 4500, 6000$. As the figure shows, the heuristic solution is seldom above 15% of the optimal value.

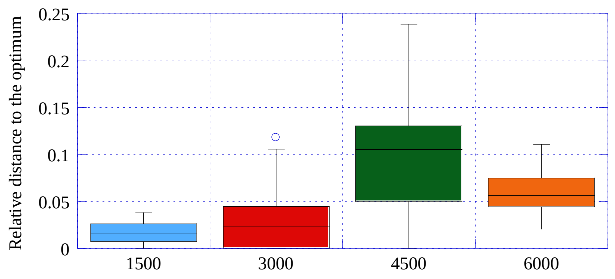


Fig. 20 - Box plot of the relative deviation between the optimal solution of the per-exit-point case and the one computed using the engineered heuristic and (20).

Note that, while computing an optimal link schedule in the per-exit-point framework is a tough problem, a *delay feasibility test* for a given link schedule, which is required to check whether the solution is admissible in the per-exit-point case, only takes $O(|E|)$ time [32]. Such time overhead is negligible with respect to the one required for computing heuristic solutions in the per-path case. Therefore, by using the above procedure, heuristics of the *same* efficiency as the per-flow/per-path cases can be applied to the per node case too.

As a final note, we observe that formulating the problem as a min-max problem pushes a solver to allocate the largest possible number of slots, so as to minimize the max violation. As discussed in [20], this intrinsically produces robust schedules, where relatively large variations in some flows' parameters can be tolerated without violating the deadlines, even when $V_{\max}$ is close to zero. This can be exploited in order to avoid computing a link schedule altogether in response to changes in the traffic parameters. We showed in [20] a method to assess in polynomial time whether a new computation is needed or not.

VI. Related work

In this section we review the available related work on link scheduling in WMNs. As already stated, no work that we are aware of (save our previous work on the same topic, [18]-[21]) considered schedulability in WMNs with: i) leaky-bucket-constrained traffic, which takes into account variable bit rates in a short-term, and ii) arbitrary end-to-end delay constraints. Most of the relevant literature on link scheduling falls into either of the following categories:

1. *rate-oriented* algorithms, that either provide flows with a *minimum guaranteed rate* (e.g. [3]-[7]), or optimize the total throughput (e.g. [8]-[11]). Guaranteeing a minimum rate no smaller than the flow's rate – by (3) or (6) – is a necessary condition for end-to-end delays to be *finite*, but does not automatically make them smaller than a pre-specified bound. Note that our schemes also fall in this category if delay constraints are *loose*. In fact, if the delay requirements δ_q of each flow are such that constraints (*i*) are never active in (4) or (11), then the solution to minMVP will still guarantee that each flow gets at least the minimum required rate ρ_q, hence our scheme can be used in this context as well. If, instead, delay requirements are *tight* (hence the related constraints active), our scheme will overallocate rates with respect to ρ_q, to satisfy the delay constraint.
2. *TDMA delay-oriented* algorithms, that either minimize (e.g. [14]-[15]) or try to guarantee a *maximum TDMA delay* (e.g. [12]-[13]). The latter is the sum of TDMA waiting times at every hop, i.e. the time it takes for a packet to travel from the source to the destination, assuming that it is never queued behind other packets. As queuing is a component (and often the dominant one) of the end-to-end delay, especially with VBR traffic, there is no guarantee that such algorithms can actually find a delay-feasible schedule if there exists one.

Within the second category, [12] gives *a priori* guarantees, whereas [13] uses admission control to check whether the required TDMA delay is feasible. [14] considers both CBR (voice) and VBR (video) flows, however assuming that VBR sources can be described as stationary, ergodic and independent processes with known statistics, so as to characterize them as equivalent CBR sources. In this work, we deliberately omit this kind of assumptions, sticking instead to more practical σ, ρ characterizations, which can be conveyed to the network using standard signaling protocols (e.g., RSVP, [22]). In [15], a WMN is modeled as a stop-and-go system. A min-max problem on the round-trip TDMA delay introduced by the scheduling in a sink-tree network is formulated and optimally solved. However, minimizing the TDMA delay does not imply computing a delay-feasible schedule when there is one. We show this by comparing our schedules against those derived from [15]. In the latter, link activations are computed based on the *rate* (which only guarantees *finite* delays), and activations are serialized so as to minimize the maximum TDMA delay. Consider a 15-node binary tree, with homogeneous traffic and 20 flows originating at each node. Fix $\delta = 20$, $\rho = 300$, and let the burst of the flows vary as $0 \le \sigma \le 4500$. We plot the $V_{\max}$ value obtained by: i) optimally solving the MinMVP in the per-flow, per-path and per-exit-point frameworks, and ii) using the optimal solutions given by [15] in the same settings. As Fig. 21 shows, [15] yields positive $V_{\max}$ values, except at the far left (i.e., when the burst is negligible), for traffic which would otherwise be schedulable using the MinMVP approach. This is because [15] optimizes *only* conflict orientations (o_{ij}) and activation instants (π_e), neglecting the activation *durations* (Δ_e, Δ_e^q). Optimizing on the latter too is instead very important to achieve delay-feasible schedules.

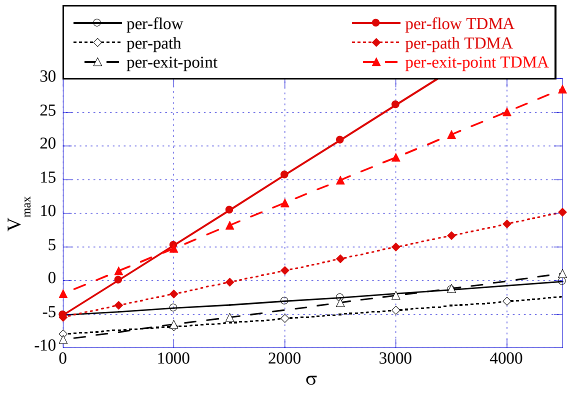


Fig. 21 – Comparison between optimizing on $V_{\max}$ and minimizing the maximum TDMA delay

Some works not falling into either of the above categories are also relevant, as they provide frameworks for computing delay bounds *a posteriori*, after link scheduling has been planned. In [16] authors define the *odd/even link activation* and routing framework, and employ internal scheduling policies at each link so that the end-to-end delay bound along a path is roughly double than the one obtained in a wired network of the same topology. Authors of [17] show that using throughput-optimal link scheduling and Coordinated-EDF to schedule packets within each link, rate-proportional delay bounds with small additive constants are achieved. Our goal is instead to have pre-specified, *arbitrary* delay bounds respected through link scheduling. In [33], authors solve the delay-constrained link scheduling problem in Wireless Sensor Networks, using effective capacity as a model. Simulation results reported therein confirm that our scheme performs better than [15] in that setting as well.

A recent work of ours, [18], marks a first step for the analysis reported in this paper. In that work, sink-tree WMNs, with per-exit-point aggregation are analyzed. However, only *feasible* solutions (with no optimality associated) were found through an iterative heuristic. The latter was not guaranteed to find a feasible solution even if there exists one. With respect to [18], we formulate the link scheduling problem as a mathematical programming problem (thanks to Theorem 5) that can be optimally solved. Furthermore, all three aggregation frameworks (per-flow, per-path and per-exit-point) are considered and compared. Finally, [21] is a first attempt to bring routing back into the framework, i.e. to address the problem of *jointly* solving the routing and link scheduling problem *optimally*, taking into account end-to-end delay guarantees. However, aggregation policies are not taken into account.

The approach pursued in this paper lends itself to some generalizations. As shown in [20], when analyzing WMNs with *per-flow* and *per-path* queuing, a sink-tree routing is not a requirement. The same framework, including the problem formulation, optimal and heuristic solution strategies, can be employed with *any* topology, provided that i) the conflict graph, and ii) routes are given. On the other hand, in a per-exit-point framework no closed-form delay bound is available except for a sink-tree topology [34]-[35].

A related stream of literature is that on *channel assignment* in Multi-Radio-Multi-Channel (MRMC) WMNs (see, e.g., [36]-[41]). For these, the standard assumption is that each mesh router has K radios that can be tuned on any of C orthogonal channels, with $C > K$ usually. Thus, channel assignment both determines connectivity and controls interference. Few works (e.g., [40]) advocate scheduling links and assigning channels jointly. However, link scheduling could also be used on a MRMC WMN to enforce mutual exclusion between residual interfering links, once collision domains have been separated through channel assignment. It can be easily observed that our schemes work seamlessly in these settings as well, and probably faster at that, since multi-channel transmission reduces the number of interfering links. On the other hand, some works (see, e.g., some of those surveyed in [41]) advocate assigning channels *dynamically* (i.e., at timescales of one or few packet transmissions). Such schemes rely on collision avoidance techniques, such as RTS/CTS handshakes, and their aim is normally to improve the network throughput or reduce average delays. None of them is concerned (or compatible) with end-to-end deterministic delay guarantees, which is instead what our schemes are about.

VII. CONCLUSIONS AND FUTURE WORK

Link scheduling for real-time traffic in Wireless Mesh Networks requires that *end-to-end delay bounds* be guaranteed *a priori*, and the existing link scheduling algorithms do not take the latter into account. We showed that, given link rates, a conflict graph and flow routes, the feasibility of a link schedule does depend on the aggregation framework, and we derived guidelines to choose the appropriate aggregation framework given a network scenario. The general problem is formulated as an integer-non-linear optimization problem that can be solved optimally for WMNs of small to medium size, depending on the aggregation framework at the nodes. For WMN of larger sizes and/or in smaller times, we proposed a heuristic solution approach that computes good suboptimal schedules; this approach can be used as an online admission control scheme for real-time traffic.

VIII. ACKNOWLEDGEMENTS

This work was carried out while Alessandro Lori was with the Dipartimento di Ingegneria dell'Informazione of the University of Pisa, Italy.

REFERENCES

[1] Akyildiz, I. F., Wang, X. and Wang, W. (2005) Wireless mesh networks: a survey. Computer Networks, 47:(4), 445-487.

[2] Nelson, R. and Kleinrock, L. (1985) Spatial TDMA: A collision-free multihop channel access protocol. IEEE Trans. on Communications, 33(9), 934-944.

[3] Bhatia R. and Kodialam M. (2004) On Power Efficient Communication over Multi-hop Wireless Networks: Joint Routing, Scheduling and Power Control. Proceedings of INFOCOM 2004, Hong Kong, 7-11 March, pp. 1457-1466. IEEE, Piscataway, NJ (USA)

[4] Kazemitabar J., Tabatabaee V. and Jafarkhani H. (2008) Global Optimal Routing, Scheduling and Power Control for Multi-hop Wireless Networks with Interference. Proceedings of GLOBECOM 2008, New Orleans (US), Nov. 30-Dec. 4, pp. 1-5. IEEE, Piscataway, NJ (USA).

[5] Cruz, R. and Sanathanam, A. (2003) Optimal Routing, Link Scheduling and Power Control in Multi-hop Wireless Networks. Proceedings of INFOCOM 2003, San Francisco (US), March 30-Apr. 3, pp. 702-711. IEEE, Piscataway, NJ (USA).

[6] Kodialam, M.S. and Nandagopal, T. (2003) Characterizing achievable rates in multi-hop wireless networks: the joint routing and scheduling problem. Proceedings of MOBICOM 2003, San Diego, (US), 14-19

September, pp. 42-54. ACM, New York City (USA).

[7] El-Najjar, J., Assi, C. and Jaumard, B. (2009) Joint Routing and Scheduling in WiMAX-based mesh networks: A Column Generation Approach. Proceedings of WoWMoM 2009, Kos, Greece, 15-19 June. IEEE, Piscataway, NJ (USA).

[8] Wang, Y., Wang, W., Li, X.-Y. and Song, W.-Z. (2008) Interference-Aware Joint Routing and TDMA Link Scheduling for Static Wireless Networks. IEEE Trans. on Parallel and Distributed Systems, 19(12), 1709-1725.

[9] Crichigno, J., Wu, M. Y., Khoury, J. and Shu, W. (2009) A Joint Routing and Scheduling Scheme for Wireless Networks with Multi-packet Reception and Directional Antennas. Proceedings of WoWMoM 2009, Kos, Greece, 15-19 June. IEEE, Piscataway, NJ (USA).

[10] Jain, K., Padhye, J., Padmanabhan, V. and Qiu, L. (2003) Impact of interference on multi-hop wireless network performance. Proceedings of MOBICOM 2003, San Diego, (US), 14-19 September, pp. 66-80. ACM, New York City (USA).

[11] Goussevskaia, O., Wattenhofer, R., Halldorsson, M. M. and Welzl, E. (2009) Capacity of Arbitrary Wireless Networks. Proceedings of INFOCOM 2009, Rio de Janeiro, Brazil, 15-25 April, pp. 1872-1880. IEEE, Piscataway, NJ (USA).

[12] Leoncini, M., Santi, P. and Valente, P. (2008) An STDMA-Based Framework for QoS Provisioning in Wireless Mesh Networks. Proceedings of. MASS 2008, Atlanta (USA), Sept. 29-Oct. 2, pp. 223-232. IEEE, Piscataway, NJ (USA).

[13] Zou, J. and Zhao, D., (2009) Connection-Based Scheduling for Supporting Real-Time Traffic in Wireless Mesh Networks. IEEE Trans. on Wireless Communications, 8(3), 1182-1187.

[14] Shetiya, H. and Sharma, V. (2005) Algorithms for Routing and Centralized Scheduling to Provide QoS in IEEE 802.16 Mesh Networks. Proceedings of WMuNeP'05, Montreal, CA, Oct. 13, pp. 140-149. ACM, New York City (USA).

[15] Djukic, P. and Valaee, S. (2009) TDMA delay aware link scheduling for multi-hop wireless networks. IEEE/ACM Trans. on Networking, 17(3), 870-883

[16] Narlikar, G., Wilfong, G. and Zhang, L. (2010) Designing Multihop Wireless Backhaul Networks with Delay Guarantees. Kluwer Wireless Networks, 16:(1), 237-254

[17] Jayachandran, P. and Andrews, M. (2010) Minimizing end-to-end Delay in Wireless Networks Using a Coordinated EDF Schedule. Proceedings of INFOCOM 2010, San Diego (US), 15-19 March, pp. 1-9. IEEE, Piscataway, NJ (USA)

[18] Cappanera, P., Lenzini, L., Lori, A., Stea, G. and Vaglini, G. (2009) Link Scheduling with End-to-end Delay Constraints in Wireless Mesh Networks. Proceedings of WoWMoM 2009, Kos, Greece, 15-19 June, pp. 1-9. IEEE, Piscataway, NJ (USA).

[19] Cappanera, P., Lenzini, L., Lori, A., Stea, G. and Vaglini, G. (2011) Efficient Link Scheduling for Online Admission Control of Real-time Traffic in Wireless Mesh Networks. Elsevier Computer Communications, 34:(8), 922-934.

[20] Cappanera, P., Lenzini, L., Lori, A., Stea, G. and Vaglini, G. (2010) Optimal Link Scheduling for Real-time Traffic in Wireless Mesh Networks in both Per-flow and Per-path Frameworks. Proceedings of. WoWMoM 2010, Montreal, Canada, 14-17 June, pp. 1-9. IEEE, Piscataway, NJ (USA).

[21] Cappanera, P., Lenzini, L., Lori, A., Stea, G. and Vaglini, G. (2012) Optimal joint routing and link scheduling for real-time traffic in TDMA Wireless Mesh Networks. Elsevier Computer Networks, 57 (11), 2301-2312.

[22] RFC 1633, (1994). Integrated Services in the Internet Architecture: an Overview. The Internet Society.

[23] RFC 2475 (1998). An Architecture for Differentiated Services. The Internet Society.

[24] Iyer, A., Rosenberg, C. and Karnik., A. (2009) What is the right model for wireless channel interference? IEEE Transactions on Wireless Communications, 8:(5), 2662-2671.

[25] Le Boudec, J.-Y. and Thiran, P. (2001) Network Calculus. LNCS 2050, Springer Verlag, New York, NY.

[26] P. Bonami and J. Lee, BONMIN Users' Manual: see http://projects.coin-or.org/Bonmin

[27] ILOG CPLEX software, http://www.ilog.com

[28] Lenzini, L., Martorini, L., Mingozzi, E. and Stea, G. (2006) Tight end-to-end per-flow delay bounds in FIFO multiplexing sink-tree networks. Elsevier Performance Evaluation, 63(9-10), 956-987.

[29] Lenzini, L., Mingozzi, E. and Stea, G. (2008) A Methodology for Computing End-to-end Delay Bounds in FIFO-multiplexing Tandems. Elsevier Performance Evaluation, 65, 922-943.

[30] BARON solver (available freely via the NEOS server): http://archimedes.cheme.cmu.edu/baron/baron.html.

[31] TFA website, http://tfa.rice.edu.

[32] Lenzini, L., Martorini, L., Mingozzi, E. and Stea, G. (2006). A novel approach to scalable CAC for real-time traffic in sink-tree networks with aggregate scheduling. Proceedings of VALUETOOLS 06, Pisa, Italy, 10-13 October. ICST, Bruxelles.

[33] Wang, Q., Wu, D. O. and Fan, P. (2010) Delay-Constrained Optimal Link Scheduling in Wireless Sensor Networks. IEEE Trans. on Vehicular Technology, 59:(9), 4564-4577.

[34] Bisti, L., Lenzini, L., Mingozzi, E. and Stea, G. (2012) Numerical analysis of worst-case end-to-end delay bounds in FIFO tandem networks. Springer Real Time Systems Journal, 48:(5), 527-569.

[35] Bouillard, A. and Stea, G. (2012). Exact worst-case delay for FIFO-multiplexing tandems. Proceedings of VALUETOOLS 2012, Cargese (FR), 5-7 October, pp. 158-167. ICST, Bruxelles.

[36] Draves, R. Padhye, J. and Zill, B. (2004). Routing in multi-radio, multi-hop wireless mesh networks. Proceedings of Mobicom-04, Sept. 26-Oct. 1, Philadelphia, PA, US, pp. 114-128. ACM, New York City (USA).

[37] Ramachandran, K.N., Belding, E. M., Almeroth, K. C. and Buddhikot, M. M. (2006). Interference-aware channel assignment in multi-radio wireless mesh networks. Proceedings of INFOCOM 2006, Barcelona, Spain, 23-29 April, pp. 1-12. IEEE, Piscataway, NJ (USA).

[38] Das, A. K., Alazemi, H. M. K., Vijayakumar, R. and Roy, S. (2005). Optimization models for fixed channel assignment in wireless mesh networks with multiple radios. Proceedings of SECON 2005, Santa Clara, California, USA, 26-29 September, pp. 463-474. IEEE, Piscataway, NJ (USA).

[39] Subramanian, P.A., Gupta, H. and Das, S. R. (2007) Minimum interference channel assignment in multi-radio wireless mesh networks. Proceedings of SECON 2007, San Diego, California, USA, 18-21 June, pp. 481-490. IEEE, Piscataway, NJ (USA).

[40] Yu, H., Mohapatra, P. and Liu, X. (2008) Channel assignment and link scheduling in multi-radio multi-channel wireless mesh networks. Mobile Networking and Applications, 13, 169-185.

[41] Crichigno, J, Wu, M.Y. and Shu, W. (2008) Protocols and architectures for channel assignment in wireless mesh networks, Ad Hoc Networks 6 (7), 1051-1077.

IX. Appendix

A. Proof of Theorem 5

Consider a generic sequence of nodes S in Σ_x, and compute the following expression:

$$Q_S = R_{n_S(|S|)} \cdot \prod_{h=1}^{|S|-1} \frac{R_{n_S(h)}}{R_{n_S(h)} + \left(r_{n_S(h+1)} - r_{n_S(h)} \right)}$$

We show that i) adding a bottleneck node to -, and ii) removing a non-bottleneck node from S leads to a smaller value for Q_S. This proves the thesis, as

$$CR_x = \min_{S \in \Sigma_x} \{Q_S\}$$

In order to simplify the notation, we drop the path subscript in the proof, without this generating ambiguity.

i) Build the sequence $S' = S \cup \{b\}$, where $b \in B \setminus S$. Now, either b is the last node in S', or it isn't. In the first case, call l the last node in S. After some straightforward algebraic manipulation, we obtain:

$$Q_{S'} = Q_S \cdot \frac{R_b}{R_l + (r_b - r_l)},$$

However, since $b \in B$, the last term is smaller than or equal to 1 (by the very definition of bottleneck), hence $Q_{S'} \le Q_S$

If, instead, b is not the last node in S', then there exist two nodes in S, call them l, m, such that $l < b < m$. Therefore:

$$Q_S = R_{n(S,1)} \cdot \prod_{h=1}^{|S|-1} \frac{R_{n_S(h+1)}}{R_{n_S(h)} + \left(r_{n_S(h+1)} - r_{n_S(h)}\right)}$$
$$= \Delta \cdot \frac{R_m}{R_l + (r_b - r_l) + (r_m - r_b)}$$

for some positive Δ, and

$$Q_{S'} = \Delta \cdot \frac{R_b}{R_l + (r_b - r_l)} \cdot \frac{R_m}{R_b + (r_m - r_b)}$$

Therefore, in order to prove that $Q_{S'} \le Q_S$, we need to prove that:

$$\frac{R_b}{R_l + (r_b - r_l)} \cdot \frac{1}{R_b + (r_m - r_b)} \le \frac{1}{R_l + (r_b - r_l) + (r_m - r_b)}.$$

After simple algebraic manipulations, the above expression boils down to:

$$\frac{R_b}{R_b + (r_m - r_b)} \le \frac{R_l + (r_b - r_l)}{R_l + (r_b - r_l) + (r_m - r_b)}$$

which is equivalent to $R_b \le R_l + (r_b - r_l)$. The latter is true since b is a bottleneck, hence $Q_{S'} \le Q_S$ in any case.

ii) Consider now a sequence S that includes *all* bottleneck nodes, i.e. $B \subseteq S$, and at least a node $f \in S \setminus B$ (i.e., a non bottleneck). Call $S' = S \setminus \{f\}$. We show that $Q_{S'} \le Q_S$. Assume first that the node preceding f, call it b, *is* a bottleneck (we will show later on that this comes with no loss of generality). Note that, since $1 \in S \cap B$, one such node exists for sure. Now, either f is the last node in S, or it isn't. In the first case, call we have:

$$Q_S = Q_{S'} \cdot \frac{R_f}{R_b + (r_f - r_b)},$$

and the last term is larger than 1 since f follows a bottleneck, hence $Q_{S'} \le Q_S$.

In the second case, call m the next node in S following f, i.e. $b < f < m$. Then:

$$Q_S = \Delta \cdot \frac{R_f}{R_b + (r_f - r_b)} \cdot \frac{R_m}{R_f + (r_m - r_f)}, \text{ and}$$

$$Q_{S'} = \Delta \cdot \frac{R_m}{R_b + (r_f - r_b) + (r_m - r_f)},$$

for some positive Δ. Again, the thesis $Q_{S'} \le Q_S$ can be easily rewritten as:

$$\frac{R_b + (r_f - r_b)}{R_b + (r_f - r_b) + (r_m - r_f)} \le \frac{R_f}{R_f + (r_m - r_f)},$$

which holds if and only if $R_b + (r_f - r_b) \le R_f$. However, since b is the last bottleneck before f and f is not a bottleneck, then $R_b + (r_f - r_b) < R_s$, and therefore $Q_{S'} \le Q_S$. This proves that you can remove every *first* node following a bottleneck and obtain $Q_{S'} \le Q_S$. However, by iterating the same procedure, you can progressively remove every *second, third,…, n^th^* node following a bottleneck. Therefore, you can ultimately remove *all* non-bottleneck nodes and obtain a smaller expression.

Wrapping up, if you take a generic sequence S and i) add *all* bottleneck nodes, and ii) remove *all* non-bottleneck nodes, you obtain a sequence B such that $Q_B = CR \le Q_S$, which is the thesis.

Q.E.D.

B. *Table of the notations used in the paper*

Symbol	**Meaning**
	General
T_s	Duration of the transmission slot
N	Number of slots in a frame
V	Set of nodes in the WMN (vertices of the graph)
E	Set of links in the WMN (edges of the graph)
$\mathcal{I}(e)$	Set of conflicting edges for link e
W_e	Transmission rate of link e
π_e	Offset of link e in the link schedule
Δ_e	Activation duration of link e in the link schedule
o_{ij}	Binary variable, equal to 1 if link i transmits after j and 0 otherwise
$\overline{S}$	Feasible region for conflict-free constraints
δ_q	End-to-end deadline for flow q
σ_q	Burst of flow q
ρ_q	Rate of flow q
P_q	Path of flow q
D_q	Delay bound for flow q along its path
$V_{\max}$	Maximum delay violation
	Per-flow and per-path queuing
Δ_e^q	Activation duration for flow q on link e in the link schedule
R_e^q	Guaranteed rate for flow q on link e
θ_e^q	Latency for flow q on link e
$R_{\min}^q$	Minimum guaranteed rate for flow q along path P_q
	Per-exit-point queuing
R_e	Guaranteed rate on link e
θ_e	Latency on link e
s_x	Output burst at link x
$M_{i,j}$	First common node between paths P_i and P_j
r_x^*	Residual rate at link x
CR_x	Clearing rate at link x