A Simulation-Driven Approach To Scheduling In Duty-Cycled Networks

by

Van Hai Ho

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science
University of Alberta

ABSTRACT

Transceiver duty-cycling (DC) is a popular technique to conserve energy in a wireless sensor network (WSN). In this thesis, our overall objective is to study the performance of a DC WSN. Namely, we consider the performance of a DC WSN from the point of throughput, as well as energy consumption, under deterministic DC behavior. Our elementary notion of traffic is that of a traffic flow, routed via multiple hops from a source to a destination node. The traffic flows are, by default, assumed to be greedy, i.e., capable of making use of as much rate as they are given. We use max-min fairness as our criterion for allocating the available capacity across the multiple flows. Towards this end, we explore the following research questions. A first question is that of relating duty-cycling and achievable throughput through the construction of deterministic transmission schedules, i.e., by ensuring coordinated medium access and hence avoiding collisions and avoiding idle listening. A second question is that of throughput improvement in DC WSN by means of employing network coding (NC), again via the construction of a deterministic transmission schedule.

The key component of our methodology is a modelling step whereby the time-varying topologies of DC WSNs are captured by the notion of repeating "stages", i.e., time durations during which the topology is constant. The periodically repeating stages allow us to express optimization objectives for the throughput of the traffic flows. Whereas the optimization enables the determination of the per-flow rates, a novel simulation-based approach is introduced enabling the construction of periodic TDMA transmission schedules achieving, or at least closely approximating, those per-flow rates. The schedule construction process is based on the conjecture that the periodic behavior of a DC WSN is bound to result in a periodic steady state behavior for the network. Consequently we use simulation to extricate, and use, the periodic scheduling induced by the periodic steady state of the network.

Through numerous simulation studies and in comparison with the per-flow rates produced by the optimization step, we demonstrate that the technique is accurate and applicable to arbitrary topology DC WSNs. We also demonstrate how this approach can be used in combination with NC as well. While the particular NC scheme is one of numerous possible, it is sufficient to demonstrate the technique's advantages. The thesis concludes by outlining how the basic idea can be extended to DC WSNs with different DC periods, non-greedy traffic demands, and variations of NC schemes.

ACKNOWLEDGEMENT

First of all, I would like to express my deep gratitude to my supervisor Dr. Ioanis Nikolaidis for consistently supervising this study. Dr. Nikolaidis suggested this topic, granted a generous amount of time and intellectual effort in problem-solving discussions, and offered emotional help and encouragement to me.

Next, I would like to thank the examing committee, Dr. Paul Lu, Dr. Janelle Harms, Dr. Hai Jiang, Dr. Ramiro Liscano, who greatly helped me improve the thesis. Also, I would like to thank Dr. Martha Steenstrup and Dr. Ehab Elmallah from Communication Networks group, Department of Computing Science, for their valuable suggestions to my NetGroup talks.

Then, I would like to thank the technical and administrative staffs in Department of Computing Science at University of Alberta. Special thanks are for Sharon Bell for her timely support throughout the course of this research. I am also deeply grateful to my parents, my wife (Giac Nguyen) and my daughter (An Ho) for their patience, understanding, love and support during my study.

Finally, I would like to thank those who provided me with financial support. This work has been supported in part by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC). My work has also been supported by a scholarship from the Government of Vietnam.

Table of Contents

List of Figures

List of Tables

CHAPTER 1
INTRODUCTION

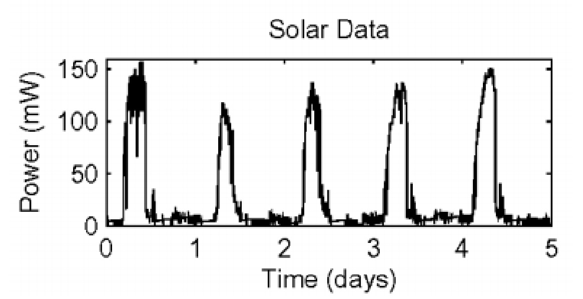
1.1 SCOPE AND MOTIVATION

Energy efficiency is one of the most important requirements in wireless sensor networks (WSNs), in particular, if the nodes are assumed to be powered by batteries that cannot be easily replaced or recharged. Therefore, the activities of the nodes should be limited as much as possible to prolong their lifetime to the maximum allowable ability of the batteries. Among these activities, the wireless transceiver operation costs significant energy to a node [1]. Some degree of activity of the transceiver is unavoidable because it is needed to transmit and receive packets intended for the node. The other two elements, the listening/sensing of the channel and the reception of packets not intended for the node are, at least in principle, avoidable. A broad technique employed to save energy is to avoid idle listening/sensing of the medium by duty-cycling (turning OFF) a node's transceivers. We use the terms “ON” “awake” and “active” (conversely, “OFF” “asleep” and “inactive”) to denote the two states of duty-cycling.

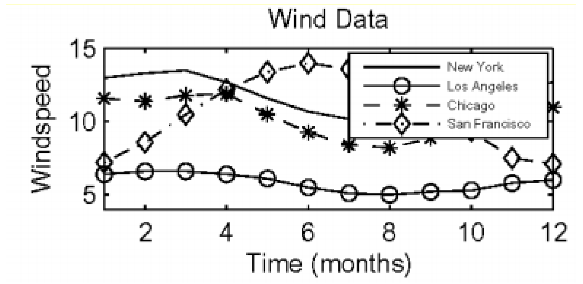
In this thesis, we assume that the data traffic that needs to be delivered is generated in a random fashion, and it has long idle durations between transmissions [2], and between generations as sensor readings. Hence, it is quite reasonable for sensor nodes to work in short-DC mode to achieve long autonomy if they are powered by non-renewable energy reserves, e.g. batteries [3]. We also consider WSN deployments with multiple origin-detination pairs of traffic flows, e.g., multiple applications served in the same WSN, each of them possibly running on certain of the nodes [4]. The need for multiple origin-destination pairs of traffic flows recently arises with the emergence of wireless sensor/actuator networks (WSANs) [57, 58, 59]. In a WSAN, sensor nodes and actuator nodes are interconnected via wireless links in a distributed fashion. Each sensor node collects information from the physical world and transmits it to actuator nodes over multi-hop communications. Based on the received information, actuator nodes can change behaviors of the physical world. In a fire scenario, for example, different actuator nodes are likely to need the information of detecting fire from the same sensor node with temperature readings. One actuator may transmit an emergency signal to the fire department while another may control a water sprinkler to extinguish the fire. The options for delivering the traffic are numerous, and, at a low layer, are described by Medium Access Control (MAC) protocols. In this context, one family of MAC protocols, determine the exact instants when a node has to be awake to receive (or transmit) a packet. This family includes Time Division Multiple Access (TDMA) schedules. Besides the ability to schedule in a fair manner the multiple traffic flows, TDMA is also excellent in terms of energy consumption due to the fact that a node can disable the transceiver when no traffic needs to be sent or received. Nevertheless TDMA schedules are considered as inflexible because they do not adapt to random fluctuations of the traffic load. At the other extreme, contention-based carrier-sensing MAC protocols, like ALOHA, attempt transmissions on an as-needed basis which is quite fit for varying traffic demands but forces the transceivers to be continuously ON in anticipation of (unpredictable with respect to timing) receptions and for sensing the medium prior to transmissions. While combinations of random (ALOHA-like) and schedule (TDMA) scheduling have been proposed, the standard for the rest of our work is TDMA because of its immediate benefits insofar energy consumption is concerned.

The reasons mentioned so far show that duty-cycling is useful for WSNs. The next question though is whether each node should duty-cycle independently or not. If all nodes duty-cycle at the same time, and have the same period, then this is equivalent to alternately switching ON and OFF the entire network for some periods of time. This might make sense in some cases, but the main concern is that WSNs have actual traffic generation and delivery requirements. They need to both sense and inform (as soon as possible) the destinations. Hence, at least it is required that some subset of nodes is active at any point in time to monitor for traffic that might be spontaneously generated or sensor measurements that have to be performed and to be delivered. In other words, all the nodes are not required to wake up the same time. Fundamentally, what this also means is that duty-cycling impacts delivery delay.

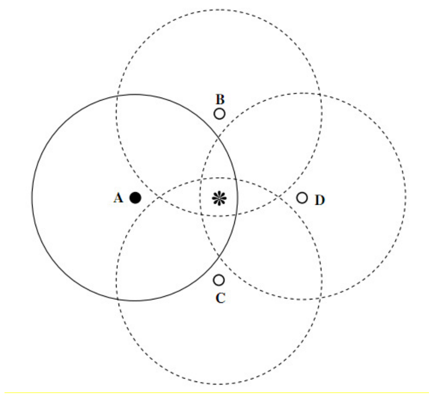
Another reason why turning OFF a transceiver at different time from those of the other transceivers is useful, beyond just avoiding idle listening/sensing of the medium, is because the energy of each node could be replenished differently by a renewable energy source (e.g., solar energy, wind energy, and kinetic energy) and depending on the node's location. In this case, the duty-cycle represents the time for which a node can confidently stay ON given the patterns of the energy replenishment. Although admittedly the degree of replenishment is a random process it is often quite predictable. For instance, Figures 1.1.a (adopted from [5]) show two possible power output variations with respect to time of a solar cell exhibiting a distinct diurnal behavior. Similarly, in Figure 1.1.b (adopted from [6]) the wind speeds at four arbitrarily chosen locations on an annual scale are shown. For example, if a renewable resource is used, and based on predictable replenishment patterns, each of the four nodes A, B, C and D in Figure 1.1.c can save energy by taking turns to sleep (a white dot) or stay awake (black dots) to sense continuously the same target area (the asterisk), because the areas of sensing (four big circles) of these nodes can all cover the target. In the example of Figure 1.1.c, even a single node being awake would have been sufficient in terms of covering the target area.



(a) Adopted from [5]



(b) Adopted from [6]



(c) Four nodes duty-cycling based on alternating sensing coverage

Figure 1.1. Illustration motivating why different nodes could duty-cycle at different times.

DC gives the possibility of energy savings due to no idle activities at the price of possibly long transmission delays caused by OFF-periods. Nevertheless, when DC is applied, the following three problems come up:

- First, since DC results in different network "links" to be present at different points in time (a "link" exists only if both adjacent nodes are ON at the same time), the topology of the network varies over time. Therefore, *routing* must be re-thought in this context, in that a path from a source to a destination might not exist at a particular point in time, but jointly (over time) the destination is reachable from the source. For practicality's sake, we will assume that all destinations are reachable from all sources in the network, but we have to be aware that this necessitates the construction of a time-varying-graph (TVG) routing path, i.e., a routing path where certain edges can only be traversed at certain (recurring, usually periodically) time periods. We consider the definition of topology to be consistent with the use of the term in TVG literature [60].

- Next, without transmission scheduling, the interaction of transmission attempts from nearby DC nodes may reduce the throughput of the network as a rush of packet transmissions happens when a link is active, i.e., both end points are ON. Using a random-access protocol like ALOHA may not be the best idea under the circumstances. Given that, as we will see, we know the DC parameters (and might even be able to control them), it should be possible to algorithmically schedule transmissions in a way that collisions can be avoided. We consider therefore a *scheduling* problem, in the sense of arranging transmissions to serve multiple flows when nodes have a link between them, i.e., during the (potentially brief) periods that their ON periods overlap.

- Finally, by switching nodes (transceivers) OFF, we inescapably reduce the attainable throughput to a certain degree. We therefore need to find ways to counter this *throughput reduction* by exploiting other mechanisms. In particular, we consider the application of Network Coding (NC), which will be discussed in detail later.

1.2 ASSUMPTIONS AND METHODOLOGICAL OVERVIEW

In our work, we assume that the traffic is well understood by the network designer. This is not an unreasonable assumption as certain types networks, such as WSNs, are built with particular applications in mind in terms of the kind of data and the pattern with which such data are produced, e.g., by virtue of periodically sampling a natural phenomenon. We also consider acceptable the use of centralized algorithms to decide how to configure and control such a network, in particular if its layout and physical topology is unlikely to change (as is the case with in many varieties of WSNs). Additionally, WSNs are routinely assumed to need the support of a powerful sink node, or at least some other infrastructure that gets to use the collected data or to further disseminate them. It is therefore straightforward to argue that such an infrastructure can be used to help in the design and configuration of a WSN using off-line centralized algorithms, possibly uploading to the sensors configuration and scheduling information. Based on the above-mentioned assumptions, it is totally natural to apply our work in WSNs because energy efficiency is crucial in the networks. However, leaving the issue of energy efficiency aside, our work is still applied in (generally)-wireless networks with multiple origin-destination pairs of traffic flows as long as the assummptions are satisfied.

To tackle the three main problems mentioned in the previous section, we use the following approaches. First, for the time-varying routing problem, we notice that the topology of the network also changes over time due to the duty cycles of the nodes. In other words, changes in the DC-network's topology can be described as Time-Varying Graphs (TVGs) and the changes are too regular and frequent to be modeled as "anomalies," i.e., network faults or failures, as pointed out by [7], in which TVG is the name of a unified framework outlined that integrates a vast collection of concepts, formalisms, and results in the literature. To simplify the problem in the early stage of study, we also assume the routing problem can be solved separately by a time-varying *fixed-single-path routing* algorithm as described in the work [8] of I. Chabini, [9] of B. Ding et al., and [10] of H. Chon et al, which in the future work could be extended to time-varying *multi-path routing* algorithms. Hence, the results of predetermined routing paths from the routing algorithm are used as part of the input to the subsequent scheduling algorithm.

For the scheduling problem, we use the TDMA skew/staggered scheme, in which time is divided into slots and all the nodes are synchronously scheduled in a contention-free manner. Particularly, they are not required to wake up the same time. For simplicity of description, let's assume that we are given the duty cycle characteristics of each node n, in the form of a period $\mathrm{T_n}$, the active (ON) period

α_n, and the phase ϕ_n of the active period, relative to time zero (of course, $\phi_n < T_n$) as described in Figure 1.2. With the skew/staggered scheme, phase ϕ_n of each node is usually different from each other. Besides, cycle period T_n and active period α_n of each node are not necessarily the same as those of the others. In other words, the duty cycle characteristics of each node n are expressed in our work by the $\langle \phi_n, \alpha_n, T_n \rangle$ tuple.

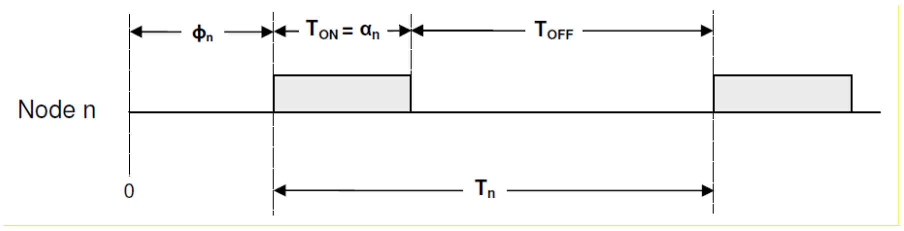


Figure 1.2. The duty cycle characteristics of node n

For the simplicity of presentation, in our thesis we will almost always assume that all T_n are the same and constant, i.e., $T_n = T$. For the same reason, we will also frequently assume that α_n is the same and constant, i.e., $\alpha_n = \alpha$. As described above, there are some reasons in having a non-constant α_n though, such as to fit the traffic load demands and/or energy reserves acquired in an energy replenishment phase (if powered by renewable resources). However, we note that our models and algorithms still work independently of the assumptions (details provided later in the thesis).

To further increase the network capacity, we use STDMA (Spatial TDMA), an extension of TDMA for multi-hop networks, which considers spatial reuse of timeslots for nodes or links spread out geographically. In STDMA, a cycle or a schedule of a fixed total number of timeslots is made up and repeated over time. The cycle is corresponding to a TDMA frame, in which a node or a link is allocated dedicated slots. The efficiency of the spatial reuse and hence the implemention of a STDMA protocol relies on the scheduling algorithm, which generates the schedule. Due to the differences of traffic load on links in multi-hop networks, the schedule needs to take traffic into consideration similarly to the traffic-sensitive scheduling algorithms in [11, 12, 13, 14]. More specifically, our traffic-sensitive STDMA scheduling algorithm is to allocate timeslots to transmissions from nodes, which depends on the link topology of the network and the traffic rates emanating from the nodes. Furthermore, our scheduling algorithm is not only to avoid collisions among the transmissions but also to "fairly minimize" the waiting time for transmissions at the nodes, i.e., to avoid penalizing particular flows. There are multiple criteria of how to fairly allocate bandwidth resources to the flow rate demands. In our work, we use the popular and well-studied criterion of max-min fairness.

Most importantly though, we attempt an unorthodox view of constructing TDMA transmission schedules for arbitrary topology networks whose nodes duty cycle (DC) in a periodic fashion. Namely, rather than exploring a purely algorithmic approach we use a simulation-based technique whereby, assuming all traffic sources are greedy, we expect a periodic behavior to develop at the steady state. We extricate this periodic behavior and, with minor adjustments, construct a template for the transmissions schedule. In short, our work [15] is based on the thesis that in a network with periodic topology changes (as is the case with duty cycling), fed by periodic traffic arrivals, and with deterministic behavior, the system's pattern of transmissions in the steady state ought to be a periodic one. Seeing how the network transmissions occur during one such periodic "cycle" in the steady state is a template which can be repeated continuously in the form of a transmission schedule.

For the purposes of establishing a performance benchmark against which to compare the constructed schedule, we consider a lexicographic maximization formulation of the network throughput problem, i.e., in essence max-min fairness objective. The lexmax formulation is implemented via our MP (max-min programming) algorithm, which works similarly to a classic water-filling scheme. The complete process is as follows: (a) first we use the MP algorithm to determine the per-flow rates, (b) the rates determined from the first step guide the generation of a slot-by-slot simulation, and (c) the steady state is detected and the (periodic) template of transmissions is extricated and forms the basis of the TDMA schedule. As long as the duty cycling behavior and the topology of the network is known, the described process can take place off-line and the resulting constructed schedule can be "downloaded" to the nodes for execution.

The process involves a number of non-trivial steps that can conspire to produce a less-than-perfect result. For example, step (b) is a simulation performed with global knowledge of the network topology and the state of the queues of all nodes, yet in order to produce a maximal number of concurrent transmissions (needed to exploit the spatial reuse in a large network) it resorts to using fast approximations to the Maximum Weighted Independent Set (MWIS). The approximation can result in "loss" of throughput compared to what would have been the per-slot optimal. Note that the complexity of the approximation MWIS algorithm by [21] used in our work is provided in Chapter 5. Additionally there exists a problem of discretization of the rate allocation derived from the water-filling algorithm which is typically a rational number but due to the nature of slotted schedule allocation it has to be rounded to a ratio of an integer number of transmissions over the (generally short) schedule length period. We add the fact that not all rates are schedulable by virtue of the underlying interference graph [16] and that even periodically scheduling slots to express different rates at a single node is a hard problem in its own right [17]. To compound the complication, the underlying communication graph is a TVG [18] and the reader can appreciate why a simulation-based is, after all, not as outlandish a strategy as it would appear at first. Our work is inspired by theoretical work in queueing systems, demonstrating the periodic behavior of the steady state of networks of queues (roughly corresponding to networks of nodes in our case) as described by Willie [19]. Albeit, our approach is a constructive one and it entails approximation steps because we need to maximize the concurrent transmissions subject to the constraints that links interfere with each other (contrary to the assumption of independent links in classical networks of queues), and that nodes operate in a half-duplex fashion.

To model the interference between simultaneous transmissions at the MAC-layer, we use the protocol interference model from the work [20] of K. Jain et al, according to which all possible simultaneous transmissions are mapped into vertices of a conflict graph. Hence, simultaneous conflict-free transmissions are corresponding to an independent set of the conflict graph at each timeslot. We consider a particular variety of this formulation to complement the MMF scheduling by means of MWIS calculation over the conflict graph (following the work [21] of S. Sakai et al.).

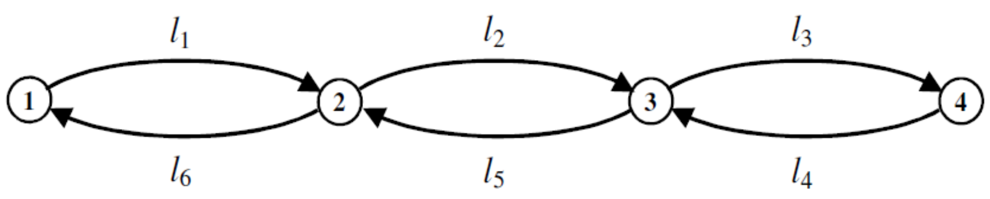
For the throughput reduction problem, we consider the application of XOR-pairwise NC approach from [22] with chain mode (detailed later), which is also a special case of linear network coding [23]. This NC approach was originally applied in wireless ad-hoc networks and certain conditions need to be met to make NC worthwhile, e.g., two neighbour nodes being able to receive broadcast packets from their common intermediate node. Even if these conditions are satisfied, however, it does not mean this approach can be applied in a DC-WSN, primarily due to the challenges of the time-varying topology. In a wireless network, network coding is necessary to improve the total throughput and also to reduce network congestion by combining packets destined for distinct users. To be more applicable in wireless networks, as mentioned earlier network coding should be accompanied with a scheduling algorithm, e.g., TDMA as proposed in our work, to avoid collisions. In a duty-cycled network, network coding is even more necessary because the time window for a node to access transmission media is shortened, which results in the total throughput reduction exhibited as network bottlenecks.

To gain more opportunities for coding with the XOR-pairwise coding, we design a delay coding scheme, in which each intermediate node waits for “valid” packets arrived, i.e., from its expected neighbor nodes, before coding. The delay coding scheme is feasible in our work because of the following assumptions: (1) all the routing paths of flows from sources to sinks are assumedly pre-determined, known, and unchanged during the system operation because the system is a deterministic one with fixed topology and given duty-cycles at nodes; (2) the combination of any two valid packets for the XOR-pairwise coding at each intermediate node is also assumedly pre-determined such that NC can be most beneficial. Note that to be able to solve the issue in implementation, we will propose, towards the end of the thesis, a heuristic approach of pairing possible flows at an intermediate node; (3) the sources of traffic of all flows are assumed greedy so that a source can introduce into the network as much traffic as it could by seizing any and all transmission opportunities afforded to it. Note that we will deal with the non-greedy sources of traffic at the end of the thesis. Next, we present notation and models for the wireless network, the duty cycling, and the traffic flows, as used throughout the rest of the thesis.

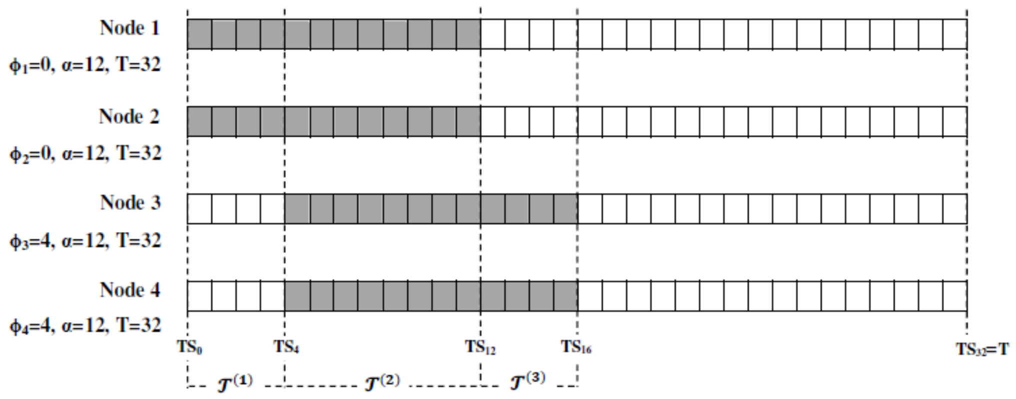
1.3 NOTATION AND DEFINITIONS

The duty cycled wireless sensor network is modeled as a set of nodes, $\mathcal{N}$, a set of directed links $\mathcal{L}$, and a set of flows $\mathcal{F}$. Unless stated otherwise, we assume that, between a pair of nodes, u and v, that are within communication range of each other, both links (u,v) and (v,u) exist in the link set $\mathcal{L}$, but the algorithms and mechanisms described in this thesis treat each direction separately, and hence apply even in cases where one of the link's direction is not present.

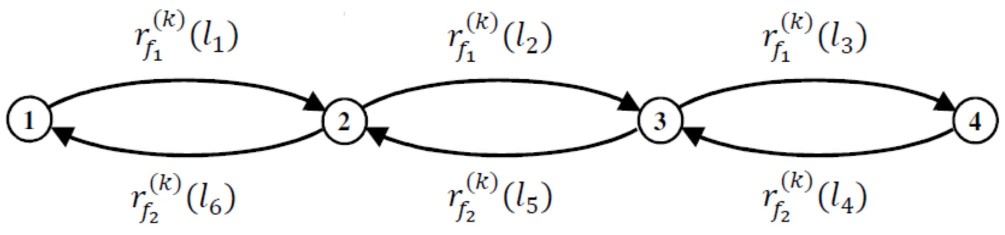
Each node is characterized by a triplet $\langle \phi_n, \alpha_n, T_n \rangle$, where T_n is its duty cyling awake+sleep (ON+OFF) period, ϕ_n is the phase, i.e., the instant within T_n at which the node switches to ON, and α_n is the duration of its awake (ON) period. For simplicity of presentation, in the following we assume *homogeneous* nodes, i.e., nodes that have the same period T_n (and we then denote it by T) and the same α_n (and we then denote it by α). Nevertheless, the techniques developed are, unless indicated, indifferent to whether the nodes are homogeneous or not. This is due to the fact that the introduced techniques have been developed around the definition of "stages" explained next.



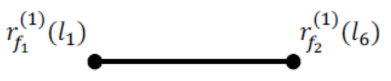
(a.1) Link topology



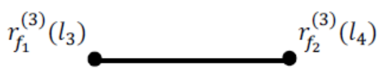
(a.2) States ON/OFF of nodes (ON slots are gray)



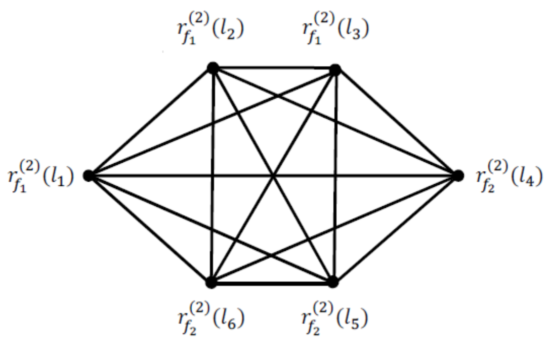
(b) Multi-hop flows



(c.1) CG during $\mathcal{T}^{(1)}$



(c.3) CG during $\mathcal{T}^{(3)}$



(c.2) CG during $\mathcal{T}^{(2)}$

Figure 1.3. An illustrative example of the defined notation.

The communication link between two nodes, within communication range of each other, is not continuously "present" as it depends on whether the two endpoint nodes are both ON. Time is assumed to be slotted and every packet has length of one slot. For computational convenience, we divide time into (a repeating sequence of) stages, $\mathcal{T}^{(i)}$, where each stage consists of a integer number of slots. A stage is defined as a sequence of successive time slots during which the duty-cycling state (ON or OFF) of all nodes remains the same, i.e., no node switches state. Hence, switching of duty-cylcing state occurs only at the boundary between stages. The computations, which are described in this thesis, are only concerned with the stages, $\mathcal{T}^{(k)}$, and for efficiency reasons only for stages in which at least one active (both endpoints are ON) link (across all nodes) exists. Note that due to the periodicity of duty cycling, the stages are also repeated in a periodic fashion. Hence, we can determine the number of such distinct and periodically repeating stages, which we denote by K. Stages $\mathcal{T}^{(k)}$ can be determined in polynomial time from the active periods of all nodes during cycle T using a simple sorting process.

Exampe 1.1. For illustration, let us consider a simple duty-cycled network with four nodes $n \in \mathcal{N} = \{1, 2, 3, 4\}$, six directed wireless links in the link set $\mathcal{L} = \{l_1, l_2, l_3, l_4, l_5, l_6\}$ shown in Figure 1.3.a.1. The DC-configurations, i.e., $\langle \phi_\mathrm{n}, \alpha_\mathrm{n}, \mathrm{T_n} \rangle$, of these nodes is as shown in Figure 1.3.a.2, which results in a sequence of three stages (K=3), in which the link topology is unchanged, repeating periodically with a period T. Note that for computational efficiency, duration from slot TS_{16} to slot TS_{32} in Figure 1.3.a.2 is not considered as a stage because there is no active link (both its ends are ON) during the duration.

We define each multi-hop flow $f \in \mathcal{F}$ as $f = (f_s, f_d)$, where $f_s = o(f)$ is the source node and $f_d = d(f)$ is the destination node. Unless otherwise noted, we assume that all traffic sources, hence all traffic flows, are greedy and attempt to get packets delivered to their destination at the highest rate possible. Each directed link $l \in \mathcal{L}$ is specified by a pair of nodes as $l = (l_s, l_d)$, where $l_s = o(l)$ is the origin and $l_d = d(l)$ is the destination of link l. We also assume that each flow is routed across a predetermined (single) path, which is a sequence of directed links / arcs. Note that a technicality we have to assume is that, for all the paths followed by flows, reachability from source to destination is guaranteed, i.e, there exist overlaps of the ON intervals between the pairs of nodes that define each arc used in the routing path of a flow. Nevertheless, such overlaps may be brief and hence not all arcs along a routing path of a flow can sustain the same traffic rate. The less the overlap, the more constrained ("bottlenecked") the possible achievable rate of the flow.

We define a sub-flow as part of a flow over a specific link/arc on its routing path during a particular stage. Specifically, given $\mathcal{F}, \mathcal{L}$ and $\mathcal{N}$ defined earlier, we define r_f to be the rate of flow f (hence, the rate from the source to the destination of the flow) and denote ${r_f}^{(k)}(l)$ as the sub-flow rate, i.e., the flow rate of f over link l during $\mathcal{T}^{(k)}$. Naturally, if not both of the link's endpoints are ON in a stage, the sub-flow rate of any flows using that link during that stage is zero. Note that with the assumption of single-path routing, each sub-flow corresponds to a single link on the path from source to destination. Hence, we denote (f, l) as a sub-flow of flow f over link l. For the sake of description, a link with both endpoints being active in a slot will be called an ON (or active) link in the slot. Otherwise, a link is called an OFF (or inactive) link. Correspondingly, sub-flows traversing an ON (or OFF) link are then called, respectively, ON (or OFF) sub-flows in a specific slot.

Two sub-flows that are ON in the same slot, contend with each other if the receiver end of one is within the transmission range of the transmitter of the other (and vice versa). Note that the interference model is the same as that in the work by Li [26]. The relationship of contending sub-flows is characterized by a sub-flow contention graph (CG) where vertices correspond to sub-flows and an edge between two sub-flows indicates the two are contending. We define as contention domain, in which each sub-flow contends with at least another sub-flow. Hence, there may be multiple contention domains in a sub-flow contention graph. The contention domains correspond to maximal cliques $q \in Q$ in the contention graph (where Q is the set of all maximal cliques in the contention graph). An implication of the time-varying nature of the graph is that the set of maximal cliques is also time-varying, because the contention graph changes to consist at any point in time of links that not only interfere because of the topology, but are also ON at that point in time. Note that a contending sub-flow can belong to multiple maximal cliques. An implication is that the maximal cliques determine the maximal rate that a sub-flow rate can be allocated among other contending sub-flows in the same clique. Finally, note that since the set of active links does not change during each stage, the contention graph remains the same throughout each stage, but is different from stage to stage.

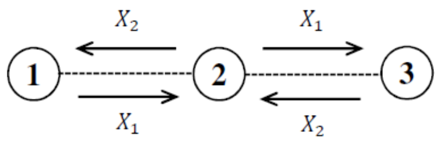
Returning to Example 1.1, note that we have defined two (opposite traversing) flows $f_1, f_2 \in \mathcal{F}$ following respectively the paths $\{l_1, l_2, l_3\}$ and $\{l_4, l_5, l_6\}$ indicated in Figure 1.3.b. During the stages, flow f_1 is represented by the three sub-flow rates $r_{f_1}^{(k)}(l_1)$, $r_{f_1}^{(k)}(l_2)$, and $r_{f_1}^{(k)}(l_3)$, and flow f_2 is represented by the three sub-flow rates $r_{f_2}^{(k)}(l_4)$, $r_{f_2}^{(k)}(l_5)$ and $r_{f_2}^{(k)}(l_6)$. As noted, the contention graph is different from one stage to the next. In this case, we expect that since we have three stages, there will be three distinct contention graphs, each of them "lasting" as long as the corresponding stage. Figures 1.3.c.1, 1.3.c.2, and 1.3.c.3, illustrate, respectively, the contention graphs in each stage for the given sub-flow graph of Figure 1.3.b and the duty-cycling behavior of Figure 1.3.a.2. Notice that in this particular example, the contention graph at each stage consists of a single maximal clique.

For the sake of implementing the contention graph in each stage, we store all the M contending sub-flows into a two-dimension contention matrix M_SF [0..M-1][0..M-1], in which entry M_SF $[i][j]$ indicates whether sub-flow i is contending with sub-flow j (e.g, presented by 1) or not (e.g, presented by 0). Note that the contention matrix is unchanged after it is created while the contention graph can change from one timeslot to the next.

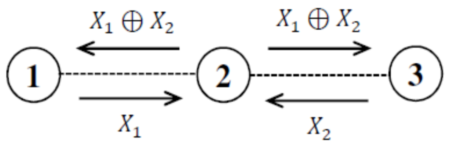
1.4 ADDITIONAL NOTATION FOR NETWORK CODING

In Chapter 4 we use XOR network coding between pairs of flows. An example is shown in Figure 1.4.c. It is a coding scheme similar to the chain mode of COPE [22], which introduces two main types of network coding, one without opportunistic listening and the other with oppornistic listening (more details described later in Section 2.5). In other words, we consider a simple linear NC without opportunistic listening, where router nodes along a routing path (e.g., nodes from 2 to n-1) broadcast a combination of at most two packets, i.e., potentially one out of every two transmissions can be saved. Packets are coded at router nodes using an XOR operation and the original packets are decoded at each node that receives the coded packet (of two combined packets) and has previously transmitted one of the two packets combined in the coded packet, i.e., it is a scheme of hop-by-hop decoding (more information with an illustrative example and a delay coding algorithm will be provided later in Section 4.2.1).

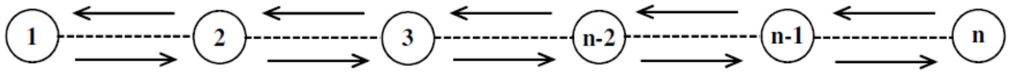
In Chapter 4 we draw the distinction between two packet types: native packets (i.e., non-coded packets), and coded packets, i.e., created by XORing two native packets. In the example of Figures 1.4.a and 1.4.b, two native packets X_1 and X_2 are shown. The receiver can decode a coded packet, e.g., $X_1 \oplus X_2$, by XORing the received packet with one of X_1 or X_2, to obtain the other one. Hence, over a given link, we can distinguish flows of coded packets and flows of native packets. The native/non-coded sub-flows remain as defined earlier, i.e., they express the flow of (native) packets on an arc of the path of a flow. The coded packets, on the other hand, and the *coded sub-flows* we introduce, are relevant to two sub-flows, namely subflows of different flows that traverse a link in opposite directions. Whereas a native sub-flow's transmissions are received by a single receiver, the coded sub-flow's transmsisisons are to be received by two receivers.



(a) Transmissions *without* NC



(b) Transmissions *with* NC



(c) Topology with XOR-pairwise NC

Figure 1.4. An illustrative example of XOR-pairwise network coding.

For the sake of completeness, note that in a DC WSN with XOR-like network coding, a coded sub-flow can be used by a node if two necessary conditions are satisfied: (1) there exists a common overlap across the active periods of the three nodes, i.e., of the transmitter and of the two receiving nodes of the coded sub-flow; (2) there exist two opposite traversing sub-flows crossing the node, i.e., they belong to two different multi-hop flows traversing in reverse directions across a common sub-path (a sub-path consisting of two or more hops) in which common sub-path the transmitting node is one of the hops.

In our work, given all the flows and the duty-cycles of nodes are known, we assume that all the single-routing paths of the flows are calculated in advance. Hence, we can pre-determine the coded sub-flows that satisfy the two conditions and hence pre-determine which nodes will be transmitting coded sub-flows in addition to native sub-flows. Technically, this means that we can tell in advance which combination of two non-coded sub-flows could be used to generate a coded sub-flow.

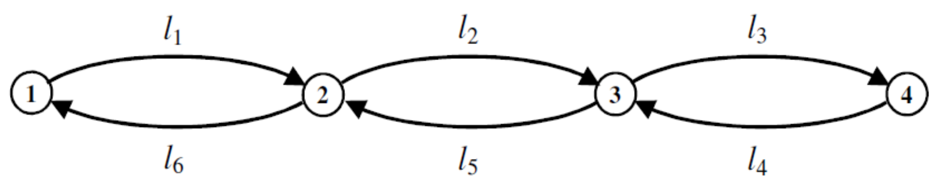
The best combination of any two non-coded sub-flows into a coded sub-flow (if it is indeed possible) can be determined in advance by enumerating all possible pair-wise combinations and then selecting the combinations that can make the best benefits of NC (details later). Because of the combinatorial nature of this approach, we will use in Chapter 4 a restricted selection criterion for the pairs of flows we will code, and discuss alternatives in Chapter 5.

We extend the notation introduced earlier to capture the notion of coded sub-flows. For any $f_1, f_2 \in \mathcal{F}$ and any $l_1, l_2 \in \mathcal{L}$, we denote $r_{f_1,f_2}^{(k)}(l_1, l_2)$ as the rate of flow f_1 transmitted over link l_1 and the rate of flow f_2 transmitted over link l_2 using NC during $\mathcal{T}^{(k)}$. Hence, we define notation (f_1, f_2, l_1, l_2) of a coded sub-flow combined from flows f_1 and f_2 over links l_1 and l_2, respectively.

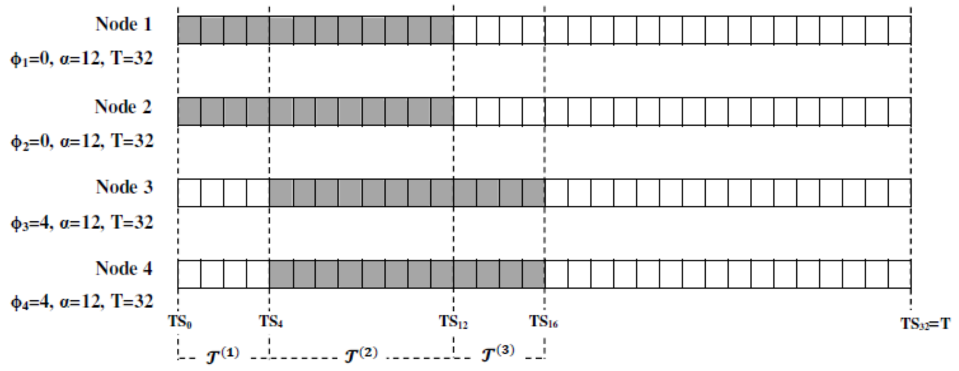
This new variable can only be defined if the two links l_1 and l_2 are both simultaneously active (corresponding to necessary condition (1) above) and such that $o(l_1) = o(l_2)$ and $l_1 \neq l_2$ (corresponding to necessary condition (2) above). Variable $r_{f_1,f_2}^{(k)}(l_1, l_2)$ represents the rate of f_1 (resp. f_2) that is network-coded and transmitted over l_1 (resp. l_2) during $\mathcal{T}^{(k)}$. During each $\mathcal{T}^{(k)}$, let $r_f^{(k)}(l)$ denote the rate of flow f over an active link l, also called the (native) sub-flow rate of flow f, i.e., corresponding to packets that are not encoded with NC. Also, let $\bar{l}$ be defined as the reverse link of active link l, i.e., the ON link such that $o(\bar{l}) = d(l)$ and $d(\bar{l}) = o(l)$. In principle, in Chapter 4 we allow each node to relay part of each flow it receives with NC (as a coded sub-flow rate) and the remaining part without NC (as a native sub-flow rate).

The definition of contention we described earlier, and hence that of contention graph between sub-flows (and hence the definition of the contention matrix), still holds if we extend it to now include coded sub-flows as well. Each coded sub-flow is also a vertex in the contention graph (or an index in the contention matrix). The difference is that a coded sub-flow has two receivers, and hence is in conflict with any other sub-flow (coded or native) which would result in collision at any of its two receivers.

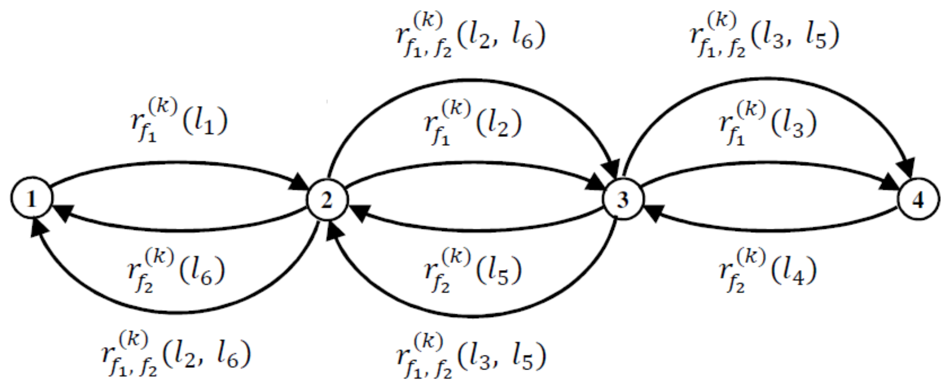
For illustration, let us consider again the simple duty-cycled network with four nodes $n \in \mathcal{N} = \{1, 2, 3, 4\}$, six directed wireless links in the link set $\mathcal{L} = \{l_1, l_2, l_3, l_4, l_5, l_6\}$ shown in Figure 1.5.a.1, and two flows $f_1, f_2 \in \mathcal{F}$ traversing in opposite directions along two single-routing paths $\{l_1, l_2, l_3\}$ and $\{l_4, l_5, l_6\}$ indicated in Figure 1.5.b. The DC-configurations, i.e., $\langle \phi_n, \alpha_n, T_n \rangle$, of the nodes is shown in Figure 1.5.a.2, and the period T is divided into three stages $\mathcal{T}^{(k)}$. As shown in Figure 1.5.b, during each stage k, flow f_1 is now represented by five sub-flow components $r_{f_1}^{(k)}(l_1)$, $r_{f_1}^{(k)}(l_2)$, $r_{f_1, f_2}^{(k)}(l_2, l_6)$, $r_{f_1}^{(k)}(l_3)$ and $r_{f_1, f_2}^{(k)}(l_3, l_5)$, and flow f_2 is represented by five sub-flow components $r_{f_1}^{(k)}(l_4)$, $r_{f_1}^{(k)}(l_5)$, $r_{f_1, f_2}^{(k)}(l_3, l_5)$, $r_{f_1}^{(k)}(l_6)$ and $r_{f_1, f_2}^{(k)}(l_2, l_6)$. Hence, the sub-flow contention graph made up from the all the coded and native sub-flow components present in each stage $\mathcal{T}^{(1)}$, $\mathcal{T}^{(2)}$, and $\mathcal{T}^{(3)}$ correspond to Figures 1.5.c1, 1.5.c.2, and 1.5.c.3, respectively.



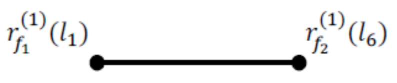
(a.1) Link topology



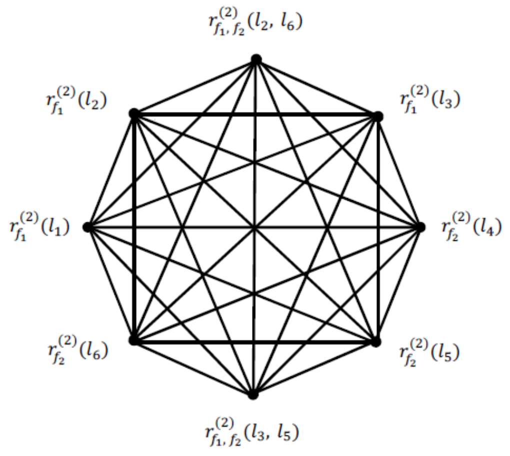
(a.2) States ON/OFF of nodes (ON slots are gray)

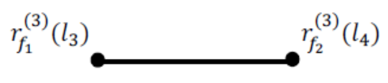


(b) Multi-hop flows



(c.1) CG during $\mathcal{T}^{(1)}$





(c.3) CG during $\mathcal{T}^{(3)}$

(c.2) CG during $\mathcal{T}^{(2)}$

Figure 1.5. An illustrative example for a DC-WSN with NC.

1.5 BROAD OBJECTIVES AND RESEARCH QUESTIONS

Our overall objectives are to formulate and improve where possible the performance of a DC WSN. In other words, we investigate the trade-off of throughput and energy consumption under DC configurations. However our ability to formulate the performance is approached from a pragmatic stance, i.e., what is the performance of an actual constructed schedule given the particular DC WSN. As we will see in the thesis, the rates established through an optimization solution need to be expressed as a schedule and a degree of success is whether the constructed schedules indeed abide by the rate specifications imposed on them.

One aspect is the question of formulation of the relation between duty-cycling and achievable throughput, as well as the related energy efficiency (if there is any). This line of inquiry accepts that we do not have control over the DC of the nodes, and it is given to us as-is. In this case, the best we can hope for is that we are making the best use possible of the particular setup. Naturally, we focus first on the ability of such a network to attain a level of throughput, or to be more exact, to ask whether a system with a particular DC configuration can support a given set of flow with greedy demands.

A second aspect is a question of mitigation of any throughput reduction in a DC network by means of using NC. Here, as in the previous paragraph, we are trying to formulate what would be the throughput ability of the system if, additionally, we employed NC. The reason we look into NC as a potential solution is that it is in agreement with the primary objective of saving energy. The combination of multiple packets into coded NC packets has the potential to save energy, and hence it is a throughput improvement technique that aligns well with the energy savings intent of DC.

1.6 CONTRIBUTIONS AND THESIS OUTLINE

We have sought to answer the broad questions above-mentioned in the thesis, through the following contributions:

- In our first attempt to answer the second research question (which predates the rest of the thesis content), we proposed a DC-NC protocol at the MAC layer applicable to a limited set of topologies, which is used for both energy savings and throughput improvement in duty-cycled WSNs. Associated with the protocol, a heuristic NC coding algorithm with opportunistic listening was also proposed.

- In order to broaden the applicability of our schemes to general topologies, we proposed a general approach of modeling time-varying topologies of duty-cycled networks via node phases by dividing node cycles into unchanged-topology stages, in which numerical rates of multi-hop flows can be expressed as solution of lex-max formulations (assuming pre-determined single-shortest-path routing) and computed via the MP (max-min programming) algorithm.

- We subsequently introduced a novel simulation-based approach of constructing TDMA transmission schedules for arbitrary topology networks whose wireless nodes duty cycle in a periodic fashion. This was performed by exploiting the periodic steady state which induces (approximately) a periodic pattern of transmissions. This approach was successful because of the combined impact of the periodic network behavior and the determinism in determining which nodes transmit, and for which flow, at each timeslot.

- Back into the domain of the second research question, we proposed a generaliation of the modeling approach and the schedule construction to include NC, in which we have noted energy consumption improvement potential of even 23%, and total throughput improvement as high as 50%. Towards this goal, the lex-max formulations solved by the MP algorithm have been extended to capture the impact of NC. Particularly, a delay coding scheme was proposed to extract better performance out of NC.

- We also proposed an approach (equivalent to changing the DC configurations) of how to turn off active spots of the flow in the template from the receiver side when the average traffic load is actually much less than expected. This extends the applicability of the technique to cases where the traffic is not behaving in a greedy fashion. Further extensions to accommodate heterogeneous duty-cycles of nodes, straightforward maximum buffer occupancy calculations, and heuristics for flow selection to apply NC are described as well.

This thesis is split into six chapters. The present chapter introduces the scope and motivation of our study, the overall methodological approach and assumptions. The notation and definitions for the rest of the thesis are also presented in the chapter. The remaining chapters are organized as follows:

Chapter 2 describes the earlier work of five main areas in the literature that are related to our study. It starts with the work of duty-cycling (DC), in which DC taxonomy is used to classify our DC scheme. Then the chapter highlights popular TDMA and STDMA algorithms, which take traffic intensity or/and max-min fairness (MMF) into consideration as our scheduling algorithm does when constructing a schedule. The concept of the periodic steady state is presented referring to the existence and proofs of the periodic steady state of a multi-server loss system. Finally Chapter 2 incorporates our early work on combining DC and NC, albeit for particular topology and with a specific non-TDMA MAC protocol.

Chapter 3 presents our general TDMA scheduling algorithm for DC WSNs and the process of constructing a schedule template, which is later downloaded to nodes for use. It starts by modeling time-varying topologies of duty-cycled networks via what we call “stages”. A lex-max formulation of the rate assignment problem is shown alongside the MP algorithm used to solve it. The derived rates are used as targets for the scheduling algorithm with the help of an approximate maximum weighted independent set (MWIS) algorithm which attempts to concurrently schedule as many sub-flows as possible in each slot, based on their weights. After the periodic behavior of sub-flow transmissions is detected at the steady state, a schedule template (with minor adjustments) is constructed.

Chapter 4 provides the extension of modeling time-varying topologies of DC networks to the use of NC, with the intention of exploiting NC to improve the total throughput. It starts with the extension of the MP algorithm associated with lex-max formulations so that it is applicable when NC is used. The chapter also describes the extension of the scheduling algorithm associated with the excision process for constructing a schedule template with minor adjustment. It finally presents our delay coding scheme, which is designed to further exploit the potential of NC.

Chapter 5 outlines extensions and generalizations of our work. Namely, it includes: the impact of non-greedy traffic from sources of flows, heterogeneous duty-cycles of nodes, and elimination of idle listening when the traffic is less than what anticipate by the greedy assumption. Additionally, the chapter describes proposed algorithms to compute the maximum length of transmission buffers and to select a suboptimal coding combination of possible flows in the case of NC. A review of the computional overhead of the MWIS algorithm, which is used in each timeslot, is also provided.

Chapter 6 briefly describes and discusses our contributions, and their limitations. We have unearthed several interesting and challenging open problems during the research for this thesis, which are also described in the end of this final chapter.

CHAPTER 2
RELATED WORK

2.1 DUTY CYCLING CLASSIFICATION

The work [24] of R. C. Carrano et al. proposes a taxonomy of DC mechanisms in WSNs. DC protocols are classified based on their fashion, either *synchronous* or *asynchronous* or *semi-synchronous*, of coordinating the schedules of sensor nodes as depicted in Figure 2.1. The class of synchronous schemes is divided further into two branches, *rendezvous* and *skew/staggered*. In the first branch, all the nodes have to turn their radio ON and OFF at the same time, which requires strict time synchronization globally among the nodes. Meantime, in the second branch each node is scheduled to wake up in a "ladder" pattern, according to its depth in the topology tree. The skew/staggered wake-up approaches are mainly aimed to reduce the end-to-end delay and particularly the Data Forwarding Interrupt Problem described in the work of G. Lu et al. [25], which happens because an upstream node, unaware that a frame is to come, goes to sleep before this frame arrives and therefore causes the data flow be interrupted.

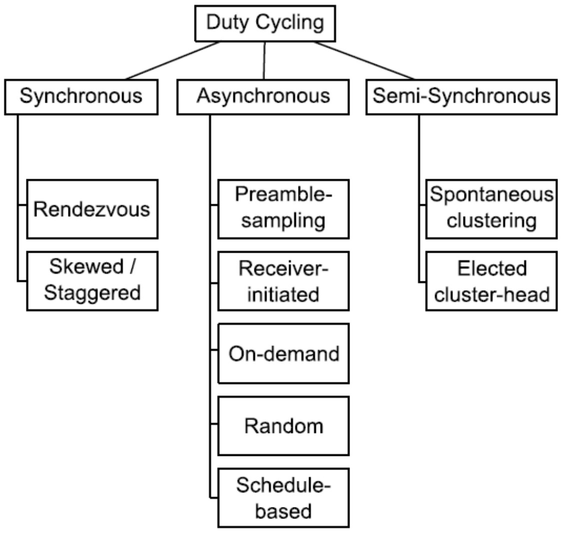


Figure 2.1. Taxonomy for DC techniques (adopted from [24])

The scheduling approach we develop in this thesis assumes that time is divided into slots and all the nodes are synchronously scheduled in a contention-free manner. Hence, it is a synchronous scheme. Furthermore, the nodes in our scheme are not required to wake up the same time and hence our scheme could be classified into the skew/staggered schemes. However, our scheduling scheme also assumes particular traffic behavior, mainly greedy sources of traffic (with some refinements made in Chapter 5), and hence scheme is assumed to operate continuously rather than invoked on-demand. The traffic is important in our work because it influences the construction of the schedule, i.e., we need to derive per-flow rates to create a schedule. Hence, a missing dimension not expressed by the taxonomy in [24], yet important for our work is that of how to treat various traffic sources, or, more specifically in our case, how to treat various flows. Consequently, our work can be characterized as a synchronous skew/staggered scheme but additionally it is traffic-aware.

2.2 TDMA SCHEDULING

There are two broad groups of MAC (Medium Access Control) protocols in wireless networks, and hence in WSNs: contention-based access protocols and TDMA (Time Division Multiple Access) protocols (including protocols that mix elements of both). Contention-based MAC protocols, starting from ALOHA, attempt transmissions on an as-needed basis which is fit for varying traffic demands but forces the transceivers to be continuously ON in anticipation of (unpredictable with respect to timing) receptions and for sensing the medium prior to transmissions. On the other hand, with TDMA protocols, time is divided into slots and each node or link is assigned to dedicated slots in order to avoid collisions. While this family of protocols is excellent in terms of energy consumption due to the fact that a node can disable the transceiver when no traffic needs to be sent or received, they are considered as inflexible because they do not adapt to random fluctuations of the traffic load.

In an attempt to increase the network capacity, STDMA (Spatial TDMA), is an extension of TDMA for multi-hop networks that exploits spatial reuse of timeslots for nodes (or links) that are spread out geographically. This idea makes a lot of sense in a WSN, where we frequently assume multi-hop flows [26, 27] and single-hop flows may not be feasible nor power efficient because of the scale of the network [28]. In STDMA, a cycle or a schedule of a fixed total number of timeslots is made up and repeated over time. Such techniques have a long history, starting with the work of Nelson and Kleinrock [29] and Kleinrock and Silvester [30]. During the TDMA cycle a node or a link is allocated a dedicated slot. The main advantage is that due to spatial reuse in multi-hop networks, multiple nodes or links in STDMA can share a single slot without collisions. The efficiency of the spatial reuse and hence the implemention of a STDMA protocol hinge on the scheduling algorithm that generates the schedule. Due to the differences of traffic load on links in multi-hop networks, a schedule needs to be constructed that honors such load differences. Some traffic-sensitive scheduling algorithms as in [11, 12, 13, 14] have been proposed by making the schedule more adaptive, i.e., giving more time slots to a node or a link when it carries more traffic. An immediate benefit of contructing an STDMA schedule that honors the various traffic needs is that the average delay can be decreased considerably [12, 13, 14] as nodes (links) do not have to wait an entire cycle until the next scheduled transmission opportunity.

As an example of traffic-sensitive STDMA, [11] allocates slots depending on the amount of uniform and point-to-point traffic passing through each node, which is calculated by summing the load of all the outgoing link of that node. The algorithm schedules the nodes in order of node ID and allocates the number of slots to each node proportional to its traffic load. After a node's required slots have been scheduled, the algorithm attempts to schedule as many additional nodes in this set of slots as possible. Instead of the node-oriented slot assignment of [11], a link-oriented traffic-sensitive STDMA scheduling algorithm was proposed in [12]. We adopt the same approach, i.e., link-oriented schedule construction in our work because the link assignment is preferable to node assignment for its ability to attain higher throughput as described, e.g., by Gronkvist in [14]. In fact, the algorithm proposed in [12] is a combination of [29] (to build a basic schedule) and [11] to allocate extra slots to heavier loaded links.

We pay particular attention to the work of Gronkvist [13, 14] that proposes a traffic-sensitive scheduling algorithm very close to our assumptions. The algorithm generates a STDMA schedule, in which each link is allocated a guaranteed number of slots to carry over its traffic load. Each link has a priority, whose value is proportional to the two factors, the relative traffic on the link and the number of timeslots since the link was previously allocated a timeslot. This is simply a ranking heuristic of the links, which bears some resemblance to the maximum weighted independent set approximation we adopt in our scheme. The algorithm is further improved for higher throughput by a novel assignment strategy [14] in which the actual traffic in queue at the transmitter node of each scheduled link is taken into account. However, [13] and [14] are different from our approach in that they are unable to schedule enough slots to accurately (or exactly for that matter) capture the rate demands of different flows (and hence of different links). The shortcoming is a result of their reliance on only the relative traffic load of the links instead of the actual, absolute, traffic (as in our algorithm). In fact the rounding of relative loads (to generate integer multiples of a normalized rate) results in the link's rate in the constructed schedule by [13, 14] to be different (more or less) by a non-trivial amount to what it should be. An example of a simple topology and the result of Gronkvist's scheme versus the scheme we outline in Chapter 3 are provided in Appendix A.

2.3 RATE FAIRNESS

There are multiple criteria of how to fairly allocate bandwidth resources to traffic flows. In our work, we use the popular and well-understood criterion of max-min fairness. Informally, an allocation of a shared resource among multiple participants is called max-min fair if (1) it is feasible and (2) an increase in the allocation to any participant must result in a decrease in the allocation to some of the other participants with equal or smaller allocation.

We assume that first the rates are calculated that satisfy the fairness criterion, and then a schedule is constructed that stays faithful to those rates. This makes our work similar to that of [31] and [32]. However, our work is different in certain key assumptions: (1) we assume that the flows, by virtue of being multi-hop, traverse multiple contention domains and are treated as such, as opposed to an often used simplifying assumption that all flows are single-hop flows, or equivalently confined to a single contention domain; but more importantly, (2) the link topology in all known previous work is unchanged over time while the underlying topology in our work is time-varying graph due to DC.

Similar to the approach of [31, 32], the scheduling algorithm by Tassiulas et al. [33] also provides max-min scheduling in wireless ad-hoc networks and their algorithm only considers single-hop flows. However, they do not need to explicitly compute the max-min rates because the max-min fair allocation of sessions is ensured by a token-based protocol that determines which links are activated. Technically, their algorithm operates using a weight assignment (via the token mechanism) and by subsequently solving the underlying maximum weighted matching problem. However, their work is different from ours in that the constraint of the single-hop contention in their work is defined differently from that of our work. In their model, any two single-hop flows not sharing a node can simultaneously transmit packets. This is because each node is assumed to have a locally unique radio frequency or a transmission code and therefore transmissions that do not have a common reception node can simultaneously carry on without any interference. In other words, the simultaneous transmissions in each timeslot constitute a matching, which is a set of edges such that no two of them have a common vertex. Meanwhile, in our model any two single-hop flows within two hops are contending with each other. More importantly, as in [31, 32], their work does not consider the case of DC and hence of TVG topology.

In conclusion, the work in [26] bears also similarities to our work in the sense that its purpose is to maximize the spatial reuse while maintaining a type of fairness among multiple multi-hop flows. However, the definition of the fairness enforced in [26] is different from the commonly understood max-min fairness and eventually develops an algorithm which is controlled by a choice of weights to each flow which does not reflect the max-min fairness. Finally, as is the case in all previous work we have studied, the algorithm in [26] does not consider situations of time-varying topologies such as those encountered in DC WSNs. In summary, and to the best of our knowledge, there has not been any work in the literature where a time-varyling topology of the kind found in DC WSNs was incorporated into the fairness definition, and subsequently in the schedule construction process.

2.4 PERIODIC STEADY STATE

It should be clear at this point, that the periodic changes in the topology due to DC are a source of complications. In this section we outline what can be described as a benefit of this periodic behavior, which then plays a vital role in the technique we develop for STDMA schedule construction. As already pointed out in Chapter 1, the network topology behavior is a periodic repetition of stages. Consider now that a per-slot scheduling scheme has been developed (we later point to this being based on an MWIS approximation) that behaves in a deterministic fashion. Add to the configuration a non-random behavior of the traffic sources, such as the greedy behavior we assume. The complete picture is that of a system with periodic and deterministic behavior. As its dynamics will unfold over time, it ought to behave in a periodic fashion, rather than randomly. It is precisely this conjecture, of the periodic behavior of the system as a whole, on which we base the schedule construction technique.

Specifically, the existence, and ways in which a periodic steady state behavior of a queueing system, can be developed has been the subject of research in queuing networks. For example, [34, 35, 36, 37, 38] show the existence of the periodic steady state of a multi-server loss system is indeed the case, albeit under two conditions, the periodic (hence non-random) inputs having a special structure, i.e., the arrival process is a Poisson process with a deterministic, periodic intensity function, and the service- and arrival times are independent. However, the independence assumption was subsequently dropped off in the work by Willie [19] which had important ramifications to the applicability of the result as in networks of queues the queues are generally correlated. Furthermore, [19] showed that the specific structure of the periodic Poisson inputs is also unnecessary and the only sufficient condition is that the traffic inputs are periodic. An illustration of the periodic steady state is shown described in Figure 2.2 indicating the queue state of several queues developing over time. Regardless of initial conditions, they behave periodically, i.e., their combine state exhibits periodicity.

There are differences between the system considered in [19] and the one in our work, which lead us to treat the anticipation for the periodic steady state as purely a conjecture. However, from the results in Chapters 3 and 4, it appears that our conjecture holds. From the differences between [19] and our work, the one with most impact is that in a queueing system like that of [19] an arrival at a queue can occur independently of any other arrivals at other queues, i.e., the "transfer" of one job from one queue to the next does not block (or enable for that matter) the "transfer" another job from another queue to another one. This behavior is similar to what one would find in a wired network, if we were to see the nodes of the network as queues and the links as the links between queues. However, in a wireless setting operating under a collision free schedule, one transmission ("transfer" from queue to queue) inhibits other such transmissions/transfers from taking place.

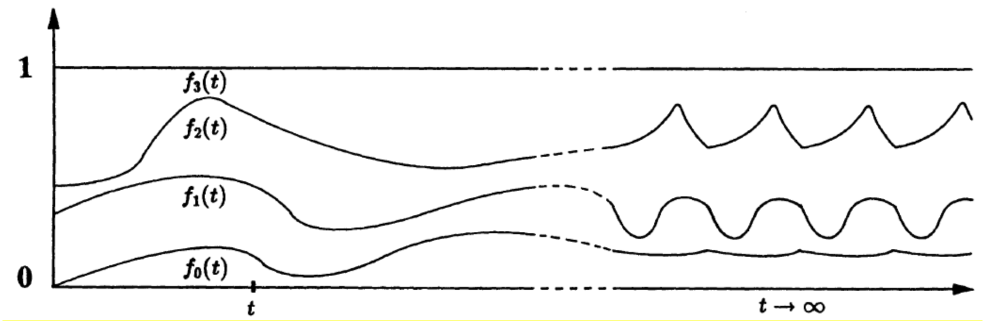
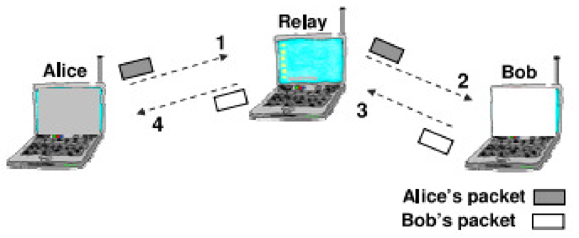


Figure 2.2. Periodic steady state for periodic input
(adopted from [19])

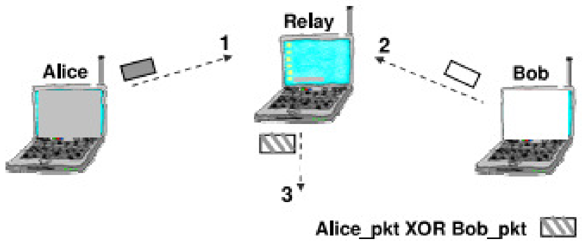
2.5 NETWORK CODING

Network coding (NC) is a technique, which is used to improve the total throughput and save energy. It is performed by combining multiple transmissions into one. The throughput achieved by NC is upper-bounded by the minimum capacity of all cuts according to the max-flow min-cut theorem [39]. It is known that the upper-bound and a respective set of edge-disjoint paths can be achieved using traditional routing (with polynomial time algorithms) in a unicast or broadcast network. However, in a multicast network, the upper-bound cannot be attained using store-and-forward routing but via adding computational tasks, i.e., network coding, performed at intermediate nodes [40].

One of the most distinctive features to differentiate wireless technology from wired technology is its broadcast nature. This can cause packet collisions if a contention-based MAC protocol is used. In order to apply NC in a wireless network, it is preferable to use a scheduling algorithm to minimize such interference/collisions. Otherwise, any gains from NC in a wireless setting are negated by the existence of collisions. Even more limiting than wired networks, wireless nodes cannot transmit and receive simultaneously, i.e., they behave in a half-duplex fashion. Though NC was originally introduced to be used at Network layer of the OSI model, in wireless networks, however, it has found applicability at the MAC or Physical layer [41, 22]. A simple illustrative example in the wireless context with three node topology is shown in Figure 2.3.a (taken from [22]), in which Alice (or Bob) wants to exchange the packet 1 (or 2) via intermediate router. Instead of broadcasting packets 1, 2 in sequence by the traditional method (Figure 2.3.b), the router broadcasts 1 XOR 2 so that both Alice and Bob can recover the packet of interest while the transmission cost can be reduced. This method is a simple form of NC.



(a) Current Approach (adopted from [22])



(b) COPE (adopted from [22])

Figure 2.3. An Illustrative example of network coding

Since DC and NC are two techniques that can be used to save energy in a WSN, it makes sense to combine them together. However, the efforts in this direction have been few and far between. Our original work (Ho and Nikolaidis [42] is described in some detail in Section 2.6) as well as the work of Chandanala et al. [43] and Rout and Ghosh [44] are the only exceptions we are aware of.

Similar to the work in this thesis, [44] uses pairwise XOR NC, which is a special case of linear NC [23]. The advantage of XOR NC is its simplicity. Also, in our earlier work [42] as well as in [43], opportunistic coding, in the vein of the one by Katti et al. [22], is used in that all destined receiver nodes need to overhear and store packets transmitted from their neighbor nodes for decoding. The advantage of opportunistic coding over the pairwise XOR NC is the ability to combine multiple transmissions (>2), i.e., more than one transmission can be saved. However, an intermediate sender must be able to broadcast to multiple destined receivers with valid packets for decoding, which is highy dependent on a specific application's traffic and not always feasible.

The work in [43] considers the simultaneous use of DC and the opportunistic NC for aggressively saving energy. In order to do this, it exploits the packet redundancy which happens particularly in flooding applications. Then, a node is put to sleep, i.e., DC is used, when a redundant transmission happens, i.e., after the node has received and successfully decoded a sequence of coded packets. Their work is different from our work in that they assume there is a single destination/sink and all senders' traffic is destined to this single sink. Instead, we assume the more general case that each flow can have a different originating and destination node. Additionally [43], having no real notion of competing flows, does not consider any fairness in terms of the rate allocation across flows.

Finally, in [44] a combination of DC and NC is proposed to enhance the lifetime of WSNs. To this end, [44] focuses on improving the energy efficiency of the bottleneck zone around the sink node and hence leads to overall improvement of the network lifetime. The work in [44] differs from this thesis in that, again, the traffic is destined to a single node, but additionally, the use of combined DC and NC addresses specifically the region around the sink, while we assume its use throughout the network. Additionally, [44] relies on a simple random DC, which appears to have served only in the development of an analytical understanding of the upper-bound of the network lifetime, rather than be justified on operational reasons.

2.6 EARLY RESULTS

In this section we present some early results obtained at the beginning steps of this thesis that have also appeared as a conference publication [42]. In this early work we attempted to direct our attention directly to the combined DC+NC problem. Given the complexity of this endeavour, certain simplifications had to be introduced, but the overall intention was to explore whether there is indeed potential in combining DC+NC; a question that we answered in the affirmative. In the following we describe the assumptions that were used in this earlier work. The reader is forewarned that these are assumptions pertaining only to the work described in Section 2.6, while the rest of the thesis obeys the assumptions put earlier in the introduction.

Specifically, in this early work we consider networks where the underlying physical communication graph is completely connected, i.e., each node is within communication range of each other. Additionally, we assume that the application implemented is that of sensing coverage, and that it is sufficient for one of the nodes to be active for the sensing to take place, i.e., the nodes "take turns" (in the sense of their DC) such that there is at least one node in the ON state at any point in time; a scenario familiar to applications where the cost of sensing needs to amortized across a number of co-located nodes. Given that there exists a single contention domain (and hence a single clique of contending nodes at any point in time), the question of STDMA scheduling (its 'S' aspect) becomes moot. Instead we describe a distributed MAC protocol to control the transmission instants.

2.6.1 SCHEME OVERVIEW

To illustrate how this early scheme works, let us consider a cluster of four wireless sensor nodes, 0, 1, 2 and 3, in which each node is within the communication range of others (Figure 2.4.a, 2.4.b, and 2.4.c). To save energy the transceiver of node 1 in Figure 2.4.a (or node 2 in Figure 2.4.b) is intentionally turned off (represented by a white circle). Let us assume that because it is OFF it cannot receive any incoming packets and thus misses packet x_1 in Figure 2.4.a (or packet x_2 in Figure 2.4.b) from node 0. However, nodes 3 and 2 in Figure 2.4.a (or nodes 3 and 1 in Figure 2.4.b), which are in the ON state (represented by a black circle) may overhear these missed packets. Later, node 3 can XOR the two packets x_1 and x_2 and broadcast the result, when both nodes 1 and 2 are turned ON (Figure 2.4.c). Nodes 1 and 2 can then obtain each other's packet by XORing the XOR-ed packet with their overheard packet. Thus, a transmission can be saved here because both nodes 1 and 2 can regain their missed packets just from a single transmission.

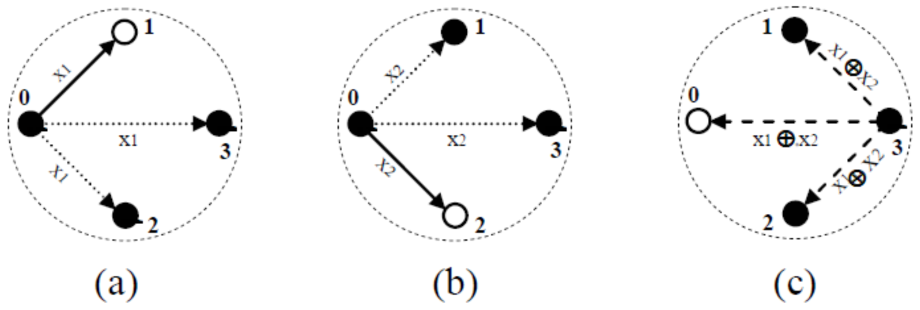


Figure 2.4. Illustration of how the scheme in [42] operates.

The main requirements that are synthesized to produce the DC+NC combination are: (1) the load of produced/incoming packets (think of them as data generated by sensing) should be distributed evenly across the clique's nodes so that every node will consume approximately equal energy in acting as "representative" of the clique, (2) there should always be some node in the ON state to receive incoming packets for the clique, (3) to be meaningful, the ON period of a node must overlap, at some point, with the ON period of other nodes to allow transmissions to be delivered eventually to their destinations -- but more crucially, to make use of NC, there should be periods where a node's ON period overlaps *at the same time* with the ON periods of two or more other nodes, and, (4) an explicit *acknowledgement* mechanism is needed to determine correctly decoded packets, in order to prevent redundant coding.

2.6.1.1 SYNCHRONIZED DC

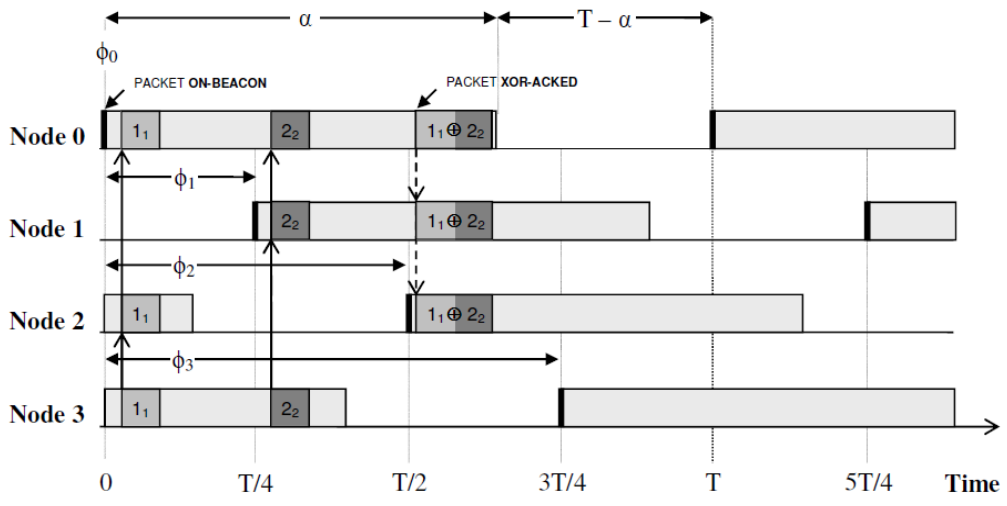


Figure 2.5. Synchronized DC for a 4-node network.

To satisfy the previously set requirements, and (3) in particular, we assume that there are N nodes i identified from (0 to N-1). These nodes have the same awake-sleep cycle length T (so $T_n = T$) consisting of the awake (ON) period α which is the same for all nodes (so $\alpha_n = \alpha$). With respect to the phases, ϕ_n, the nodes are assumed synchronized in such a way that the awake interval α of a node with id i (from 1 to N-1) is delayed with respect to the previous (i-1) node's α by T/N and node 0 acts as the time reference, i.e. $\phi_i = i(\frac{T}{N})$. For example, in Figure 2.5, in a clique of 4 nodes with ids from 0 to 3, the α of node 1 is starts after node 0's, node 2's after node 1's, and node 3's after node 2's, each phased T/4 time units apart. An additional requirement is that α is larger than T/N so that at any point in time there is always an awake node within the clique to receive incoming packets. In other words, $T \geq \alpha \geq T/N$. Notice also that the length α determines the extent of overlap with the ON period of other nodes. For the sake of terminology, if two nodes have overlapping ON periods, they are called *overlapped* nodes (otherwise they are called *non-overlapped* nodes).

A node needs to infer the packet reception status from its overlapped nodes, but given the DC timing outlined above, the process is greatly simplified. Assume that all nodes know the (trivial) DC timing as defined. If a node is ON, then it can be implied that they are ready to receive incoming packets (packet losses due to link errors are not accounted for). If the node is OFF then it is trivial to surmise that any packet sent to that node is going to be lost. In principle, an ON-BEACON packet can be broadcast from a node at the beginning of its α periods which will be received by the nodes overlapped with it at that point, thus allowing those other nodes to infer whether the node is ON or OFF. It is also assumed that this approach is not needed (or needed often) if this scheme could rely on perfectly synchronized clocks, in which case, the exchange of the ON-BEACON can happen at the initialization phase (during a first period of length T where all nodes are assumed to be ON continuously to exchange those ON-BEACONS) and all subsequent sleep periods are synchronized with respect to this first cycle.

Next, to demonstrate how the NC coding applies with respect to the synchronized schedule, consider Figure 2.5. Suppose that packet 1_1, in which the main digit represents the packet's receiver node (1 in this case) and the subscript digit (1 in this case) is the packet's serial number, is transmitted from node 3 to node 1. Knowing that node 1 is in its sleep period, nodes 0 and 2, which are in their awake periods, overhear this packet and consider it as a missed packet. Note that the overhearing is indicated by two non-dotted arrows from node 3 to nodes 0 and 2. Similarly, since packet 2_2 is transmitted from node 3 to node 2, which is currently sleeping, this missed packet is then overheard by nodes 0 and 1, which are awake at this moment. Note that the overhearing is indicated by two non-dotted arrows from node 3 to nodes 0 and 1. Hence, the two missed packets 1_1 and 2_2 can be XOR-ed into a single encoded packet and broadcast from node 0 to both nodes 1 and 2 when they are both awake so that node 1 with packet 2_2 can decode this encoded packet to regain packet 1_1 and node 2 with packet 1_1 can decode it to regain packet 2_2. Note that the broadcasting is indicated by two dotted arrows from node 0 to nodes 1 and 2. The reader may wonder why node 3 did not wait until node 1 (node 2) woke up and then transmitted packet 1_1 (packet 2_2) to the node instead of transmitting the packet right away. This is mainly aimed to create more opportunities for network coding to happen so that transmissions and therefore energy can be saved. Note that as described in [22], with the opportunistic network coding we can combine more than 2 packets into a coded packet and then more than one transmission can be saved by a single coded transmission. With the NC application, we are well aware of packet delay that may cause by the coding scheme. Since the delay is obvious, the early work only considers throughput and energy.

2.6.1.2 ACKNOWLEDGEMENT MECHANISM

A node that (having received a suitable NC packet) successfully decodes a previously missed packet, should reply with an ACK as soon as possible to prevent further coding attempts to include the (now recovered) packet. The same ACK is in fact of value not only to the sender of the NC packet but also to any other (overlapped) node that might have overheard the same missed packet and could attempt to include them in its NC packets in the future.

To be exact, the process of acknowledgements in the scheme works as follows: the sender node that transmitted an NC packet which combines some missed packets implicitly assumes that the receiving nodes *will* successfully decode the NC packet. Hence, immediately after the transmission, the sender removes the overheard packets that contributed to the NC packet from its own cache. Additionally, any other node that receives the NC packet *and* has overheard the missed packets that are contained in the NC packet, will also assume that the receiving nodes will correctly decode the packets they miss, and hence remove the corresponding overheard packets from their caches. Finally, for nodes that have overheard the missed packets but have not received this NC packet because either they were OFF or just failed to receive it (due to limited link reliability), they will get the chance to remove those packets from their caches when they hear the ACK packet from the destinations that successfully decoded missed packets. At each node, the ACK packet triggered from successful decoding is combined with the rest of the pending ACK packets of the same node and they are transmitted piggybacked on the next NC packet the node will get an opportunity to transmit. Note that because transmissions (including the piggybacked ACKs) are not guaranteed to be received, a final safety mechanism for the disposal of cached packets is a timeout, which, when it expires, drops the corresponding packet from the cache, if its successful reception has not been confirmed so far. With the above mechanisms, it is able to bound the number of cached packets, accepting that an occasional loss of ability to deliver a packet is considered acceptable if it is fairly rare. In the simulations, the packet delivery ratio captures the overall ability to deliver (directly or via NC packets) the combined impact of all the above mechanisms.

For example, in Figure 2.5, after node 0 has transmitted an NC packet combining the missed packets 1_1 and 2_2 destined for respective nodes 1 and 2, node 0 implicitly assumes that nodes 1 and 2 will successfully decode the NC packet. Therefore, node 0 removes those missed packets from its packet cache. In addition, nodes 1 and 2, which have received the NC packet and have overheard the missed packets, also remove those packets from their caches. For the other sleeping overlapped nodes of nodes 1 and 2, if there are any, an ACK packet which is incorporated in the pending ACK packets will become part of the header in the next NC packet from nodes 1 and 2, which will be discussed furthermore in the next section.

2.6.2 PROTOCOL ELEMENTS

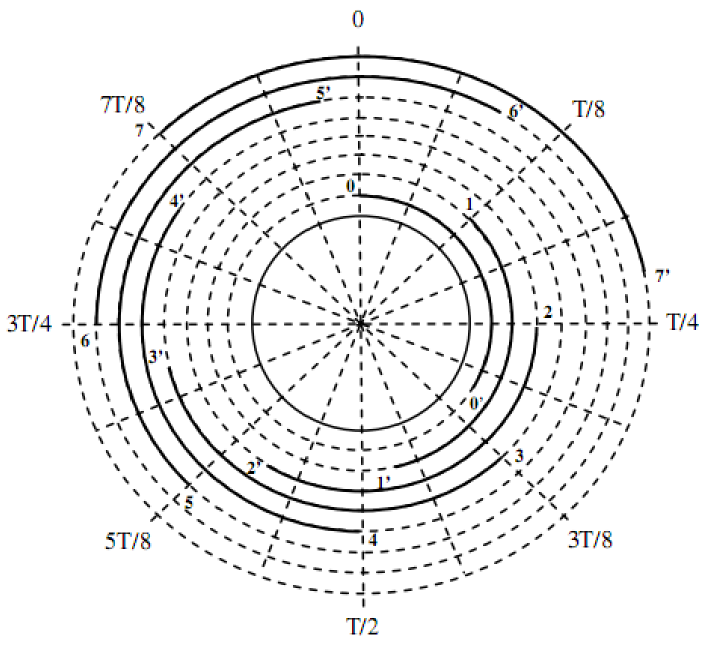
2.6.2.1 PACKET TYPES

There are three packet types, ON-BEACON, TRAF-DATA, and XOR-ACKED defined in the scheme. The purpose of the first one was briefly discussed in earlier subsections and given the assumption of perfect synchronization it is not needed beyond the first, initialization, phase when the phasing of the ON periods is established.

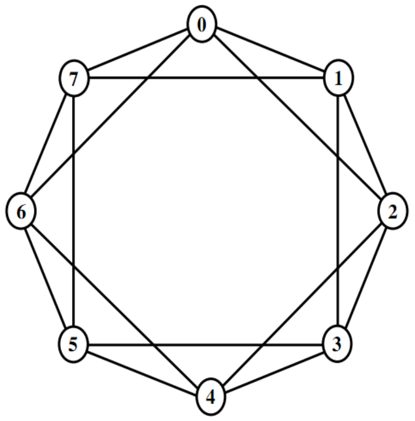
The TRAF-DATA are the "pure" data traffic packets, and they can be generated by any node in the network, externally from the clique or from any node in the clique. A node in possession of a TRAF-DATA packet views the packet as belonging into one of two possible (logically distinct) groups: either destined for a node that overlaps with the current node, or destined to a non-overlapped node. The first group can be delivered directly during the overlapped period. This group includes packets that have been received by a node in its role as a "representative" of the clique (because its destination is OFF), in which case the TRAF-DATA packets will be cached and effort is going to go into combining many of them into NC packets which, again, will be transmitted during the appropriate overlapped period (overlapped with more than one other ON periods). The second group of packets is transmitted from a node which has generated these packets according to some application-level traffic needs, or it is forwarded from a node which has received it during the node's ON periods. How to forward the second group of packets will be discussed in the next section.

The XOR-ACKED packets are the NC coded packets and they are transmitted by a node during its ON periods to forward (in a single transmission) multiple missed packets destined (as their eventual destination) for its overlapped nodes. As the name suggests, XOR-ACKED contains both NC content and acknowledgements. The NC part contains the sender addresses, the receiver addresses, and the serial numbers of the missed packets, which are XOR-ed into the XOR-ACKED packet's payload. The ACK part contains the sender addresses, the receiver addresses, and the serial numbers of the missed packets that the node has regained through decoding XOR-ACKED packets received from its overlapped nodes but has not acknowledged yet.

2.6.2.2 ROUTING ALGORITHM



(a)



(b)

Figure 2.6. Pictorial depiction of the DC timing and related routing graph (N=8).

Not all TRAF-DATA packets will be able to be immediately (or at all) coded into XOR-ACKED packets. Hence, a form of direct delivery is still necessary to bring those packets to a node that overlaps with the destination (at which point its inclusion in a NC packet *might* be possible), but this requires a rudimentary form of routing. The node determines the least-hop path to the packet's destination node and forwards the packet. Note that when the duty cycle α/T is greater than 1/2, each node overlaps with the others and thus there is no non-overlapped node in this situation. In other words, the routing algorithm is only needed in the scheme when its duty cycle α /T is less than or equal to 1/2.

The least hop path is defined as the least number of transmissions between nodes that overlap on the way to the destination. For illustration purposes, the duty cycling of a clique of eight nodes, identified from 0 to 7, with duty cycle α/T (less than 1/2) is considered in Figure 2.6.a. The solid line arcs, 00' 11' and so on, represent α and the dotted line arcs, 0'0 1'1 and so on, represent T- α (i.e., the sleep period). The overlaps of this schedule are captured by the graph in Figure 2.6.b, in which the vertices are the nodes and an edge exists if the two vertices/nodes it connects overlap. It is trivial to observe that for each pair of nodes there exist at least two paths (one in the clockwise and one in the counter-clockwise direction). Naturally, the one with the least number of hops will be chosen.

2.6.2.3 THE NC ALGORITHM

The purpose of the algorithm is to maximize the number of missed TRAF-DATA packets that can be delivered in a single transmission combined as an XOR-ACKED packet, subject to the constraint that each of its overlapped nodes has the right packets already in its possession to successfully decode this XOR-ACKED packet. The principle is that in order to encode n TRAF-DATA packets p_1, ..., p_n into a XOR-ACKED packet and transmit the packet to its n overlapped nodes o_1, ..., o_n, a node can XOR the n packets together only if each overlapped node o_i for which the missed packet p_i is destined has all $n - 1$ packets p_j for $i \neq j$.

Consider Figure 2.7 and suppose that a node maintains n "missed packet" caches MC_1, ..., MC_n to store the TRAF-DATA packets destined for its n respective overlapped nodes. Assume that each cache MC_i contains m_i missed packets. Additionally, the node maintains n overheard caches OC_1, ..., OC_n to store the overheard packets that its *n* respective overlapped nodes have also overheard. Also assume that each cache OC_i contains k_i overheard packets. These caches could be implemented as linked lists where *Tail* points to the oldest packet, which has been inserted first in the list, and *Head* points to the most recent packet which has been inserted last. Figure 2.8 provides the pseudo code of the coding algorithm which provides an XOR-ACKED packet given the current content of the packet caches at a node. It is an approximation heuristic because it is able to produce an acceptable solution rapidly but without searching all possible solutions, hence missing some coding opportunities. However, what is important and different from previous work is that the scheme does not consider just coding pairs of packets. Indeed, it can produce XOR-ACKED packets conveying *multiple* data packets.

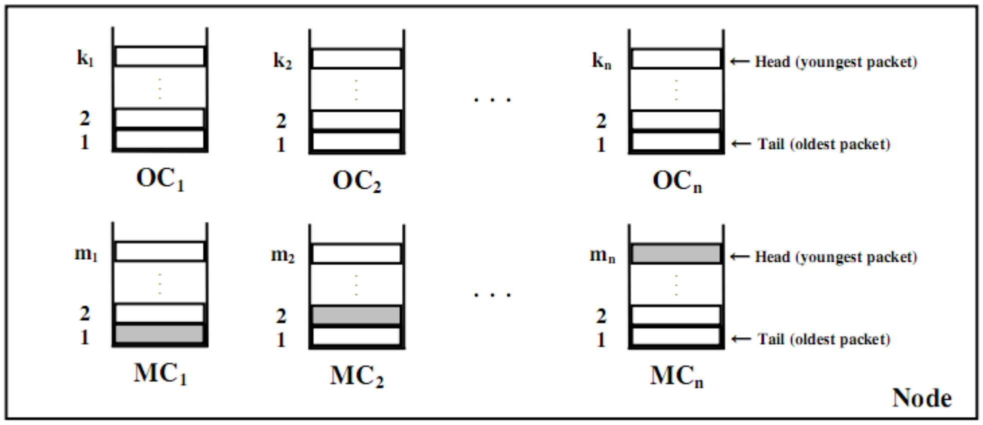


Figure 2.7. Structure of caches MCs and OCs at each node.

In Figure 2.8, *Maximum_Missed_Packets* returns an approximation to the maximal set of packets that can be coded together as a single XOR-ACKED packet and be decoded successfully by the nodes that will receive this transmission. *Temporary_Missed_Packets* is a work temporary variable holding the progressively expanding set of missed packets that can be coded together. *Missed_Packet* (and the corresponding index, *r*) indicates the missed packets in the MC_i cache. *Overlapped_Node* (and the corresponding index, *i*) indicates the nodes with which the current node overlaps in the duty-cycling schedule. There are four nested **for** loops in the algorithm. The first and third loops scan across the overlapped nodes. The second and fourth loops scan across missed packets in a MC cache. The first and second loops are also used to select the first missed packet to enter in *Temporary_Missed_Packets*, while the third and fourth loops are used to select and combine additional missed packets in *Temporary_Missed_Packets*. To decide whether more missed packets can be included or not, it needs to check two conditions. First, the next missed packet's receiver node must have overheard all missed packets of *Temporary_Missed_Packets*, so that it could be able to decode the new coded packet (this condition is checked by the second **if**). Second, the next missed packet must have been overheard by every receiver node which has its missed packet in *Temporary_Missed_Packets* (this condition is checked by the third **if**). The fourth **if** ensures that *Maximum_Missed_Packets* contains the maximum number of packets found possible to combine so far.

We emphasize that this is not an exhaustive search algorithm because whenever the next missed packet is chosen (which means it satisfies the two conditions) the fourth **for** loop stops searching any further in the current MC, so the next combination that it needs to try out starts with the next available MC. For example, in Figure 2.7, when the first missed packet 1 is chosen from MC_1 and added into *Temporary_Missed_Packets*, MC_2 is searched to find the next missed packet for *Temporary_Missed_Packets*. Suppose that the missed packet 2, which is satisfied with the two above conditions, is chosen from MC_2 and added into *Temporary_Missed_Packets*, then any remaining missed packets in MC_2 are ignored. In this example, an encoded packet with the shaded missed packets, 1, 2, ..., and m_n are returned by *Maximum_Missed_Packets*.

Even though it is not exhaustive, this algorithm is (at the cost of a slight computation overhead) better than limiting the XOR-ACKED packets to be composed solely out of pairs of data packets. This is in contrast with previous works on practical NC schemes. It is recognized though that the extra gain by considering multiple (>2) packet combinations may not be useful under some scenarios. For example, under low traffic loads, or when multiple missed packets are combined into a transmission it could be that the remaining missed packets cannot be combined (even in pairs) any more.

Heuristic Coding Algorithm - Network Coding with Heuristic Search

```
Maximum_ Missed_Packets = { }
for Overlapped_Node i = 1 to N   do
    Temporary_Missed_Packets = { }
    if   i^th Overlapped_Node is awake  and  $MC_i \neq \phi$  then
        for Missed_Packet r = 1 to $m_i$  do
            Temporary_ Missed_Packets  =  Temporary_ Missed_Packets $\cup$
                                          { r^th Missed_Packet }
            for Overlapped_Node j = 1 to N   do
                if   $j \neq i$  and  j^th Overlapped_Node is awake  and
                     $MC_j \neq \phi$  and  Temporary_ Missed_Packets $\subset$ $OC_j$   then
                     for  Missed_Packet s = 1 to $m_j$  do
                         if   $\forall$ r^th Missed_Packet $\in$ Temporary_ Missed_Packets :
                              s^th Missed_Packet $\in$  $OC_r$   then
                              Temporary_ Missed_Packets  =  Temporary_ Missed_Packets $\cup$
                                                            { s^th Missed_Packet }
                              break
                         end if
                     end for
                end if
            end for
            if   $\Sigma$ packets of Temporary_ Missed_Packets >
                 $\Sigma$ packets of Maximum_ Missed_Packets   then
                 Maximum_ Missed_Packets  =  Temporary_ Missed_Packets
            end if
            Temporary_ Missed_Packets  =  { }
        end for
    end if
end for
return   Maximum_ Missed_Packets
```

Figure 2.8. Pseudo-code of the NC algorithm executed at each node.

The decoding algorithm is obvious: check whether the node's id is included in the packet's header and check to see if the node's overheard cache OC contains the remaining n–1 missed packets included in the XOR-ACKED packet. If yes, decode the packet.

2.6.2.4 PRIORITIZING TRANSMISSIONS

XOR-ACKED and TRAF-DATA are scheduled to be transmitted during α periods. TRAF-DATA are furthermore distinguished to those heading to overlapped nodes and those heading for non-overlapped nodes. TRAF-DATA to overlapped nodes, are given the lowest priority, and scheduled to be transmitted from a node during its α periods whenever there are no packets waiting in any of the other two transmission queues (XOR-ACKED, and TRAF-DATA to non-overlapped nodes). TRAF-DATA to non-overlapped nodes are scheduled to be transmitted from a node during its two intervals at the two ends of the node's α period (and they are subjected to the routing algorithm).

Generally, XOR-ACKED packets are given higher priority whenever there are opportunities for NC to take place (i.e., during periods when one or more nodes' ON periods overlap with the current node). Note that at the beginning of every interval T/N of the α period, one more node of the node's overlapped nodes wakes up and another node goes to sleep. Therefore, XOR-ACKED packets can be transmitted at these points to make use of the new opportunities.

However in order to compromise and not starve the TRAF-DATA that are heading to non-overlapped nodes, the following policy is implemented: if the duty cycle is less than or equal to 1/2, one XOR-ACKED packet is broadcast at the beginning of every of these T/N intervals except the two end intervals of the node's α to allow for TRAF-DATA packets to non-overlapped nodes. However, if the duty cycle is greater than 1/2, XOR-ACKED are allowed to be scheduled even at these end intervals because there cannot exist TRAF-DATA to non-overlapped nodes (all nodes overlap at some point during their ON periods).

2.6.3 SIMULATION RESULTS

The proposed scheme is simulated using the SMURPH/SIDE tool [45]. Only a single clique of N nodes is considered and all data traffic is produced by the nodes is destined to randomly uniformly distributed destinations across the remaining N-1 nodes. The message inter-arrival time is exponentially distributed with the mean from 0.15 to 100 secs, and the message (payload) length is uniformly distributed between 256 and 16384 bits. The period, T, is 32 secs and the channel's transmission rate, C, is 115000 bits/s. All results are presented with their corresponding 95% confidence intervals.

To compute the energy consumption, it is assumed that a node consumes energy only during its α period. The energy consumption $E_{node\text{-}T}$ of a node during a period T is calculated by the following formula:

$$E_{node\text{-}T} = E_{tx\text{-}Ton} + E_{rx\text{-}Ton} + E_{idle\text{-}Ton} \tag{2.1}$$

where, $E_{tx\text{-}Ton}$ (or $E_{rx\text{-}Ton}$) is its energy spent to transmit (or receive) all packets including XOR-ACKED and TRAF-DATA during its α period, and $E_{idle\text{-}Ton}$ is its energy spent when the node is in the idle listening state during its α period. We note that though the nodes are synchronized they are still in the idle listening mode. This is because the traffic load is unknown (due to its randomly and exponentially distributed time between message arrivals) and there is no scheduling scheme for packet transmissions. Also note that the received TRAF-DATA packets include the packets which are overheard by the node. Due to the broadcast nature of wireless transmissions, though the TRAF-DATA packets are from unicast transmissions, they are still able overheard by nearby nodes that are within the transmission coverage of the sender nodes. The received XOR-ACKED packets also include the packets which are not destined for the node. To save more energy, instead of receiving a whole packet of this type, the node just reads part of the header on the fly until it finds that the packet is not useful for itself and then discards it. When the duty cycle is less than or equal to 1/2, the forwarded

TRAF-DATA packets routed to non-overlapped nodes also show up in the energy cost.

If the ratio of power consumption is represented by *x*:*y*:*z* corresponding to *Transmission*:*Reception*:*Idle* then the definition of the energy consumption for a node becomes:

$$E_{node\text{-}T} = x \cdot T_{tx\text{-}Ton} + y \cdot T_{rx\text{-}Ton} + z \cdot (\alpha - (T_{tx\text{-}Ton} + T_{rx\text{-}Ton})) \qquad (2.2)$$

where, $T_{tx\text{-}Ton}$ (or $T_{rx\text{-}Ton}$) is the time spent to transmit (or receive) all packets during the ON period α of the node. $T_{tx\text{-}Ton}$ (or $T_{rx\text{-}Ton}$) is calculated based on the number of all transmitted (or received) packets, the length in bits of a packet, and the data rate of the channel in bits per second. Note that when the duty cycle is equal to 1/1 (all nodes continuously ON) period α is equal to T and no XOR-ACKED are needed then.

2.6.3.1 NETWORK THROUGHPUT

Let's start by studying the impact that NC can have on DC networks. We anticipate that, despite NC having the ability to improve the throughput, this this is limited by two factors: (a) the need to forward, i.e., route packets because only a specific subset of nodes overlap in their ON periods with each other node, and (b) the ability to have two or more receivers overlapping (ON) such that NC packets will be of value. Clearly, (a) is more pressing when the traffic load is high, the number of nodes, N, is large, and the duty cycle (α/T) is small. The last two conditions (large N and short duty cycle) force the traversal of more intermediate hops, thus amplifying the offered load to the system. As far as (b) is concerned, NC is expected to be useful when there are long overlaps with more than two other nodes, which means either a large duty cycle (α/T) and/or many nodes (higher N).

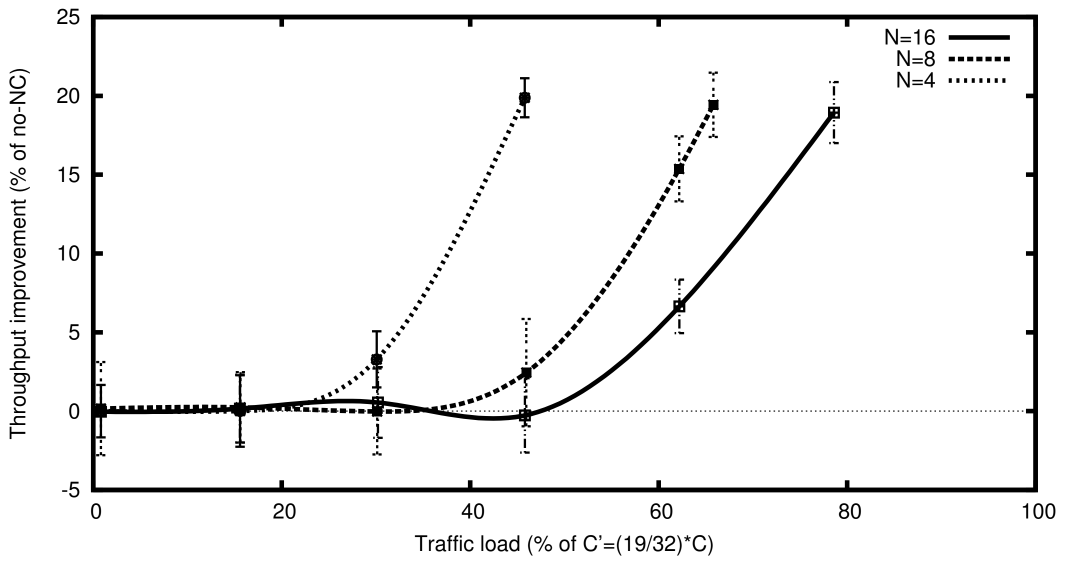


Figure 2.9. Throughput gain for 19/32 duty cycle.

As a representative plot of the throughput performance, consider Figure 2.9 where a duty cycle of 19/32 (19 seconds awake, 13 seconds asleep) is considered. We use a normalizing factor for expressing the improvement brought, as a percentage, by NC compared to no-NC use. The x axis is annotated in terms of traffic load compared not against C, but against a “reduced” C by a factor that depends on the duty cycle, C’ = (α /T) C. Positive (negative) numbers indicate an improvement (deterioration) of the throughput versus duty cycling without the use of NC. It can be seen that at loads lower than a threshold (for these parameters approximately below 25% of C') the NC scheme provides no throughput improvement. Quite the contrary, the construction of XOR-ACKED packets and their transmission (which, incidentally, as outlined before, is given higher priority) is counter-productive.

The advantage of NC is evident at high traffic loads, leading to a throughput improvement up to 20% above what is possible with duty cycling but without NC. The relatively large (>50%) duty cycle results in no need of forwarding while at the same time there are plenty of opportunities to transmit XOR-ACKED packets. Note that the lines are terminated early because the system reaches saturation noticeably lower than at 100% of C'. However, the same saturation points are reached in the case where duty cycling is used without NC, so there are compelling reasons to use NC at high loads for its ability to sustain higher throughput when nearing the saturation point. A final note is the larger the number of nodes, N, the larger the overlapping ON periods, and hence the more drastic the impact of NC.

The shape of Figure 2.9 is typical of what is seen at duty cycle values longer than 50%, but not too long. That is, once it reaches very long duty cycles, e.g., 28/32, the improvement in throughput remains around 0%, i.e., no real difference versus not using NC, which is to be expected because a smaller fraction of packets are missed by virtue of the receiving node being OFF for shorter time intervals. At the other extreme, when operating with shorter duty cycles, e.g., 9/32, and a large number of nodes, e.g., 16, saturation is possible due to the need to perform forwarding of the packets. The addition of NC has a similar impact as before (in the best case it is found close to 18% and at 0% in the worst case) but the total throughput, even after the improvement provided by NC remains small in absolute numbers to be of any practical interest. Nevertheless, what was really sought to confirm in this subsection was that there was no significantly detrimental effect on the throughput (given that capacity was trimmed away by duty cycling and adding overhead), and in fact it was able to notice an improvement in certain cases.

2.6.3.2 ENERGY SAVINGS

Assume a network consisting of a single clique of sixteen nodes (N=16) and let the relative cost of transmission, over reception, over idling be $x{:}y{:}z = 1.8{:}1.2{:}1.0$. Suppose that the duty cycle is being changed and the NC scheme described earlier is applied. The same throughput may be achievable by different duty cycles, but each duty cycle is associated with different energy costs for transmitting forwarded packets and/or XOR-ACKED packets as well as, of course, the ability to switch the transceiver OFF and not incur any energy cost during those periods.

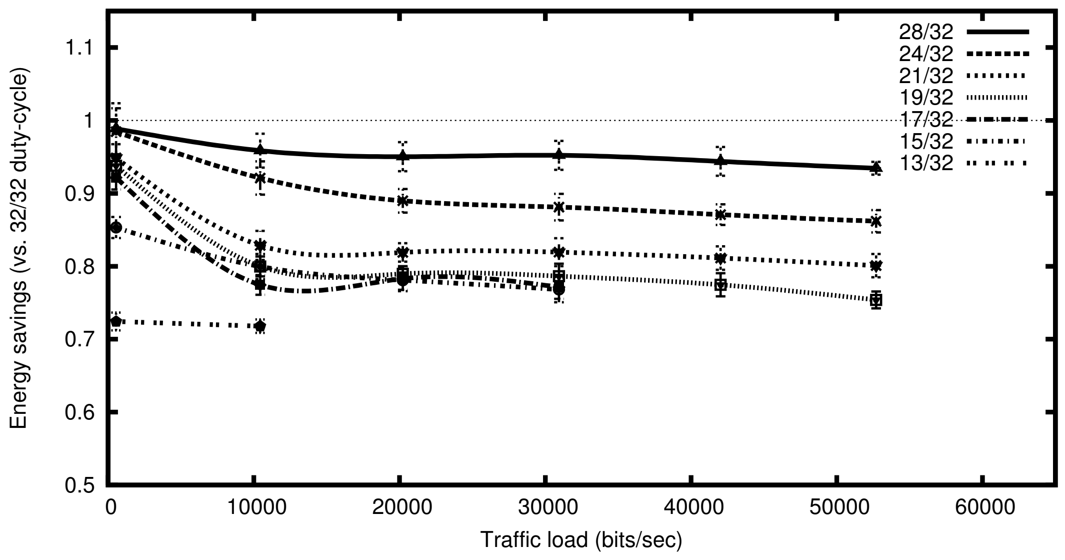


Figure 2.10. Duty cycle energy savings vs. 32/32 (continuously ON)

To express the fact that the same throughput is achievable using different duty cycles, Figure 2.10 plots the throughput (x axis) and the energy saved versus keeping all nodes continuously ON (a 32/32 duty cycle). A value less than 1.0 on the y axis means a corresponding savings in energy. Each line represents a different duty cycle. However note that different duty cycles result in the network saturating at different loads. This is the reason why not all lines of Figure 2.10 extend all the way from the lowest throughput (10,000 bits/sec) to the highest (60,000 bits/sec). For example, duty cycles 17/32 and 15/32 cannot reach throughputs above 30,000 bits/sec without saturating. A squinting reader will discover that there is also a point for a duty cycle of 13/32, appearing (for 10,000 bits/sec throughput) on the y axis just above the 0.7 tick mark. Notice that in all simulations a simplistic MAC protocol is used to avoid yet one more complex interaction between scheduling and MAC. The MAC is a listen-before-transmit with a fixed back-off window which (for C=115000 bits/sec) saturates close to 60,000 bits/sec for small networks, e.g., N=16. Hence, the range of throughputs in Figure 2.10 spans up to the intrinsic saturation point of the network even if no duty cycling was employed.

From the results in Figure 2.10, it can be clearly seen that, for a given desirable throughput, there exists a smallest duty cycle that can sustain the required throughput and spend the least energy. For example, at 20,000 bits/sec throughput a duty cycle of 15/32 would be sufficient. The energy savings also exhibit a trend of diminishing returns, as can be seen by the co-incidence of the lines for duty cycles 19/32 and lower. Unfortunately, once it crosses into really small duty cycles (13/32 and lower) performance is drastically reduced because of the increased load caused by the forwarding.

2.7 CHAPTER CONCLUSIONS

In this chapter we attempted to summarize the relevant previous work by other researchers as well as some initial results on combining DC and NC. The results from [42] show that we can combine DC and NC to save energy and improve throughput as long as we have opportunities to apply NC, e.g., there is an overlap among duty-cycles of three sequential neighbour nodes. The early results demonstrated that small duty cycles are, inherently, limiting as they do not allow nodes to convey enough traffic. Towards relieving this problem, we will have to expand our scope to allow for NC to be performed at intermediate nodes in the routing paths and to ensure having sufficient traffic in order to perform NC, possibly at the cost of delaying traffic. Both these facets will be addressed in Chapter 4.

Figure 2.10 shows that significant savings of 25% versus continuously-ON operation could be achieved which makes us hopeful that this approach has potential. However, even though the early simulations provide convincing indications, the model used to demonstrate the strength of DC+NC was simplified with respect to the topology and the particular DC pattern. Hence, from Chapter 3 onwards, we will expand our study to arbitrary topologies and DC patterns. We will do so by first studying DC in isolation in Chapter 3, and then expanding to DC+NC in Chapter 4. In addition, in the early results our approach to build a

MAC protocol, apart from being complicated, raised issues of prioritizing transmissions and accounting for backlog of the nodes. Both these are essentially scheduling issues. Hence, rather than embark on the minutae of building a MAC protocol, we will, starting with Chapter 3, build in a centralized manner a complete schedule in order to have a high performance benchmark against which future MAC protocols could be compared.

CHAPTER 3
A TDMA SCHEDULING ALGORITHM FOR DUTY-CYCLED WIRELESS SENSOR NETWORKS

3.1 PROBLEM FORMULATION

In this chapter we formulate and compute max-min rates for a DC network. The results are subsequently used to set performance benchmark for the rates to be achieved by each flow during a slotted STDMA schedule construction. The STDMA schedule construction operates by performing a simulation of the system up to the point where a periodic steady state behavior is detected, at which point a single period of the scheduling decisions is excised and used as a schedule template. Simulation results and techniques to perform minor adjustments in the excised schedule to ensure e.g., the flow balance property, are also presented.

3.1.1 MAX-MIN FAIRNESS FORMULATION

The following shows the max-min fairness formulation, which is used by either the watter filling (WF) algorihm (in section 3.1.2) or the max-min programming (MP) algorithm (in section 3.1.3) intended to compute the numerical rates of all the flows $f \in \mathcal{F}$. The formulation is for DC WSNs, which is an extension of the work [46] where it was originally introduced for wireless mesh networks. Let $\langle \boldsymbol{r_f} \rangle = (\langle r_f \rangle_1, \dots, \langle r_f \rangle_m)$ denote a version of vector $\boldsymbol{r_f} = (r_{f_1}, \dots, r_{f_m}) \in \mathbb{R}^m$ ordered in the non-decreasing order, i.e., for some permutation φ on the set $\{1, \dots, m\}$ it holds that $\langle r_f \rangle_j = r_{f_{\varphi(j)}}$for $j = 1, \dots, m$ and $\langle r_f \rangle_1 \leq \ \dots \leq \langle r_f \rangle_m$. The objective (3.1) denotes the lexicographic maximization of the sorted outcome vector $\boldsymbol{r_f}$ over $\mathcal{R}$, as defined in [49, 50, 51, 52]. Since the physical topology is arbitrary, at any stage (using the definition of "stage" appearing in the notation & definitions of Chapter 1) a different (but known/computed) number of maximal cliques exist, representing the contention graphs of the network. Constraint (3.2) specifies the flow conservation constraints which require that at any node n the rate of each flow summed over all stages is conserved.

$$lexmax_{\ r_f \in \mathcal{R}} \langle \boldsymbol{r_f} \rangle \tag{3.1}$$

Subject to

$$\sum_{k\in\mathcal{K};\, o(l)=n} r_f^{(k)}(l) - \sum_{k\in\mathcal{K};\, d(l)=n} r_f^{(k)}(l) = \begin{cases} r_f, & n = f_s \\ -r_f, & n = f_d \\ 0, & else \end{cases} \quad \forall n \in \mathcal{N}, f \in \mathcal{F} \tag{3.2}$$

$$\sum_{f\in\mathcal{F};\, l\in\mathcal{L};\, r_f^{(k)}(l)\in q} r_f^{(k)}(l) \leq\ r.\mathcal{T}^{(k)} \qquad \forall k \in \mathcal{K}, \forall q \in \mathcal{Q} \tag{3.3}$$

$$r_f \geq 0,\ r_f^{(k)}(l) \geq\ 0 \qquad \forall f \in \mathcal{F}, l \in \mathcal{L}, k \in \mathcal{K} \tag{3.4}$$

Constraint (3.2) captures the key difference of this max-min formulation from general forms of flow rate formulations, in that it confines the flow balance to be met within a single sequence of stages (which, as described in Chapter 1, will be periodically repeating). That is, the flow conservation is enforced over a short time frame and not over an unbounded time frame. Constraint (3.3) specifies, for each stage, the link capacity restrictions that apply to contending sub-flows which are active in that stage. The rate r = 1/T (transmissions/slot) is the maximum rate in a slot and $\mathcal{T}^{(k)}$ is the length of stage k in number of slots. Note that constraint (3.3) considers each maximal clique ($q \in \mathcal{Q}$) of the contention graph, as defined in each stage k. Those cliques are, in principle, different in each stage due to the time varying nature of the underlying graph.

3.1.2 COMPUTATION OF MAX-MIN FAIR RATES BY THE WATER FILLING ALGORITHM

Figure 3.1 shows the $LP(r_f)$ formulation, which is used by the watter filling (WF) algorihm intended to compute the numerical rates of all the flows $f \in \mathcal{F}$. The formulation has the same flow conservation (3.6) and link capacity constraints (3.7) as those in the max-min fairness formulation in section 3.1.1. The other constraints are just used by the WF algorithm in Figure 3.2 and LP solver to compute the max-min fair rates. In the formulation, the objective (3.5) denotes the maximization of the total numerical rates of all the flows. Constraint (3.8) shows the numerical rates are changed at each time of submitting $LP(r_f)$ to the WF algorithm.

$$max \sum_{f \in \mathcal{F}} r_f \qquad (3.5)$$

Subject to

$$\sum_{k \in \mathcal{K};\, o(l)=n} r_f^{(k)}(l) - \sum_{k \in \mathcal{K};\, d(l)=n} r_f^{(k)}(l) = \begin{cases} r_f, & n = f_s \\ -r_f, & n = f_d \\ 0, & else \end{cases} \quad \forall n \in \mathcal{N}, f \in \mathcal{F} \qquad (3.6)$$

$$\sum_{f \in \mathcal{F};\, l \in \mathcal{L};\, r_f^{(k)}(l) \in q} r_f^{(k)}(l) \leq r.\mathcal{T}^{(k)} \qquad \forall k \in \mathcal{K}, \forall q \in \mathcal{Q} \qquad (3.7)$$

$$r_f = R_f \qquad S_f = FIXED, \forall f \in \mathcal{F} \qquad (3.8)$$

$$r_f \geq 0,\ r_f^{(k)}(l) \geq 0 \qquad \forall f \in \mathcal{F}, l \in \mathcal{L}, k \in \mathcal{K} \qquad (3.9)$$

Figure 3.1. Formulation $LP(r_f)$

The WF algorithm in Figure 3.2 orignates from the standard water-filling algorithm as described by Bertsekas and Gallager [47] for wireline networks but is extended to be applicable to duty-cycled wireless networks. It is associated with formulation $LP(r_f)$ to compute the max-min rates for multi-hop flows in a DC-WSN. The algorithm works like the water-filling mechanism, in which all the flow demands are met equally with the tentative rates, which are increased gradually with a tiny amount of rate. The process is performed repeatedly until the least demands are fully met first, or determined, then the higher demands next, and the highest demands last.

```
Input: $\langle \phi_n, \alpha_n, T_n \rangle$ with $n \in \mathcal{N}$; $\mathcal{T}^{(k)}$ with $k \in \mathcal{K}$; $\mathcal{L}$; $\mathcal{F}$; $Q$;
        Formulation $LP(r_f)$ with $f \in \mathcal{F}$;
Output: Values $R_f$ of numerical max-min rates $r_f$ of flows $f \in \mathcal{F}$;
Begin
01.  Set $S_f = UNFXD$ and $R_f = 0$ for all the flows $f \in \mathcal{F}$;
02.  Repeat
03.      Select flow $f$ with $S_f = UNFXD$ in a round robin manner;
04.      Increase $R_f$ by a minuscule value $\varepsilon$, i.e., $R_f = R_f + \varepsilon$;
05.      Submit $LP(r_f)$ to LP solver;
06.      If LP solver indicates $LP(r_f)$ is infeasible Then
07.          Restore $R_f$ to the previous value, i.e., $R_f = R_f - \varepsilon$;
08.          Change $S_f = FIXED$;
09.      End If
10.  Until $S_f = FIXED$ for all the flows $f \in \mathcal{F}$;
End
```

Figure 3.2. The WF algorithm

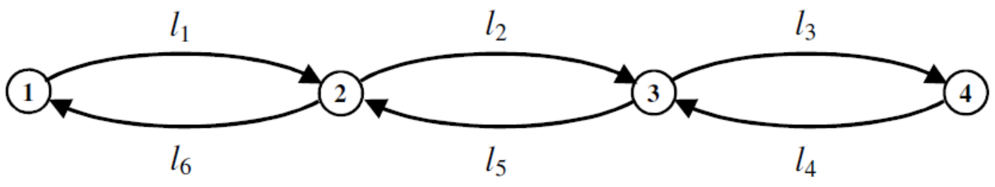
In step 1, states S_f are initialized to UNFIXED to indicate all the respective tentative rates R_f are undetermined. Steps from 2 to 10 are repeatedly performed until all states S_f are FIXED, i.e., determined, at which all the max-min rates r_f are figured out. In steps 3 and 4, a flow is chosen in a round-robin manner and then the R_f rate is increased by a minuscule value ε, which determines the accuracy of the algorithm. In step 5, formulation $LP(r_f)$ is submitted to LP solver. In steps from 6 to 9, the return value from LP solver indicates formulation $LP(r_f)$ is infeasible by the recently-increased rate R_f of flow f. Hence, at least one of the active sub-flows of the flow is bottlenecked across all the stages $\mathcal{T}^{(k)}$, i.e., the sub-flow's rate cannot increase anymore. This causes all the maximal contention cliques constructed by the bottlenecked sub-flow to become bottlenecked, i.e., the rates of sub-flows constructing the cliques cannot further increase, otherwise the sub-flow's rate could have further increased. Hence, in steps 7 and 8 the rate has to be restored to the previous value, i.e., $R_f - \varepsilon$, and its state S_f is set to the determined state, i.e., $FIXED$. In other words, the numerical rate r_f of flow f has been determined.

Suppose there are m flows in $\mathcal{F}$, i.e., $f_1, \dots, f_m \in \mathcal{F}$. We define $\boldsymbol{r_f}$, (in bold), as a vector of flow rates, i.e., $\boldsymbol{r_f} = (r_{f_1}, \dots, r_{f_m})$. Hence, we have $\boldsymbol{r_f} \in \mathcal{R}$, which is a set of all feasible flow rates, i.e., satisfying constraints (3.6) to (3.9), and $\mathcal{R} \subseteq \mathbb{R}^m$, which is the set of all real m-vectors.

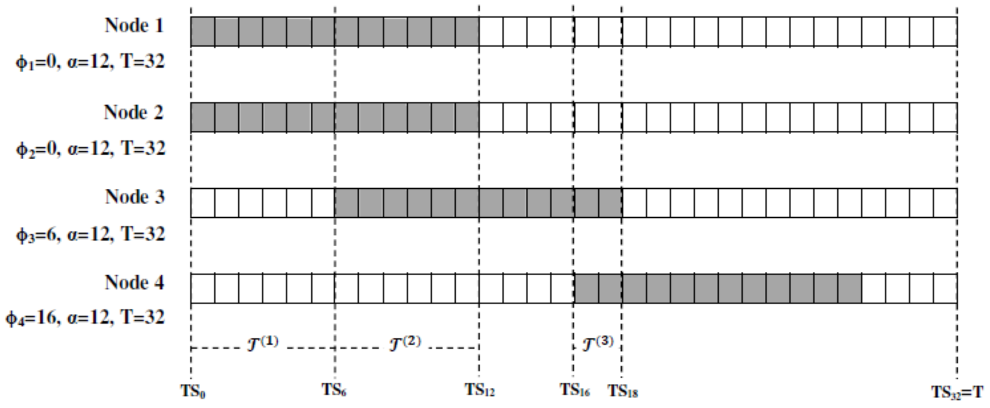
Property 1. *For any allocation vector $\boldsymbol{r_f} \in \mathcal{R}$ and for any flow f_i such that $r_{f_i} > r^0_{f_i}$ there exists a flow f_j such that such that $r_{f_j} < r^0_{f_j} \leq r^0_{f_i}$.*

Proof 1. We will prove that the feasible allocation vector $\boldsymbol{r^0_f} = (r^0_{f_1}, \dots, r^0_{f_m}) \in \mathcal{R}$ constructed from the WF algorithm is max-min fair by showing that $\boldsymbol{r^0_f}$ fulfills Property 1.

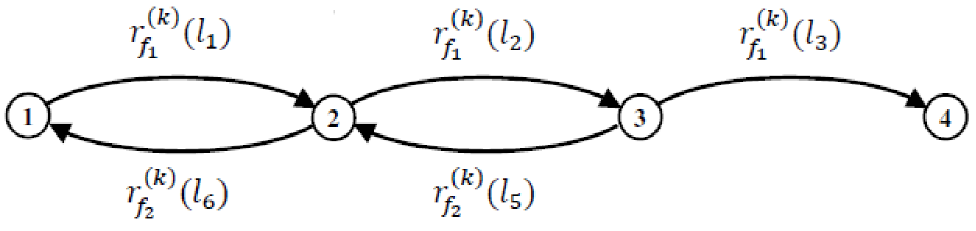
Example 3.1. To illustrate the proof, we consider a simple duty-cycled network with four nodes $n \in \mathcal{N} = \{1, 2, 3, 4\}$, six directed wireless links in the link set $\mathcal{L} = \{l_1, l_2, l_3, l_4, l_5, l_6\}$ shown in Figure 3.3.a.1, and two opposite flows $f_1, f_2 \in \mathcal{F}$ traversing, respectively, the paths $\{l_1, l_2, l_3\}$ and $\{l_5, l_6\}$ indicated in Figure 3.3.b. Note that there is a subtle distinction regarding the link topology between Figure 2.5 and Figure 3.3.a.1. The former has a circular topology while the latter has a linear topology although both networks have a single clique for their contention graph of sub-flows. In other words, nodes in Figure 2.5 can directly communicate with each other while those in Figure 3.3.a.1 can only directly communicate with their neighbors along the linear topology. The DC-configurations, i.e., $\langle \phi_n, \alpha_n, T_n \rangle$, of the nodes is as shown in Figure 3.3.a.2, in which period T is divided into three stages $\mathcal{T}^{(k)}$, in units of slots, with $k \in \mathcal{K} = \{1,2,3\}$, such that in each of these stages the ON/OFF state of the nodes is unchanged and there exists at least an active link (both its ends are ON). Note that the further detailed definition of a stage can be refered to Example 1.1. In Figure 3.3.b, during each stage k, flow f_1 is represented by three sub-flow components $r_{f_1}^{(k)}(l_1)$, $r_{f_1}^{(k)}(l_2)$, and $r_{f_1}^{(k)}(l_3)$, and flow f_2 is represented by two sub-flow components $r_{f_2}^{(k)}(l_5)$ and $r_{f_2}^{(k)}(l_6)$. Hence, the maximal cliques of the sub-flow contention graph which is made up from all sub-flows competing in each stage $\mathcal{T}^{(1)}$ (from slots TS_0 to TS_6), $\mathcal{T}^{(2)}$ (from slots TS_6 to TS_{12}), and $\mathcal{T}^{(3)}$ (from slots TS_{16} to TS_{18}) are shown in Figures 3.3.c.1, 3.3.c.2, and 3.3.c.3, respectively.



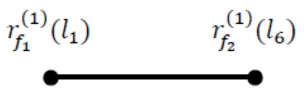
(a.1) Link topology



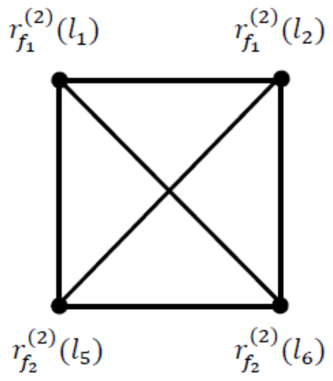
(a.2) States ON/OFF of nodes (ON slots are gray)

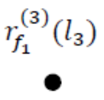


(b) Multi-hop flows



(c.1) CG in $\mathcal{T}^{(1)}$





(c.3) CG in $\mathcal{T}^{(3)}$

(c.2) CG in $\mathcal{T}^{(2)}$

Figure 3.3. An illustrative example for the WF and MP algorithms' proofs (CG = Contention Graph)

Suppose we have a feasible allocation vector $\boldsymbol{r_f} \in \mathcal{R}$, and a flow f_i, e.g., f_2 in Figure 3.3, such that $r_{f_i} > r_{f_i}^0$. From the WF algorithm's pseudo-code, we have the following observations:

(1) Flow f_i must consist of at least an active bottlenecked sub-flow, which participates in a maximal bottlenecked clique in a stage, e.g., f_2 has bottlenecked sub-flow $r_{f_2}^{(2)}(l_5)$ participates maximal bottlenecked clique in Figure 3.3.c.2. In the other stages, the sub-flow must also be bottlenecked in respective maximal bottlenecked cliques when it is active, e.g., f_2 also has bottlenecked sub-flow $r_{f_2}^{(1)}(l_6)$ participates maximal bottlenecked clique in Figure 3.3.c.1. Otherwise, the rate could have been further increased;

(2) Because the maximal cliques participated by the bottlenecked sub-flow of flow f_i are bottlenecked, the other sub-flows that construct the cliques cannot increase their rates, i.e., they are also bottlenecked, e.g., f_1 has bottlenecked sub-flows $r_{f_1}^{(1)}(l_1)$ and $r_{f_1}^{(2)}(l_2)$ participates maximal bottlenecked clique in Figures 3.3.c.1 and 3.3.c.2;

(3) The maximal bottlenecked cliques and respective flow rates are sequently found in an increasing order of flow rates, e.g., smallest rate r_{f_1} of flow f_1 is found first with maximal bottlenecked clique in Figure 3.3.c.3 and larger rate r_{f_2} of flow f_2 is found next with maximal bottlenecked cliques in Figures 3.3.c.1 and 3.3.c.2;

(4) Hence, we can always find a flow f_j, e.g., flow f_1, whose rate is determined earlier than that of flow f_i (i.e., $r_{f_j}^0 \leq r_{f_i}^0$), e.g., flow f_2 with $r_{f_1}^0 \leq r_{f_2}^0$, by the algorithm and whose bottlenecked sub-flow shares maximal bottlenecked cliques with that of flow f_i when the flow f_i's rate is found, e.g., flows f_1 and f_2 share maximal bottlenecked cliques in Figures 3.3.c.1 and 3.3.c.2 when rate r_{f_2} of flow f_2 is found.

Therefore, when the rate r_{f_i} of flow f_i is increased ($> r_{f_i}^0$), the bottlenecked sub-flow's rate of the flow is also increased. Since the maximal clique shared between two bottlenecked sub-flows of the two respective flows f_i and f_j is bottlenecked, the sub-flow's rate of flow f_j must be decreased. This causes the rate r_{f_j} of flow f_j to be decreased, i.e., $< r_{f_j}^0$. In other words, we have proved that there exists a flow f_j such that such that $r_{f_j} < r_{f_j}^0 \leq r_{f_i}^0$. For example, when the rate r_{f_2} of flow f_2 is increased ($> r_{f_2}^0$), we can always find a flow f_1 such that such that $r_{f_1} < r_{f_1}^0 \leq r_{f_2}^0$.

3.1.3 COMPUTATION OF MAX-MIN FAIR RATES BY THE MAX-MIN PROGRAMMING ALGORITHM

It is obvious that the time complexity of the WF algorithm depends on the ε value. In order to achieve acceptable accuracy, this value needs to be small enough, which can slow down the computation process. To accelerate the computation and still ensure the accuracy, the WF algorithm is generalized to the MP algorithm based on the framework proposed by of Radunovic and Le Boudec [48]. The MP algorithm no longer uses the ε value in figuring out the numberical rate of a flow and hence the computational time of numerical per-flow rates is significantly less than that of the WF algorithm and mostly dependent on the number of flows in the most practical cases, which will be explained later in the next paragraph.

```
Input: $\langle \phi_n, \alpha_n, T_n \rangle$ with $n \in \mathcal{N}$; $\mathcal{T}^{(k)}$ with $k \in \mathcal{K}$; $\mathcal{L}$; $\mathcal{F}$; $\mathcal{Q}$;
        Formulations $LP\ (f)$ and $LP\ (f, R_{min})$ with $f \in \mathcal{F}$;
Output: Values $R_f$ of numerical max-min rates $r_f$ of flows $f \in \mathcal{F}$;
Begin
01.   Set $S_f = UNFXD$ for all the flows $f \in \mathcal{F}$;
02.   Repeat
```

```
03.         Select flow $f \in \mathcal{F}$ with $S_f = UNFXD$;
04.         Submit $LP\ (f)$ to LP solver;
05.         Store rate $R_{min}$ of flow $f$ returned from LP solver;
06.         **For** each flow $f \in \mathcal{F}$ with $S_f = UNFXD$ **Do**
07.             Submit $LP\ (f, R_{min})$ to LP solver;
08.             Get rate $r_f$ of flow $f$ from LP solver;
09.             **If** $r_f = R_{min}$ **Then**
10.                 Set $S_f = FIXED$ for flow $f$;
11.             **End If**
12.         **End For**
13.     **Until** $S_f = FIXED$ for all the flows $f \in \mathcal{F}$;
*End*
```

Figure 3.4. The MP algorithm

The MP algorithm in Figure 3.4 is associated with formulations $LP(g)$ in Figure 3.5 and $LP\ (g, R_{min})$ in Figure 3.6 to compute the max-min rates for multi-hop flows in a DC-WSN. The algorithm works like the WF algorithm, in which the smallest flow rates are found first, then the higher flow rates next, and the greatest flow rates last. However, it takes much less time for the MP algorithm than for the WF algorithm to find a flow rate, which includes two main stages in each round of loop Repeat Until from steps 2 to 13. Note that Figure 3.4 shows the complexity of the MP algorithm is $O(m^2(LP(m,p))^2)$, where $LP(m,p)$ is the complexity of linear programming determined by the number of flows m and p linear equalities in the LP formulations, i.e., $LP(f)$ or $LP(f, R_{min})$. For the sake of completeness we note that we use LP solvers that, theoretically, have exponential complexity, however in the most practical cases such as the ones encountered in this thesis the worst case behaviour did not emerge [48]. For our MP algorithm, the complexity is $O(m^2p^2)$.

$$max\ (r_g) \qquad g \in \mathcal{F}, S_g = UNFXD \quad (3.10)$$

Subject to

$$\sum_{k\in\mathcal{K};\, o(l)=n} r_f^{(k)}(l) - \sum_{k\in\mathcal{K};\, d(l)=n} r_f^{(k)}(l) = \begin{cases} r_f, & n = f_s \\ -r_f, & n = f_d \\ 0, & else \end{cases} \quad \forall n \in \mathcal{N}, f \in \mathcal{F} \quad (3.11)$$

$$\sum_{f\in\mathcal{F};\, l\in\mathcal{L};\, r_f^{(k)}(l)\in q} r_f^{(k)}(l) \leq r.\mathcal{T}^{(k)} \qquad \forall k \in \mathcal{K}, \forall q \in \mathcal{Q} \quad (3.12)$$

$$r_f = R_f \qquad S_f = FIXED, \forall f \in \mathcal{F} \quad (3.13)$$

$$r_f = r_g \qquad S_f = UNFXD, \forall f \in \mathcal{F} \quad (3.14)$$

$$r_f \geq 0,\ r_f^{(k)}(l) \geq 0 \qquad \forall f \in \mathcal{F}, l \in \mathcal{L}, k \in \mathcal{K} \quad (3.15)$$

Figure 3.5. Formulation $LP(g)$

In the first stage, from steps 3 to 5, the smallest rate R_{min} among those of the flows with undetermined rates is calculated by submitting formulation $LP(g)$ to LP solver, in which g is a flow with undetermined rate, i.e., $UNFXD$. In the second stage, from steps 6 to 12, the flows with their rates equal to R_{min} are identified, i.e., their numberical rates are determined, i.e., $FIXED$. This is done by submitting formulation $LP(g, R_{min})$ to LP solver (step 7) and then comparing their returned rates to R_{min} (steps 9 to 11), in which g is a flow with undetermined rate. Note that in step 1 the rates of all the flows are initialized to undetermined rates to properly start the process.

$$max\ (r_g) \qquad g \in \mathcal{F}, S_g = UNFXD \quad (3.16)$$

Subject to

$$\sum_{k\in\mathcal{K};\, o(l)=n} r_f^{(k)}(l) - \sum_{k\in\mathcal{K};\, d(l)=n} r_f^{(k)}(l) = \begin{cases} r_f, & n = f_s \\ -r_f, & n = f_d \\ 0, & else \end{cases} \quad \forall n \in \mathcal{N}, f \in \mathcal{F} \quad (3.17)$$

$$\sum_{f\in\mathcal{F};\, l\in\mathcal{L};\, r_f^{(k)}(l)\in q} r_f^{(k)}(l) \le r.\mathcal{T}^{(k)} \qquad \forall k \in \mathcal{K}, \forall q \in \mathcal{Q} \quad (3.18)$$

$$r_f = R_f \qquad S_f = FIXED, \forall f \in \mathcal{F} \quad (3.19)$$

$$r_f \ge R_{min} \qquad S_f = UNFXD, \forall f \in \mathcal{F} \quad (3.20)$$

$$r_f \ge 0,\ r_f^{(k)}(l) \ge 0 \qquad \forall f \in \mathcal{F}, l \in \mathcal{L}, k \in \mathcal{K} \quad (3.21)$$

Figure 3.6. Formulation $LP(g, R_{min})$

It is clearly seen that constraints, (3.11), (3.12), (3.13) and (3.15), of formulation $LP(g)$ in Figure 3.5 and constraints, (3.17), (3.18), (3.19) and (3.21), of formulation $LP\ (g, R_{min})$ in Figure 3.6, have the same meaning as constraints, (3.6), (3.7), (3.8) and (3.9), of formulation $LP(r_f)$ in Figure 3.1, respectively. Hence, two formulations $LP(g)$ and $LP\ (g, R_{min})$ need to be described the following constraints:

(1) The objective constraint (3.10) in $LP(g)$ is used in the first stage to find the smallest max-min rate R_{min} among the flows with the undetermined status $UNFXD$ by maximizing all the equal undetermined rates r_g of flows g;

(2) The constraint (3.14) in $LP(g)$ is to set the undetermined rates r_f (with status $UNFXD$) of flows $f \in \mathcal{F}$ to the rate value r_g, which are not determined by earlier steps in the MP algorithm. Note that this constraint associated with the objective constraint (3.10) is mainly aimed to maximize all the undetermined equal rates;

(3) The objective constraint (3.16) in $LP(g, R_{min})$ is used in the second stage to find the flows with status $UNFXD$, whose rates are greater than or equal to the

smallest rate R_{min} by maximizing only the undetermined rate r_g of flow g each time submitting formulation $LP(g, R_{min})$ to LP solver;

(4) The constraint (3.20) in $LP(g, R_{min})$ is to guarantee all the undetermined rates r_f are equal to or greater than the smallest rate R_{min}. Note that this constraint associated with the objective constraint (3.16) is to make sure that the undetermined rate r_g of flow g is both maximized and equal to or greater than the rate R_{min};

Proof 2. We can reuse Proof 1 of the WF algorithm as that of the MP algorithm because the latter is an improved version of the former just in the way to find each numerical flow rate, which does not impact the correctness of the proof.

To illustrate how the MP algorithm in Figure 3.4 figures out all the numerical flow rates, we reconsider Example 3.1 in Figure 3.3 and walk through the algorithm step-by-step on the network. Since we have two flows in the network, there are at most two iterations of loop *Repeat Until* in steps from 2 to 13. In the 1st iteration, after submitting $LP\ (f)$ to LP solver in step 4, we get rate R_{min} returned from LP solver in step 5, which is 0.0625 (packets/slot). It is the max-min rate, which is achievably-allocated to flows with undetermined rates, i.e. flows f_1 and f_2. This is because among the three cliques in three Figures 3.3.c.1, 3.3.c.2 and 3.3.c.3, the clique in Figure 3.3.c.3 is bottlenecked first based on the capacity constraint (3.12). Since there is only flow f_1 with undetermined rate in the bottlenecked clique, numerical rate of flow f_1 is figured out first by loop *For* in steps from 6 to 11. Note that from now on the rate of flow f_1 is determined and unchanged, i.e., with status $FIXED$. More specifically, the flow and sub-flow rates of all the flows on each stage are determined as follows.

(1) **Flow $\boldsymbol{f_1}$ -** Flow rate: $r_1 = 0.0625$;

Stage $\mathcal{T}^{(1)}$: $r_{f_1}^{(1)}(l_1) = 0.0625$; $r_{f_1}^{(1)}(l_2) = 0$; $r_{f_1}^{(1)}(l_3) = 0$;

Stage $\mathcal{T}^{(2)}$: $r_{f_1}^{(2)}(l_1) = 0$; $r_{f_1}^{(2)}(l_2) = 0.0625$; $r_{f_1}^{(2)}(l_3) = 0$;

Stage $\mathcal{T}^{(3)}$: $r_{f_1}^{(3)}(l_1) = 0$; $r_{f_1}^{(3)}(l_2) = 0$; $r_{f_1}^{(3)}(l_3) = 0.0625$;

(2) **Flow $\boldsymbol{f_2}$ -** Flow rate: $r_2 = 0.0625$;

Stage $\mathcal{T}^{(1)}$: $r_{f_2}^{(1)}(l_6) = 0.0625$; $r_{f_2}^{(1)}(l_5) = 0$;

Stage $\mathcal{T}^{(2)}$: $r_{f_2}^{(2)}(l_6) = 0$; $r_{f_2}^{(2)}(l_5) = 0.0625$;

In the 2nd iteration, after submitting $LP\ (f)$ to LP solver in step 4, we get rate R_{min} returned from LP solver in step 5, which is 0.125 (packets/slot). It is the max-min rate, which is to indicate the max-min rate achievably-allocated to flows with undetermined rates, i.e. flow f_2. This is because among the three cliques in three Figures 3.3.c.1, 3.3.c.2 and 3.3.c.3, two cliques in Figures 3.3.c.1 and 3.3.c.2 are both bottlenecked next based on the capacity constraint (3.12). Since there is only flow f_2 with undetermined rate in the bottlenecked cliques, numerical rate of flow f_2 is figured out next by loop *For* in steps from 6 to 11. More specifically, the flow and sub-flow rates of all the flows on each stage are determined as follows.

(1) **Flow $\boldsymbol{f_1}$ -** Flow rate: $r_1 = 0.0625$;

Stage $\mathcal{T}^{(1)}$: $r_{f_1}^{(1)}(l_1) = 0.0625$; $r_{f_1}^{(1)}(l_2) = 0$; $r_{f_1}^{(1)}(l_3) = 0$;

Stage $\mathcal{T}^{(2)}$: $r_{f_1}^{(2)}(l_1) = 0$; $r_{f_1}^{(2)}(l_2) = 0.0625$; $r_{f_1}^{(2)}(l_3) = 0$;

Stage $\mathcal{T}^{(3)}$: $r_{f_1}^{(3)}(l_1) = 0$; $r_{f_1}^{(3)}(l_2) = 0$; $r_{f_1}^{(3)}(l_3) = 0.0625$;

(2) **Flow $\boldsymbol{f_2}$ -** Flow rate: $r_2 = 0.125$;

Stage $\mathcal{T}^{(1)}$: $r_{f_2}^{(1)}(l_6) = 0.125$; $r_{f_2}^{(1)}(l_5) = 0$;

Stage $\mathcal{T}^{(2)}$: $r_{f_2}^{(2)}(l_6) = 0$; $r_{f_2}^{(2)}(l_5) = 0.125$;

3.2 SIMULATION-BASED PERIODIC PATTERN EXCISION

3.2.1 SCHEDULING ALGORITHM

The network is simulated using greedy traffic sources, i.e., capable to provide as much traffic as could possibly be accommodated by transmission opportunities. The transmissions that can be scheduled to occur simultaneously in the network are determined by invocations of a fast approximation [21] to the Maximum Weighted Independent Set (MWIS) problem. The simulation progresses in a slotted fashion, in each slot, sub-flows with backlog (i.e., non-zero queues) and whose corresponding link is active (both endpoints ON) are assigned a weight to be used by MWIS. The weight assignment at timeslot i tries to capture two aspects of the selection of which sub-flow (and hence link) to schedule over others: (a) sub-flows on link/arc l, of a flow f whose transmissions scheduled so far, denoted by $\#_f(i,l)$, lag compared to what should have been the analytically derived rate r_f are given higher weight to "catch" up with the rate they are supposed to achieve, and (b) sub-flows whose corresponding link which will be ON for a period of $a(l)$ and the nearest future point at which the link will become OFF (either one of the endpoints entering the OFF state) is time $u(i,l)$ (seen as a function of current time i) are given higher weight the closest we are to the "deadline" of $u(i,l)$. Symbolically, the weight assigned to the arc for the purposes of MWIS execution is:

$$weight(f,l,i) = max\left\{0, i.r_f - \#_f(i,l)\right\} + (a(l) - (u(i,l) - i)) \quad (3.22)$$

where the first component captures the sub-flow's lag and the second is an expression of the proximity to the link's next OFF deadline, increasing (hence becoming more "urgent") the closer the current time is to the deadline.

Figure 3.7 shows the pseudo-code of the scheduling algorithm in a given slot. Note that the algorithm only works for the ON sub-flows with available packets in transmission queues. With the input of the contention matrix M_SF $[0..\mathrm{M}-1][0..\mathrm{M}-1]$ of all the M contending sub-flows, the weight of each of the M' sub-flows of flow f over link l which are ON with available packets in slot i, the algorithm outputs the array S_SF $[0..\mathrm{M}''-1]$ of M'' ON sub-flows with available packets, which are to be scheduled in the timeslot. To this end, the algorithm performs the three following tasks. First, in step 2, function $weight$ () assigns weight value to each of the ON sub-flows as presented in (3.22), which indicates its priority among the other ON sub-flows for scheduling in the slot. Next, in step 3, function $graph$ () creates a contention graph $G\_SF[0..\mathrm{M}'-1][0..\mathrm{M}'-1]$ of M' weighted ON sub-flows with available packets, in which sub-flows are vertices and an edge between two vertices indicates the two contending sub-flows. Finally, in step 5, a maximum weighted independent set of M'' of the ON sub-flows is found and stored in array S_SF $[0..\mathrm{M}''-1]$ by function $MWIS$ () for scheduling.

```
Input:   Contention matrix $M\_SF$ [0..M-1][0..M-1] of all the M ON sub-flows;
         Weight of each of the $\mathrm{M}'$ ON sub-flows with available packets in slot $i$;
Output:  Array $S\_SF$ [0..M″-1] of $\mathrm{M}''$ of ON sub-flows to be scheduled in slot $i$;
Begin
01.   For Each ON sub-flow $(f, l)$ with available packets in queue  Do
02.       $w \leftarrow weight\ (f, l, i)$;
03.       $G\_SF\ [0..\mathrm{M}'-1][0..\mathrm{M}'-1] \leftarrow graph(f, l, w, M\_SF\ [0..\mathrm{M}-1][0..\mathrm{M}-1])$;
04.   End For
05.   $S\_SF\ [0..\mathrm{M}''-1] \leftarrow MWIS\ (G\_SF\ [0..\mathrm{M}'-1][0..\mathrm{M}'-1])$;
End
```

Figure 3.7. The scheduling algorithm during slot i

3.2.2 SCHEDULE EXCISION PROCESS

Without harm to generality and because the DC period is equal to T we anticipate the periodic steady state to develop over periods that are multiples of T (indeed as the experiments will show, in some cases, it is exactly T). We forego the discussion of how it is decided that the steady state has been reached, pointing to relevant literature on the topic, i.e., [53]. After executing in the steady state, the simulation-based is interrupted (at points equal to a multiple of T) and a simple pattern matching procedure is performed. The decisions made by the simulation of whether a sub-flow transmits or not in a given slot are stored in a two dimensional vector whereby rows are sub-flows and columns are activity (transmission or not) taking place in a particular time slot. The array is illustrated in Figure 3.8 where transmissions that occurred are represented as black circles (white if the corresponding sub-flow did not transmit). The most recent interval of length T is used as a prefix template that will be matched to see if it has occurred over the recent past (scanning backwards in time). Let us denote this as the T_{sample} template. The pattern of transmissions within T_{sample} is compared and assuming it is found to repeat periodically, it is evidence that not only during the T_{sample} prefix, but the entire pattern (of length $T_{schedule}$) between successive T_{sample} matches might be repeated. A complete comparison then takes place between the transmissions scheduled over the last $T_{schedule}$ intervals as delineated by the T_{sample} prefix and if the schedules are found in agreement, the most recent (later in simulation time) of them is excised as the schedule template. If not, the simulation continues and the same process is attempted at a later point in time. An exact match is not always possible and hence a mismatch "budget" for the comparison of the two last success $T_{schedule}$ intervals is accounted for.

The careful reader will notice that the periodic pattern excision we perform is only with respect to the pattern of transmissions constructed by the simulation-based process. Clearly, this is a source of approximation because the total state of the system (even if it is deterministic) is guided by the queueing behavior, i.e., the backlogs at the nodes. However, the pattern excision we perform disregards matching the queue backlogs. In other words, the excision we perform is based on partial state matching of the system. As long as the queues are on average at close to zero occupancy (which itself is a sign that the match between analytical rate and simulated rate are approximately equal) we anticipate little impact from ignoring the queue state and the results seem to confirm this view.

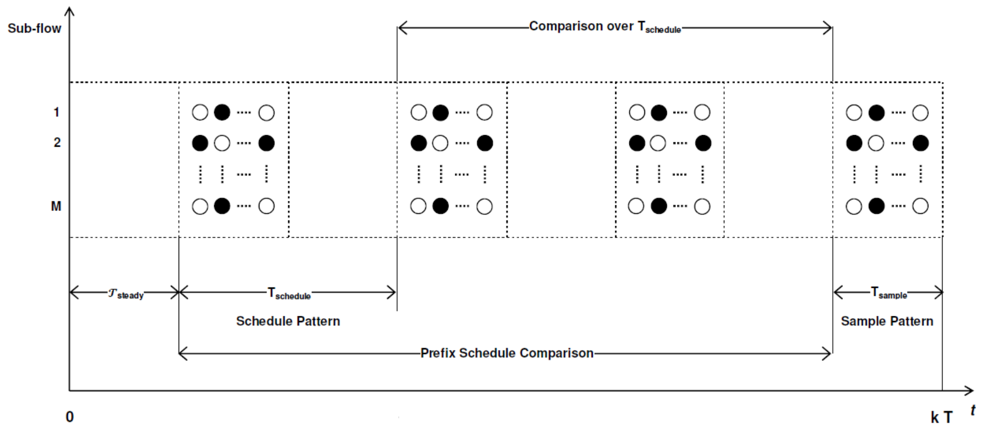


Figure 3.8. Illustration of the simulation-based schedule excision process

To further understand the excision process in Figure 3.8, we present main algorithms, which implement the process. For convenience of description, we first present notations and their functions in the algorithms before we elaborate each of the algorithms.

We suppose that the scheduling of M sub-flows in each timeslot after the steady state is detected, i.e., *Start Slot*, up to the start moment of executing the excision process, i.e., *Check Slot*, have been stored in array *ScheduleArray* [0..$T_{schedule}$-1][0..M-1]. Note that values of *Start Slot* and *Check Slot* are chosen as a multiple of T. A sample pattern is extracted during the last T_{sample} slots of these slots and stored in array *SampleArray* [0..T_{sample}-1][0..M-1]. Note that T_{sample} from the above description is a pre-assigned value of T. Next, the sample pattern is used to approximately match with a series of scheduling patterns that is started backwards in time from *Check Slot* to *Start Slot*. If the sample pattern is approximately found at a given number of times MAX_REPTS, i.e., periodically found, then two values can be returned, $\mathcal{T}_{steady}$, time from *Start Slot* the scheduling becomes steady, and the scheduling period, $T_{schedule}$, time between two sequential found sample patterns.

There are three main algorithms used in the excision process, which are **patternMatching** (), **periodMatching** () and **slotMatching** (). Algorithm **patternMatching** () is used to verify if given a threshold of missed-slots, i.e., *MismatchThreshold*, there is any matching between the sub-flow scheduling in array *ScheduleArray* [0..$T_{schedule}$-1][0..M-1], or *ScheduleArray* [ ][ ] for short, and that in array *SampleArray* [0..T_{sample}-1][0..M-1], or *SampleArray* [ ][ ] for short, from slot *StartSlot* to slot *CheckSlot*. In each slot *StartSchedule* between *StartSlot* and *CheckSlot*, the algorithm calls algorithm **periodMatching** () to verify the matching between the sub-flow scheduling in array *ScheduleArray* [ ][ ] and that in array *SampleArray* [ ][ ] during *Duration* given a threshold *MismatchThreshold*. To this end, algorithm **slotMatching** () is used to verify the matching between the sub-flow scheduling in array *ScheduleArray* [ ][ ] at slot *ScheduleSlot* and that in array *SampleArray* [ ][ ] at slot *SampleSlot*.

Hence, the periodicity of the prefix template T_{sample}, i.e., the prefix schedule comparison in Figure 3.8 is performed by algorithm **patternMatching** () including algorithms **patternMatching** (), **periodMatching** () while the entire template $T_{schedule}$, i.e., the comparison over $T_{schedule}$ in Figure 3.8, is done by algorithm **periodMatching** (), in which *Duration* is set to $T_{schedule}$.

Input: Slot *SampleSlot*; Slot *ScheduleSlot*;

Output: YES if the scheduling in *SampleArray* [ ][ ] at *SampleSlot* is matching

with that in *ScheduleArray* [ ][ ] at *ScheduleSlot* or NO otherwise;

```
Begin
01.  For $i$ from 0 to M – 1  Do
02.      If  ($SampleArray\ [SampleSlot][i] \neq$
             $ScheduleArray\ [ScheduleSlot][i]$)  Then
03.          Return NO;
04.      End If
05.  End For
06.  Return YES;
End
```

Figure 3.9. Pseudo-code of algorithm **slotMatching ()**

Figure 3.9 shows the pseudo-code of algorithm **slotMatching** (), in which loop **For** in steps from 1 to 6 traverses each of the M sub-flows. For each sub-flow, in steps from 2 to 4, the algorithm checks if the scheduling in *SampleArray* [ ][ ] at slot *ScheduleSlot* matches that in *ScheduleArray* [ ][ ] at slot *SampleSlot*.

```
Input: Slot StartSchedule; Duration Duration; Threshold MismatchThreshold;
Output: YES if the scheduling in SampleArray [ ][ ] is matching with that in
        ScheduleArray [ ][ ] from StartSchedule during Duration or NO
        otherwise;
Begin
01.  MismatchCount = 0;
02.  For Each i from 0 to Duration – 1  Do
03.      If  (slotMatching (i, StartSchedule + i) = NO)  Then
04.          If  (MismatchCount = MismatchThreshold)  Then
05.              Return NO;
```

```
06.             **Else**
07.                 *MismatchCount*++;
08.             **End If**
09.         **End If**
10.     **End For**
11.     **Return** YES;
*End*
```

Figure 3.10. Pseudo-code of algorithm **periodMatching ()**

Figure 3.10 shows the pseudo-code of algorithm **periodMatching** (), in which loop **For** in steps from 2 to 10 traverses each slot during *Duration*. In each slot, from steps 3 to 9 it checks if the scheduling in array *SampleArray* [ ][ ] at slot *i* matches that in array *ScheduleArray* [ ][ ] at slot *StartSchedule*. In case of mismatching, from steps 4 to 8 it checks if *MismatchThreshold* is reached and then returns NO. Otherwise, *MismatchCount* is increased by one.

```
**Input:** Slot *StartSlot*; Slot *CheckSlot*; Threshold *MismatchThreshold*;
**Output:** Slot *FoundSlot* and YES if the scheduling in *SampleArray* [ ][ ] is
            matching with that in *ScheduleArray* [ ][ ] from *StartSlot* during
            *Duration* or NO otherwise;
*Begin*
01.     *Count* = 0; *RepeatThreshold* = MAX_REPTS – 1; *Duration* = $T_{sample}$;
02.     **For** Each *i* from *CheckSlot* – *Duration* + 1 to *StartSlot*  **Do**
03.         **If**  (*periodMatching* (*i*, *Duration*, *MismatchThreshold*) = YES)  **Then**
04.             **If**  (*Count* = 0)  **Then**
05.                 *Attempt0* = *i*; *Count*++;
06.             **Else**
```

```
07.                 If   (Count = 1)   Then
08.                     Attemp1 = i; Count++; PrevCount = i;
09.                 Else
10.                     If   (i – PrevCount = Attempt1 – Attempt0)   Then
11.                         If   (Count = RepeatThreshold)   Then
12.                             T_Schedule  = i – PrevCount;
13.                             T_Steady = Attemp0 – StartSlot + 1;
14.                             Mis_Slots = MismatchThreshold;
15.                             FoundSlot = Attemp0;
16.                             Return YES;
17.                         Else
18.                             Count++; PrevCount = i;
19.                         End If
20.                     End If
21.                 End If
22.              End If
23.         End If
24.     End For
25.     Return NO;
End
```

Figure 3.11. Pseudo-code of algorithm **patternMatching ()**

Figure 3.11 shows the pseudo-code of algorithm **patternMatching** (), in which loop **For** in steps from 2 to 24 traverses each of the slots from *StartSlot* to *CheckSlot* – *Duration* + 1. For each slot *i*, from steps 3 to 23 it checks if any matching between the scheduling in array *ScheduleArray* [ ][ ] and that in array

SampleArray [ ][ ] during *Duration*, i.e, T_Sample or T_{sample}, given a threshold *MismatchThreshold*. Steps 4 to 22 determine (1) how long the schedule, i.e., *T_Schedule* or $T_{schedule}$, and the steady time, i.e., *T_Steady* or $\mathcal{T}_{steady}$ in Figure 3.8, are; (2) how many times the schedule is repeated, i.e., MAX_REPTS; (3) what the number of missed-slots, i.e., *MismatchThreshold*, is; and (4) where the found slot, i.e., *FoundSlot*, is.

3.2.3 FLOW BALANCE APPROACHES

A final technicality is that the excised schedule pattern (whose periodic repetition will be the schedule executed by the nodes) does not necessarily obey the flow conservation constraint. For example the excised schedule does not necessarily offer the same number of transmissions for inbound traffic of a flow to a node as it does for outbound traffic of the same flow from that node. This is because the (analytical) flow conservation property pertains to an infinite time horizon but fluctuations (imbalances) are always possible over small intervals of time as long as they are cancelled out in the long run. To address this point we use an ad-hoc strategy of equating the inbound and outbound transmission for all sub-flows of a flow either by making it a criterion during the pattern matching process (option A) that has to be satisfied as much as possible or enforcing it after the excision has taken place (option B) by trimming away the transmissions that result in the imbalance (at the loss of some throughput).

To implement options A and B, we use two algorithms **CheckBalance ()** and **MakeBalance ()** presented by pseudo-codes in Figures 3.12 and 3.13, respectively. Algorithm **CheckBalance ()** is to check if all the flows in a given schedule template are balanced, which is used during the pattern matching process. Meanwhile, algorithm **MakeBalance ()** is to enforce the flow balance on a given imbalanced schedule template, which is used after the pattern matching process.

With the pseudo-code in Figure 3.12, we suppose that a given schedule template with length $T_{schedule}$ and M sub-flows is stored in array *ScheduleArray* [0..$T_{schedule}$ – 1][0..M – 1]. We also assume that the minimum value, i.e., $min\ (f)$, which is among the numbers of transmissions of a given flow f over each of its links l along its routing path, is predetermined. In addition, we use variable $\#_f(l)$ to store the number of transmissions of each flow f over link l, which is initialized to zero in step 1. The variable is increased by one whenever f over link l is scheduled, i.e., ACTIVE (step 3) during traversing the template (steps 2 to 9). Note that notation (f, l) is to indicate flow f goes over link l. The algorithm returns imbalance indication (step 6) whenever the variable is greater than the minimum respective value $min\ (f)$ or balance indication otherwise (step 10).

```
Input:  Schedule array ScheduleArray [0..$T_{schedule}$ – 1][0..M – 1];
        Number $min\ (f)$ of each flow $f \in \mathcal{F}$ in the array;
Output:  YES if the array is balanced or NO otherwise;
Begin
01.   Set $\#_f(l)$ to zero for all the flows $f \in \mathcal{F}$ over link $l \in \mathcal{L}$;
02.   For  ($i$ from 0 to $T_{schedule}$ – 1, $j$ from 0 to M – 1)  Do
03.       If  ((ScheduleArray [$i$][$j$] = ACTIVE) and ($j$ = $(f, l)$))  Then
04.           $\#_f(l) = \#_f(l) + 1$;
05.           If  ($\#_f(l) > min\ (f)$)  Then
06.               Return NO;
07.           End If
08.       End If
09.   End For
10.   Return YES;
End
```

Figure 3.12. Pseudo-code of algorithm **CheckBalance ()**

With the pseudo-code in Figure 3.13, we suppose that a given imbalanced schedule template with length $T_{schedule}$ and M sub-flows is stored in array *ScheduleArray* [0..$T_{schedule}$ – 1][0..M – 1]. We also assume that we have walked through the active spots of sub-flows of each flow along its routing path from the source to the sink in the template and accordingly marked the active spots in array *VisitedArray* [0..$T_{schedule}$ – 1][0..M – 1]. Note that only an active spot is marked as visited only when we have walked through the entire path, i.e., we have reached the sink. Hence, active spots, which have not marked yet, i.e., NO, are considered as imbalanced spots by step 2. The imbalanced active spots need to be removed from the template, i.e., turned to inactive spots, i.e., INACTIVE, by step 3. Eventually, after traversing the whole template, the algorithm turns the given schedule array from an imbalanced template to a balanced one.

```
Input:  Imbalanced array ScheduleArray [0..$T_{schedule}$ – 1][0..M – 1];
        Visited array VisitedArray [0..$T_{schedule}$ – 1][0..M – 1];
Output:  Balanced array ScheduleArray [0..$T_{schedule}$ – 1][0..M – 1];
Begin
01. For  ($i$ from 0 to $T_{schedule}$ – 1, $j$ from 0 to M – 1)  Do
02.     If  ((ScheduleArray [$i$][$j$] = ACTIVE) and (VisitedArray [$i$][$j$] = NO)) Then
03.         ScheduleArray [$i$][$j$] = INACTIVE;
04.     End If
05. End For
End
```

Figure 3.13. Pseudo-code of algorithm **MakeBalance ()**

To illustrate the schedule excision process after flow balance is checked and guaranteed, we run the scheduling algorithm on the network in Example 3.1. As a result, we construct the schedule in Figure 3.14, in which timeslot $\boldsymbol{t}$ is within duration T_{schedule} from 1 to 32, and sub-flow $\boldsymbol{r_f^{(k)}(l)}$ indicates flow f, i.e., from f_1 to f_2, over link l, i.e., from l_1 to l_6, across stages k, i.e., from 1 to 3. Note that white circles (black circles) mean the respective sub-flows are not scheduled (are scheduled).

From the results in Figures 3.14, we observe that the constructed schedule is guaranteed flow balance. For example, the rates of flow f_1 over links l_1, l_2 and l_3 along the routing path are equal to 2/32 or **0.0625** while the rates of flow f_2 over links l_5 and l_6 along the routing path are equal to 4/32 or **0.125**. The results are consistent with those from the MP algorithm as described above.

$\boldsymbol{t}$	$\boldsymbol{r_{f_1}^{(k)}(l_1)}$	$\boldsymbol{r_{f_1}^{(k)}(l_2)}$	$\boldsymbol{r_{f_1}^{(k)}(l_3)}$	$\boldsymbol{r_{f_2}^{(k)}(l_5)}$	$\boldsymbol{r_{f_2}^{(k)}(l_6)}$
1	○	○	○	○	●
2	○	○	○	○	●
3	○	○	○	○	●
4	○	○	○	○	●
5	●	○	○	○	○
6	●	○	○	○	○
7	○	○	○	●	○
8	○	○	○	●	○
9	○	○	○	●	○
10	○	●	○	○	○
11	○	○	○	●	○
12	○	●	○	○	○

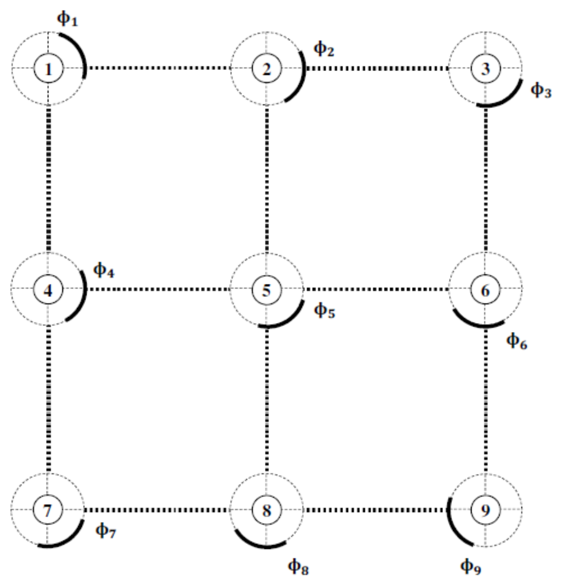
13	○	○	○	○	○
14	○	○	○	○	○
15	○	○	○	○	○
16	○	○	○	○	○
17	○	○	●	○	○
18	○	○	●	○	○
19	○	○	○	○	○
…	○	○	○	○	○
32	○	○	○	○	○

Figure 3.14. Schedule constructed in Example 3.1

3.3 EVALUATION

3.3.1 OVERALL PERFORMANCE

In order to evaluate the ability of the proposed technique to produce schedules that capture the desired performance objectives, we restrict ourselves to examples of regular topologies and experiment with different per-node phases. There is no restriction to just regular topologies for the application of the proposed scheme, but the reader can follow easily the layout, phase relation, and flow paths on a regular topology.



(a) 3x3 Grid topology

Flow Id	Routing Path
F_1	1 → 2 → 3
F_2	4 → 5 → 6
F_3	7 → 8 → 9
F_4	1 → 4 → 7
F_5	2 → 5 → 8
F_6	3 → 6 → 9

(b.1) Flow pattern PT-1

Flow Id	Routing Path
F_1	1 → 4 → 5 → 8 → 9
F_2	4 → 5 → 6
F_3	7 → 8 → 5 → 6 → 3
F_4	8 → 5 → 2

(b.2) Flow pattern PT-2

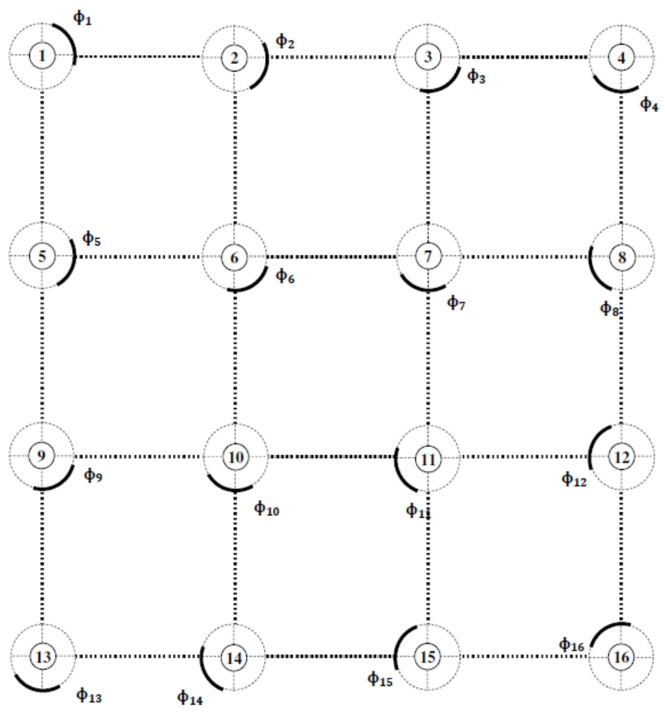
Flow Id	Routing Path
F_1	4 → 5 → 2
F_2	7 → 8 → 5 → 6 → 3
F_3	8 → 9 → 6
F_4	4 → 7 → 8
F_5	1 → 4 → 5 → 8 → 9
F_6	2 → 5 → 6

(b.3) Flow pattern PT-3

Phase Gaps	ϕ_1, ϕ_2, ϕ_3 ϕ_4, ϕ_5, ϕ_6 ϕ_7, ϕ_8, ϕ_9
2	0 , 2 , 4 2 , 4 , 6 4 , 6 , 8
4	0 , 4 , 8 4 , 8 , 12 8 , 12 , 16
6	0 , 6 , 12 6 , 12 , 18 12 , 18 , 24
8	0 , 8 , 16 8 , 16 , 24 16 , 24 , 0
10	0 , 10 , 20 10 , 20 , 30 20 , 30 , 8

(c) Phase scheme for G3x3

Figure 3.15. Topologies, flows and phases used in the simulations



(a) 4x4 Grid topology

Flow Id	Routing Path
F_1	1 → 2 → 3 → 4
F_2	5 → 6 → 7 → 8
F_3	9 → 10 → 11 → 12
F_4	13 → 14 → 15 → 16
F_5	1 → 5 → 9 → 13
F_6	2 → 6 → 10 → 14
F_7	3 → 7 → 11 → 15
F_8	4 → 8 → 12 → 16

(b) Flow pattern PT-1

Phase Gaps	ϕ_1 , ϕ_2 , ϕ_3 , ϕ_4
	ϕ_5 , ϕ_6 , ϕ_7 , ϕ_8
	ϕ_9 , ϕ_{10}, ϕ_{11}, ϕ_{12}
	ϕ_{13}, ϕ_{14}, ϕ_{15}, ϕ_{16}
2	0 , 2 , 4 , 6
	2 , 4 , 6 , 8
	4 , 6 , 8 , 10
	6 , 8 , 10 , 12
4	0 , 4 , 8 , 12
	4 , 8 , 12 , 16
	8 , 12 , 16 , 20
	12 , 16 , 20 , 24
6	0 , 6 , 12 , 18
	6 , 12 , 18 , 24
	12 , 18 , 24 , 30
	18 , 24 , 30 , 4
8	0 , 8 , 16 , 24
	8 , 16 , 24 , 0
	16 , 24 , 0 , 8
	24 , 0 , 8 , 16
10	0 , 10 , 20 , 30
	10 , 20 , 30 , 8
	20 , 30 , 8 , 18
	30 , 8 , 18 , 28

(c) Phase scheme for G4x4

Figure 3.16. Topologies, flows and phases used in the simulations

We consider a 3x3 and a 4x4 topology as shown in Figures 3.15.a and 3.16.a. Circular arcs are used to pictorially depict which nodes have the same phases (e.g., in the 3x3 topology nodes 3, 5, and 7 have the same phase). The duty cycle period is 32 slots and the ON intervals are all 12 slots long. Given that the phases can be staggered differently, we explore the impact of staggering by various "phase gaps" (Figures 3.15.c and 3.16.c) and indicate the phase (counted in slots) at which the corresponding node switches to ON. Additionally, a number of flows are simulated in each configuration following certain patterns groups (PT-1 to PT-3 for 3x3 and PT-4 for 4x4 topology) shown in Figures 3.15.b.1 to 3.15.b.3 and 3.16.b noting that the routes in each pattern group were computed based on single-shortest-path time-varying routing. The path directions of the four pattern groups are distinctly different; in patterns PT-1 and PT-4, the directions are along the sides of the grid; in pattern PT-2, the directions cross the center of the grid; in pattern PT-3, the directions are parallel with the diagonals of the grid.

Another facet of the experiments is the phase relation between adjacent nodes. We consider three schemes:

1. *Synchronized Phases* (SP), in which all the phases are synchronized to start at the same point in time. This is a benchmark value which essentially eliminates the time-varying nature of the communication graph.

2. *Fixed Ladder Phases* (FLP), in which any two adjacent nodes in the grid (vertically or horizontally) have their phases staggered 2 slots apart. This scheme is meant to test the impact of flow patterns and their interference on the schedule construction but with a fairly benign impact by the phase differences, i.e., adjacent ON periods overlap significantly over relatively long periods of time.

3. *Varied Ladder Phases* (VLP), in which we vary the staggering of the phases from 2 to 10 slots (Figures 3.15.c and 3.16.c for topologies 3x3 and 4x4 respectively), creating increasingly "difficult" short periods over which links are

active. Note that when the VLP scheme is used, the number of flows in each flow pattern is kept maximized and unchanged, i.e., in PT-1, PT-2, PT-3 and PT-4 the number of flows are 6, 4, 6 and 8, respectively.

In all cases, and as a matter of convention, instead of denoting the specific flows that comprise a certain mix, we create larger sets of flows by adding more flows in their numerical order. That is, sets of 1, 2, 3, etc. flows comprise correspondingly of the flows {F1}, {F1, F2}, {F1, F2, F3}, etc. Finally, note that because of the time varying nature of the underlying communication graph, the set of parameters used here that guides the duty cycling, generates a great deal of actual link topologies.

Representative results are summarized in Tables 3.1 (for PT-1 and PT-2) and 3.2 (for PT-3 and PT-4). The **#F or Gap** is the number of flows for phase schemes SP and FLP, or the stagger gap for phase scheme VLP. **Pre-Sim** is the total throughput of all the flows with sub-flows scheduled in the constructed excised periodic schedule. **Numerical** is the total throughput of all the flows including numerical per-flow rates as derived from the water filling (WF) or max-min programming (MP) algorithm associated with the max-min fairness formulation. In other words, Pre-Sim are simulation results while Numerical are analytical values. **Fairness** is a fairness index between what should be the achieved rates (as per water-filling) and what is achieved by the excised schedule. **Tschedule/T** expresses how many multiples of the duty cycle period is the length of the excised periodic schedule. **Mismatch** captures the fraction as percentage of sub-flows who did not match exactly during the prefix comparison. **Err** captures the fraction of sub-flows that did not match exactly during the complete schedule length comparison. The note field indicates whether Option A or Option B for correcting the flow balance was necessary in the corresponding case and which of the two options produced the best results.

The virtually identical numbers in the Pre-Sim and Numerical columns demonstrate that the process of schedule excision is a reasonable approach to produce accurate schedules. Despite the fact that a key component of the simulation-based process is the invocation of an approximation to MWIS [21] the fact is that a schedule is constructed via multiple (one per slot) invocation of MWIS, hence what matters is less the worst-case behavior of the approximation and more the average performance.

Leaving MWIS aside, the process of matching the T_{sample} prefix cannot be expected to lead to perfect matches. For this reason, we allowed a maximum of mismatch (seen as difference in transmissions scheduled for a sub-flow between two compared patterns) of 10% which turned out to be very pessimistic, i.e., we did not reach that degree of mismatch. Given that the prefix is only a fraction of the possible eventual schedule, the mismatch is generally larger (denoted by Err in our performance tables) when comparing complete $T_{schedule}$ periods but again not as large as we had originally anticipated.

Table 3.1. Simulation results with PT-1 and PT-2

Pattern	#F or Gap	Pre-Sim	Numerical	Fairness	Tschedule/T	Mismatch	Err	Scheme	Notes
PT-1	1	0.1875	0.1875	1	1	0	0	SP	
PT-1	2	0.1875	0.1875	1	1	0	0	SP	
PT-1	3	0.28125	0.28125	1	1	0	0	SP	
PT-1	4	0.28125	0.28125	1	4	0	0	SP	
PT-1	5	0.234375	0.234375	1	2	0	0	SP	
PT-1	6	0.28125	0.28125	1	2	0	0	SP	
PT-1	1	0.1875	0.1875	1	1	0	0	FLP	
PT-1	2	0.21875	0.21875	1	1	0	0	FLP	
PT-1	3	0.328125	0.328125	1	2	0	0	FLP	
PT-1	4	0.3359375	0.34375	0.99	4	0	0	FLP	Option B
PT-1	5	0.3125	0.3125	1	1	0	0	FLP	
PT-1	6	0.328125	0.328125	1	4	0	0	FLP	
PT-1	2	0.328125	0.328125	1	4	0	0	VLP	

PT-1	4	0.375	0.375	1	1	0	0	VLP	
PT-1	6	0.375	0.375	1	1	0	0	VLP	
PT-1	8	0.234375	0.234375	1	4	0	0	VLP	
PT-1	10	0.09375	0.09375	1	2	0	0	VLP	
PT-2	1	0.125	0.125	1	1	0	0	SP	
PT-2	2	0.15	0.15	1	5	0	0	SP	
PT-2	3	0.140625	0.140625	1	2	0	0	SP	
PT-2	4	0.15	0.15	1	5	0	0	SP	
PT-2	1	0.140625	0.140625	1	2	0	0	FLP	
PT-2	2	0.171875	0.171875	1	4	0	0	FLP	
PT-2	3	0.15234375	0.15234375	1	8	0	0	FLP	
PT-2	4	0.15625	0.15625	1	4	0	0	FLP	
PT-2	2	0.15625	0.15625	1	4	0	0	VLP	
PT-2	4	0.15625	0.15625	1	4	0	0	VLP	
PT-2	6	0.125	0.125	1	1	0	0	VLP	
PT-2	8	0.078125	0.078125	1	8	0	0	VLP	
PT-2	10	0.03125	0.03125	1	4	0	0	VLP	

Another aspect of the results is that the produced schedule of duration $T_{schedule}$ is a small multiple of T (the DC period). The less loaded the network with flows, the smaller this multiple tends to be, but there are exceptions. Fundamentally, the relation $T_{schedule}/T$ reveals the impact of the queue state. That is, the queues, even though they tend to behave periodically, exhibit dynamics that develop across multiple T periods, as traffic possibly enters (inflating the queue) in one period to be delivered (not necessarily completely) to the next hop in the next period.

Table 3.2. Simulation results with PT-3 and PT-4

Pattern	#F or Gap	Pre-Sim	Numerical	Fairness	$T_{schedule}/T$	Mismatch	Err	Scheme	Notes
PT-3	1	0.1875	0.1875	1	1	0	0	SP	
PT-3	2	0.1484375	0.1484375	1	8	0	0	SP	
PT-3	3	0.15234375	0.15234375	1	8	0	0	SP	
PT-3	4	0.1875	0.1875	1	2	0	0	SP	
PT-3	5	0.15625	0.15625	1	1	0	0	SP	
PT-3	6	0.1640625	0.1640625	1	8	0	0	SP	

PT-3	1	0.15625	0.15625	1	1	0	0	FLP	
PT-3	2	0.1484375	0.1484375	1	8	0	0	FLP	
PT-3	3	0.1875	0.1875	1	1	0	0	FLP	
PT-3	4	0.1953125	0.1953125	1	4	0	0	FLP	
PT-3	5	0.1875	0.1953125	1	5	0	0	FLP	Option A
PT-3	6	0.1875	0.1875	1	1	0	0	FLP	
PT-3	2	0.1875	0.1875	1	1	0	0	VLP	
PT-3	4	0.1953125	0.1953125	1	4	0	0	VLP	
PT-3	6	0.234375	0.25	0.98	2	0	0	VLP	Option A
PT-3	8	0.1484375	0.15625	0.99	8	0	0	VLP	Option A
PT-3	10	0.0625	0.0703125	0.96	4	0	0	VLP	Option A
PT-4	1	0.125	0.125	1	1	0	0	SP	
PT-4	2	0.1875	0.1875	1	1	0	0	SP	
PT-4	3	0.28125	0.28125	1	1	0	0	SP	
PT-4	4	0.375	0.375	1	1	0	0	SP	
PT-4	5	0.3515625	0.3515625	1	4	0	0	SP	
PT-4	6	0.3125	0.3258929	0.93	1	0	0	SP	Option A
PT-4	7	0.328125	0.328125	1	2	0	0	SP	
PT-4	8	0.375	0.375	1	2	0	0	SP	
PT-4	1	0.145833	0.145833	1	3	0	0	FLP	
PT-4	2	0.21875	0.21875	1	2	0	0	FLP	
PT-4	3	0.28125	0.28125	1	1	0	0	FLP	
PT-4	4	0.375	0.375	1	1	0	0	FLP	
PT-4	5	0.4	0.4296875	0.99	10	0	0	FLP	Option B
PT-4	6	0.3515625	0.375	0.96	4	0	0	FLP	Option B
PT-4	7	0.328125	0.328125	1	2	0	0	FLP	
PT-4	8	0.375	0.375	1	4	0	0	FLP	
PT-4	2	0.375	0.375	1	4	0	0	VLP	
PT-4	4	0.375	0.375	1	2	0	0	VLP	
PT-4	6	0.5	0.5	1	1	0	0	VLP	
PT-4	8	0.3125	0.3125	1	4	0	0	VLP	
PT-4	10	0.125	0.125	1	2	0	0	VLP	

Across the board, we note that the first set of patterns PT-1 and PT-2 (Table 3.1) rarely resulted in excised schedule templates that did not exhibit flow balance. The flow imbalance (and hence the need for corrective action) was more prevalent in PT-3 and PT-4 (Table 3.2). Moreover when flow balance correction had to be applied, it was always for schedules that were larger than a single T period, i.e., who had some queueing dynamics influencing successive T periods. We conjecture from what we have seen so far that the problem lies with flows that, as per the numerical findings, ought to have been given a tiny fraction of the data rate. Hence situations in which one such flow is scheduled (during the simulation) for a single transmission for one of its sub-flows over a long run of time slots, is a flow that will likely demonstrate a mismatch during the excision process, or will exhibit a flow imbalance in the excised schedule. Indirect evidence to this end is the fact that when the flow balance corrections (options A or B) were taken, after removing the flow imbalance, certain flows ended up being (unfairly) victimized, i.e. losing the little allocation they had. Also, the less the overlap (the larger the phase stagger between nodes) the links remain active only briefly, resulting in smaller rates assigned to flows traversing them, as they become more "bottlenecked" leading to smaller (and hence problematic to schedule as just mentioned) rate allocations.

3.3.2 TOTAL THROUGHPUT

In this section, we evaluate our work by verifying in more detail the impact of the main factors on the total throughput, which are different types of flow patterns, i.e., PT-1 to PT-4, different types of phases schemes, i.e., SP, FLP and VLP, changes of the number of flows of the same pattern and scheme, and changes of the gaps between two neighbors' duty-cycles.

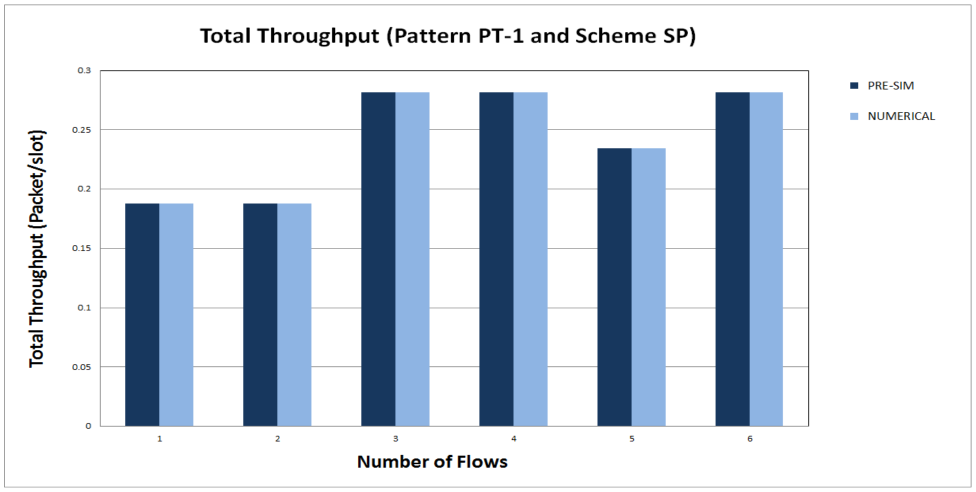
To this end, from the results in Tables 3.1 and 3.2, we plot bar charts in Figures 3.17 to 3.20 (corresponding to four patterns, PT-1 to PT-4 with three schemes SP, FLP and VLP), in which X-axis represents the number of flows (with SP and FLP) or the gaps between two neighbors' duty-cycles (with VLP), and Y-axis represents the total throughput (in packets/slot). The **PRE-SIM** and

NUMERICAL indicate **Pre-Sim** and **Numerical** values in Tables 3.1 and 3.2, respectively, from which we capture the follwing.

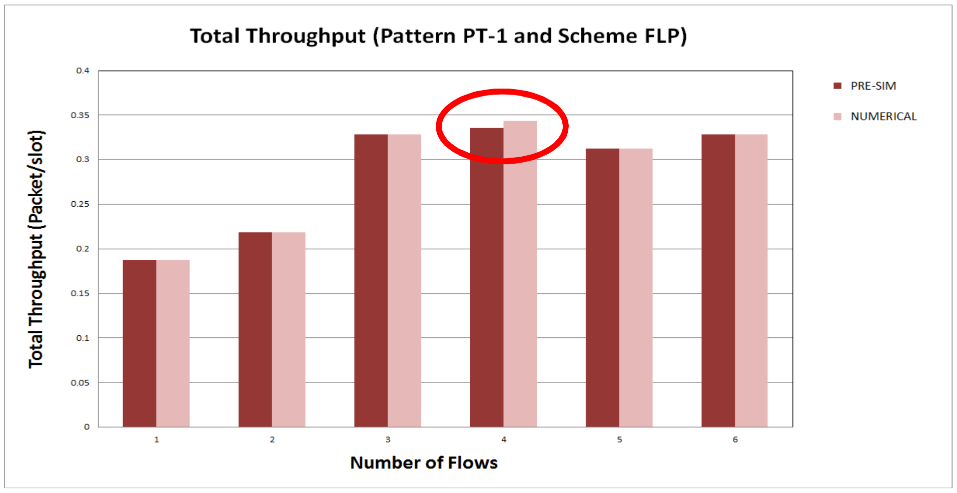
With the same pattern and the same number of flows in schemes SP and FLP, the gap between PRE-SIM and NUMERICAL values is greater in SP than in FLP, which indicates the impact of phase changes in FLP more than that in SP on the scheduling algorithm. This is because when the number of flows is increased, topology change in FLP happens more often than in SP.

The gap between PRE-SIM and NUMERICAL values of the same number of flows is almosts the same in PT-1 and PT-2 while the gap is larger in PT-3 and PT-4. This is due to more sub-flows in PT-3 and PT-4 than in PT-1 and PT-2. As a result, there is more impact on the scheduling alsorithm in PT-3 and PT-4 than in PT-1 and PT-2. Note that PRE-SIM and NUMERICAL values are not always the exactly same for the mismatches during the schedule excision process and then the flow balance corrections (options A or B) are taken as discussed in the previous sections. Such a case is shown in Table 3.1 with Pattern PT-1, #F 4 and Scheme FLP (in a red rectangle). The respective case is shown by two bars in Figure 3.17.b (in a red circle).

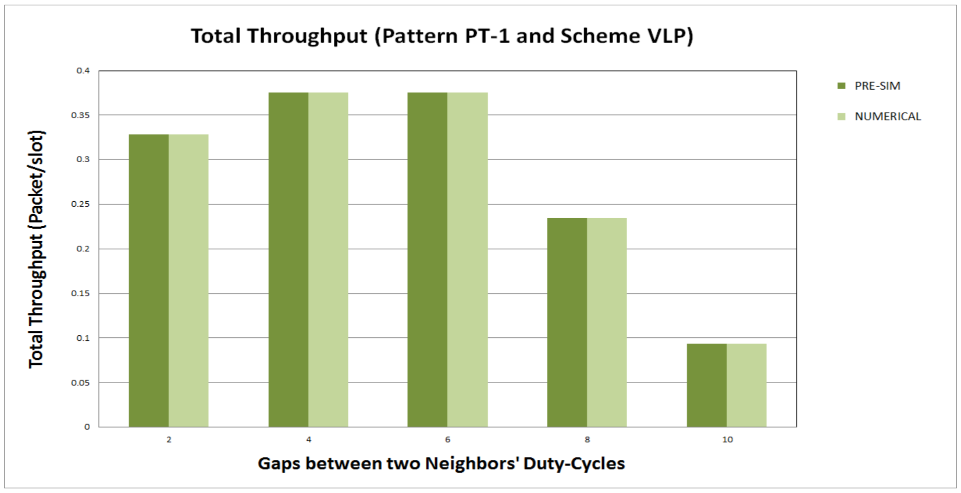
In VLP, when the gaps between two neighbors' duty-cycles increase, the throughput decreases. The main reason is that the greater gaps cause the smaller overlaps between two neighbors' duty-cycles and hence the more congestion at the neighbors. Also, the flow rates get smaller and then easily become victimized by the excision process as shown in Figure 3.19.c.



(a)

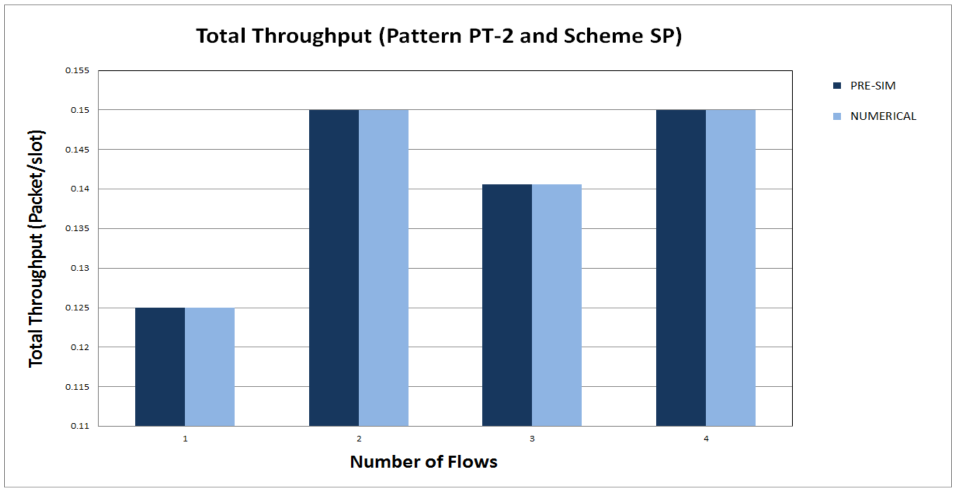


(b)

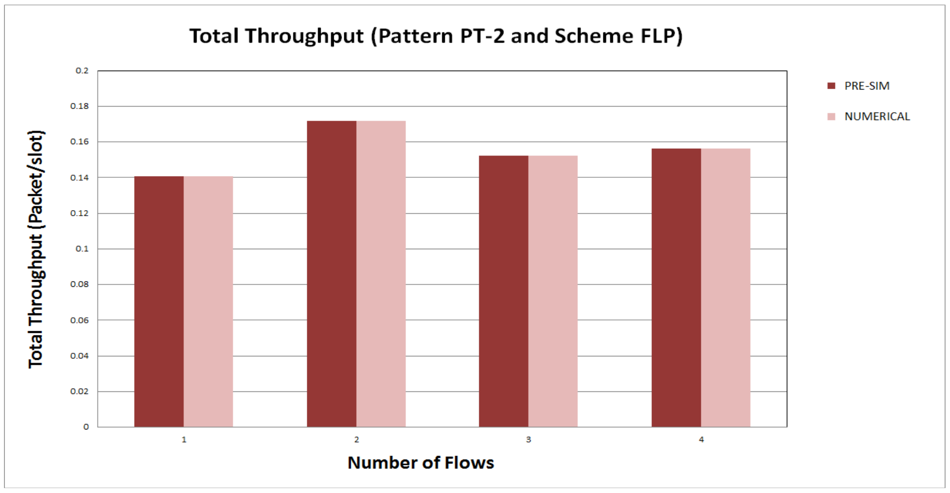


(c)

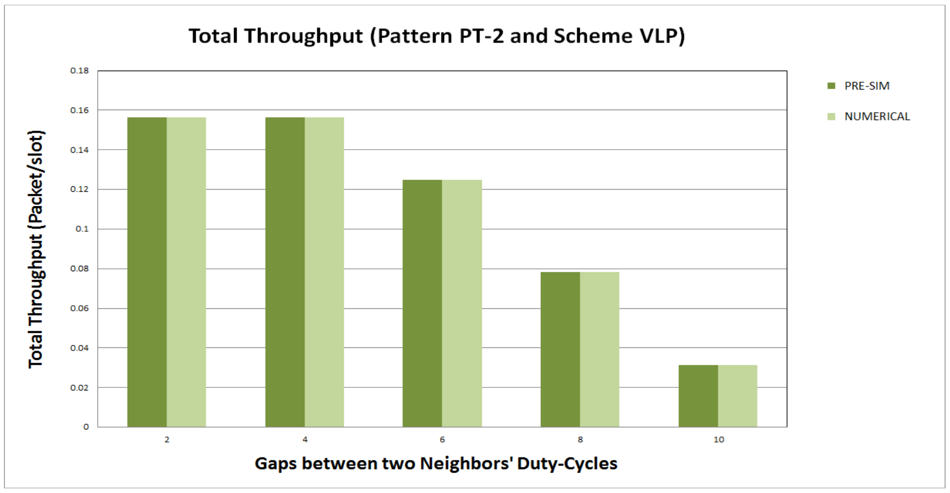
Figure 3.17. Throughput in 3x3 grid with pattern PT-1



(a)

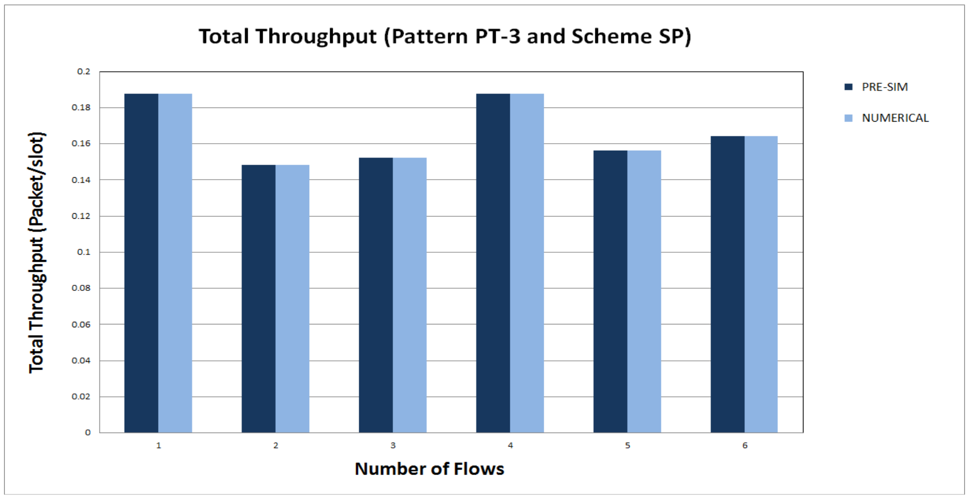


(b)

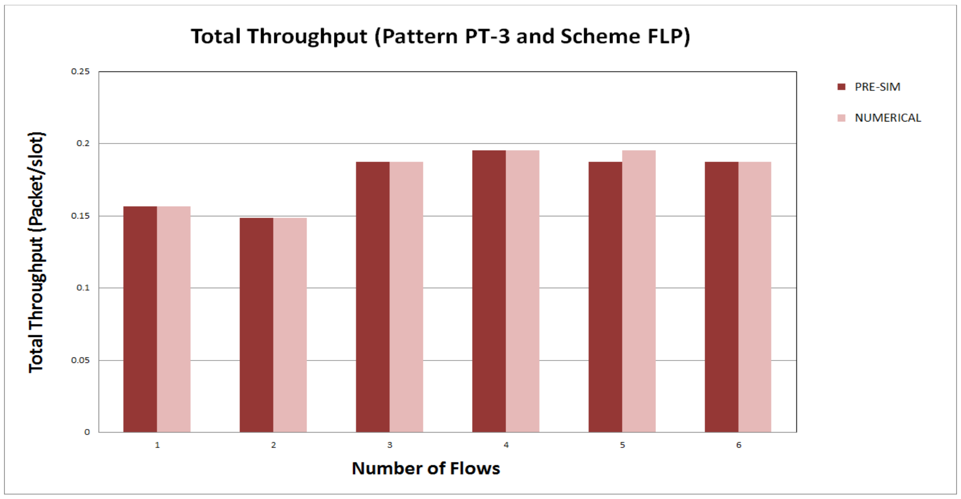


(c)

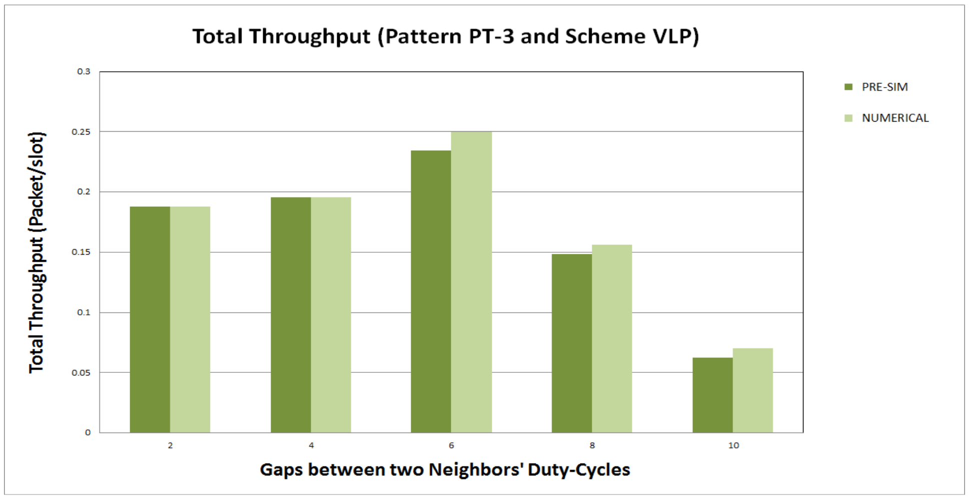
Figure 3.18. Throughput in 3x3 grid with pattern PT-2



(a)

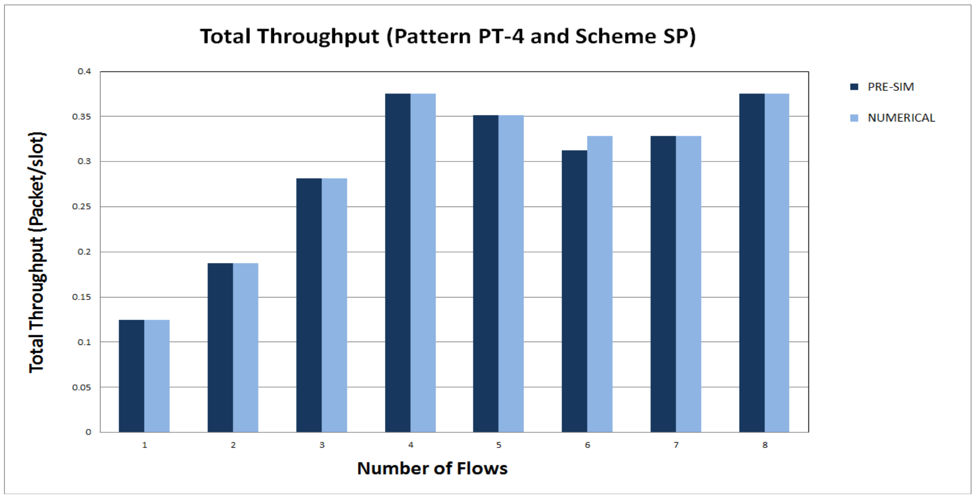


(b)

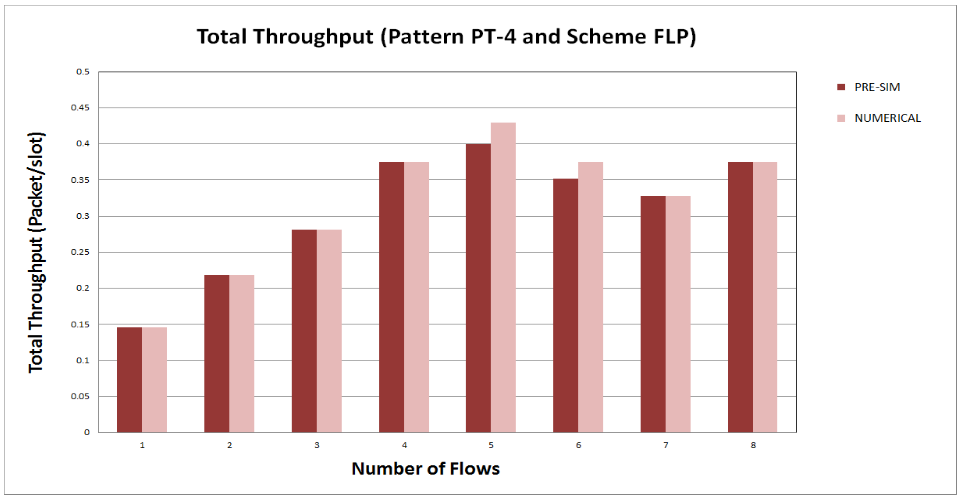


(c)

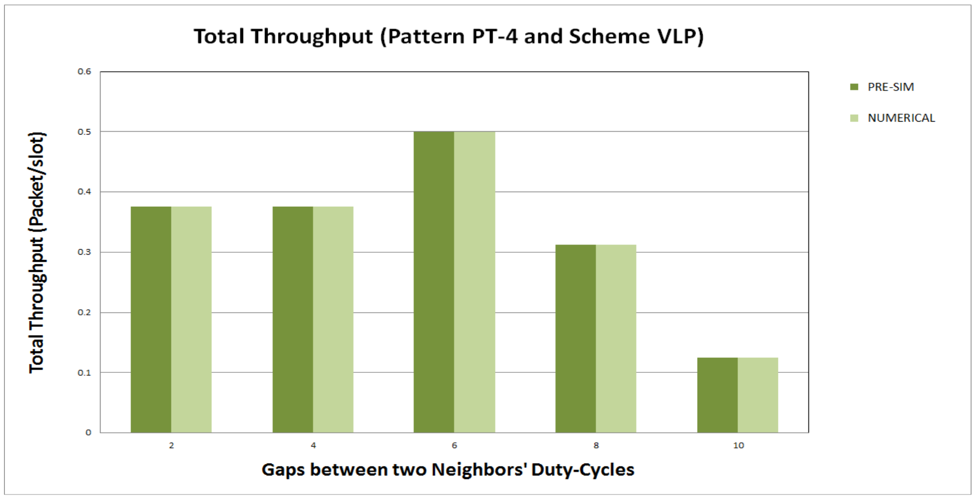
Figure 3.19. Throughput in 3x3 grid with pattern PT-3



(a)



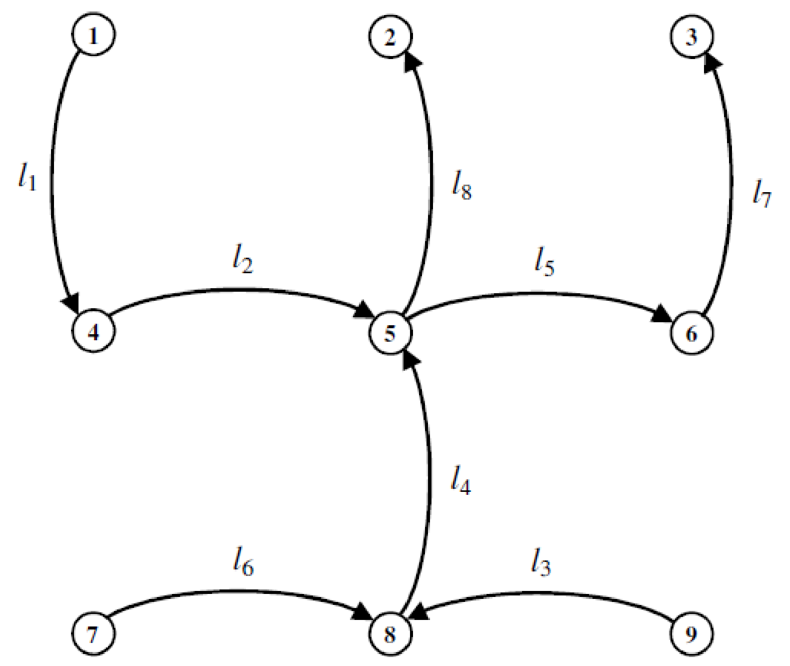
(b)



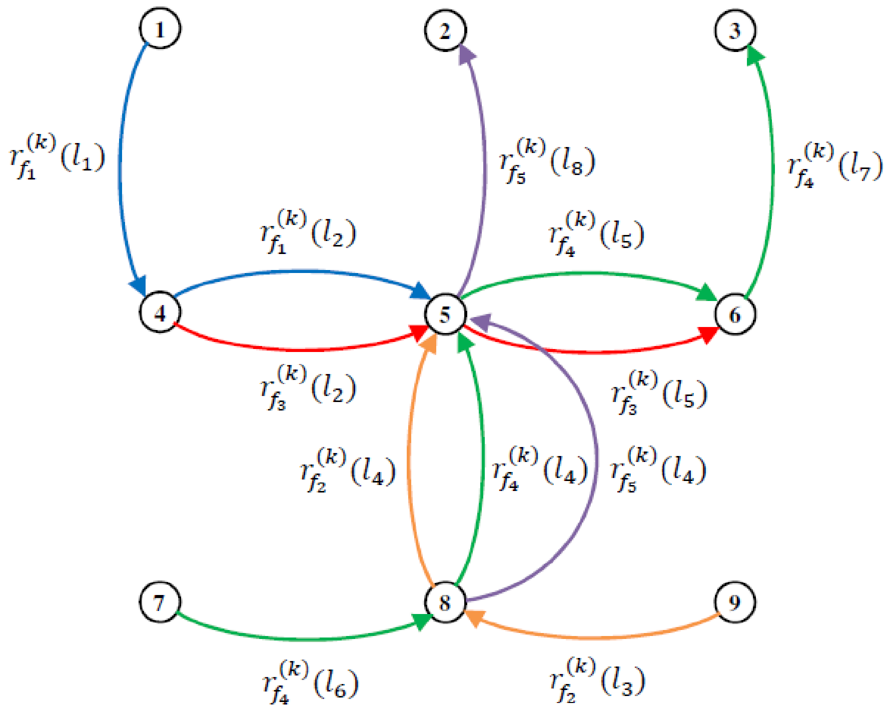
(c)

Figure 3.20. Throughput in 4x4 grid with pattern PT-4

3.3.3 SCHEDULING COMPARISON



(a) Subset of 3x3 grid links involving only links traversed by flows



(b) Multi-hop flows of pattern PT-2

Figure 3.21. Topology and flows used in the comparison

To further evaluate the performance of our scheduling algorithm, we compare our TDMA scheduling algorithm with one of those mentioned in the “Related Work” chapter, which is the work of Gronkvist [13]. We select this work for performance comparison because it is most related to our work in the following main aspects: (1) the work considers multi-hop networks, which are more popular than single-hop ones; (2) it uses a STDMA scheme for spatial resuse of timeslots; (3) for reducing the averge delay, its algorithm takes traffic load into consideration in the STDMA scheme.

For the comparison, we select the network with link topology presented in Figure 3.21.a and flow pattern presented in Figure 3.21.b. The link topology is a subset of the 3x3 grid links involving only links traversed by flows, which is described in the previous section. In addition, the flow pattern is also originated from pattern PT-2 in the previous section, which is described by the table of flow paths. We select this network for comparison because it is most representive for both of the algorithms by the following reasons: (1) it is a multi-hop network; (2) there are multiple contention domains in the network so we can apply the spatial reuse of timeslots; (3) to be more general, the network intentionally has multiple flows traversing on a single link.

For easier description, we use our notations in most of the time, and convert those in [13] to our own when we need to introduce new definitions from this work for consistency purposes. Since our algorithm can be used for duty-cycled networks and the work [13] is used for networks without duty-cycling, we need to “downgrade” it to that without duty-cycling before the comparison. Hence, there is a single stage, i.e., $K = 1$, in our algorithm but we still keep a general stage k in Figure 3.21.b for consistency with notations described so far. Note that in general, a single stage means all the nodes have to be ON/OFF at the same time and therefore there is one stage including the ON periods of all the active nodes in this comparison.

There are 8 directional links, from l_1 to l_8 in Figure 3.21.a. Also, there are five flows with flow paths in Figure 3.21.b, flow f_1 in blue from source 1 to sink 5, flow f_2 in orange from source 9 to sink 5, flow f_3 in red from source 4 to sink 6, flow f_4 in green from source 7 to sink 3, and flow f_5 in purple from source 8 to sink 2. Note that the algorithm in [13] does not have the concept of sub-flows like in our work and only needs average traffic loads on links for schedule construction. Therefore, we need to run our scheduling algorithm on the network to get all the rates $r_f^{(k)}(l)$ of flow f over link l, from which we then calculate the average traffic load, λ_{ij} on link (i, j), or λ_l on link l for consistency with our work.

t	$r_{f_1}^{(k)}(l_1)$	$r_{f_1}^{(k)}(l_2)$	$r_{f_2}^{(k)}(l_3)$	$r_{f_2}^{(k)}(l_4)$	$r_{f_3}^{(k)}(l_2)$	$r_{f_3}^{(k)}(l_5)$	$r_{f_4}^{(k)}(l_6)$	$r_{f_4}^{(k)}(l_4)$	$r_{f_4}^{(k)}(l_5)$	$r_{f_4}^{(k)}(l_7)$	$r_{f_5}^{(k)}(l_4)$	$r_{f_5}^{(k)}(l_8)$
1	○	●	○	○	○	○	○	○	○	○	○	○
2	○	○	●	○	○	○	○	○	○	○	○	○
3	●	○	○	●	○	○	○	○	○	○	○	○
4	○	○	○	○	●	○	○	○	○	○	○	○
5	○	○	○	○	○	●	○	○	○	○	○	○
6	○	○	○	○	○	○	○	●	○	○	○	○
7	○	○	○	○	○	○	○	○	●	○	○	○
8	○	○	○	○	○	○	●	○	○	●	○	○
9	○	○	○	○	○	○	○	○	○	○	●	○
10	○	○	○	○	○	○	○	○	○	○	○	●

Figure 3.22. Schedule is constructed by our MMF algorithm
(black circles mean sub-flows are scheduled)

For the interference, we assume both the algorithms use the same logical model of link interference used in the work by Li [26], i.e., two sub-flows that are ON in the same slot, contend with each other if either the transmitter end or the receiver end of one sub-flow is within the transmission range of the transmitter end or the receiver end of the other sub-flow. The definition of a contention graph is also described earlier in Example 1.1. By running our MMF scheduling algorithm on the network, we construct the schedule for all the sub-flows in each timeslot $\boldsymbol{t}$ shown in Figure 3.22. With the interference model, for example, sub-flow $r_{f_1}^{(k)}(l_2)$ cannot be scheduled simultaneously with sub-flows $r_{f_1}^{(k)}(l_1)$ and/or $r_{f_2}^{(k)}(l_3)$ in timeslot 1.Hence, we have the rates of all the flows over links as follows.

$$
\begin{aligned}
& r_{f_1}^{(k)}(l_1) = r_{f_1}^{(k)}(l_2) = r_{f_2}^{(k)}(l_3) = r_{f_2}^{(k)}(l_4) = \\
& r_{f_3}^{(k)}(l_2) = r_{f_3}^{(k)}(l_5) = \\
& r_{f_4}^{(k)}(l_6) = r_{f_4}^{(k)}(l_4) = r_{f_4}^{(k)}(l_5) = r_{f_4}^{(k)}(l_7) = \\
& r_{f_5}^{(k)}(l_4) = r_{f_5}^{(k)}(l_8) = \tfrac{1}{10} = 0.1 \text{ packets/slot} \qquad (3.19)
\end{aligned}
$$

From (3.19) and flow paths in Figure 3.21.b, we can calculate all the following traffic loads on links.

$$
\begin{aligned}
& \lambda_{l_1} = r; \lambda_{l_2} = 2r; \lambda_{l_3} = r; \lambda_{l_4} = 3r;\ \lambda_{l_5} = 2r; \lambda_{l_6} = r; \\
& \lambda_{l_7} = r; \lambda_{l_8} = r; \qquad \text{where } r = 0.1 \text{ (packets/slot)} \qquad (3.20)
\end{aligned}
$$

From the network, we have.

$$\mathrm{N} = 9 \text{ (nodes)};\ \mathrm{M} = 8 \text{ (links)} \qquad (3.21)$$

Hence, we have the total traffic load of five flows in the network as follows.

$$\lambda = 5r = 0.5 \text{ (packets/slot)} \qquad (3.22)$$

From [13], we have relative traffic Λ_l on link l defined as follows:

$$\Lambda_l = \lambda_l / (\lambda / \mathrm{N}(\mathrm{N}-1))$$

Hence, we can calculate the following.

$$\Lambda_{l_1} = 14.4; \Lambda_{l_2} = 28.8;\ \Lambda_{l_3} = 14.4; \Lambda_{l_4} = 43.2; \Lambda_{l_5} = 28.8;$$
$$\Lambda_{l_6} = 14.4; \Lambda_{l_7} = 14.4;\ \Lambda_{l_8} = 14.4; \qquad (3.23)$$

From [13] and with (3.21), (3.22) and (3.23), we have the average relative traffic in the network, which is defined as follows.

$$\begin{aligned}\bar{\Lambda} &= \frac{1}{M}\sum_{\forall l} \Lambda_l \\ &= (\Lambda_{l_1} + \Lambda_{l_2} + \Lambda_{l_3} + \Lambda_{l_4} + \Lambda_{l_5} + \Lambda_{l_6} + \Lambda_{l_7} + \Lambda_{l_8})/M \\ &= 21.6 \qquad (3.24)\end{aligned}$$

From [13], we have link l is guaranteed the following number of slots.

$$\left\lceil \frac{\Lambda_l}{\bar{\Lambda}} \right\rceil \qquad (3.25)$$

From (3.24) and (3.25), we have the following.

$$\left\lceil \frac{\Lambda_{l_1}}{\bar{\Lambda}} \right\rceil = 1; \left\lceil \frac{\Lambda_{l_2}}{\bar{\Lambda}} \right\rceil = 2; \left\lceil \frac{\Lambda_{l_3}}{\bar{\Lambda}} \right\rceil = 1; \left\lceil \frac{\Lambda_{l_4}}{\bar{\Lambda}} \right\rceil = 2; \left\lceil \frac{\Lambda_{l_5}}{\bar{\Lambda}} \right\rceil = 2;$$
$$\left\lceil \frac{\Lambda_{l_6}}{\bar{\Lambda}} \right\rceil = 1; \left\lceil \frac{\Lambda_{l_7}}{\bar{\Lambda}} \right\rceil = 1; \left\lceil \frac{\Lambda_{l_8}}{\bar{\Lambda}} \right\rceil = 1; \qquad (3.26)$$

Figure 3.23 shows the steps (in timeslots) run by the Gronkvist's algorithm on the network on Figure 3.21 in order to construct a schedule, in which $\boldsymbol{\Lambda_l \tau_l}$ is the link priority of link l used in the algorithm, where $\boldsymbol{\tau_l}$ is the number of timeslots since the link was previously allocated a timeslot. Also, **List A** is the set of links that still has not been given all their guaranteed timeslots. Note that the algorithm stops when the list is empty. Finally, **Schedule** indicates which links are to be scheduled in each timeslot $\boldsymbol{t}$.

t	$\Lambda_{l_1}\tau_{l_1}$	$\Lambda_{l_2}\tau_{l_2}$	$\Lambda_{l_3}\tau_{l_3}$	$\Lambda_{l_4}\tau_{l_4}$	$\Lambda_{l_5}\tau_{l_5}$	$\Lambda_{l_6}\tau_{l_6}$	$\Lambda_{l_7}\tau_{l_7}$	$\Lambda_{l_8}\tau_{l_8}$	**List A**	**Schedule**
0	0	0	0	0	0	0	0	0	$l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8$	-
1	0	0	0	0	0	0	0	0	$l_2, l_4, l_5, l_6, l_7, l_8$	l_1, l_3
2	-	28.8	-	**43.2**	28.8	14.4	14.4	14.4	$l_2, l_4, l_5, l_6, l_7, l_8$	l_4
3	-	**57.6**	-	0	57.6	28.8	28.8	28.8	$l_2, l_4, l_5, l_6, l_7, l_8$	l_2
4	-	0	-	43.2	**86.4**	43.2	43.2	43.2	$l_2, l_4, l_5, l_6, l_7, l_8$	l_5
5	-	28.8	-	**86.4**	0	57.6	57.6	57.6	l_2, l_5, l_6, l_7, l_8	l_4
6	-	57.6	**-**	-	28.8	**72**	**72**	72	l_2, l_5, l_8	l_6, l_7
7	-	**86.4**	-	-	57.6	-	-	86.4	l_5, l_8	l_2
8	-	-	-	-	86.4	**-**	-	**100.8**	l_5	l_8
9	-	-	-	-	**115.2**	-	**-**	-	ϕ	l_5

Figure 3.23. Steps in the Gronkvist's algorithm to construct a schedule

From (3.20), we rewrite in more details the traffic loads λ_l scheduled on link l by our scheduling algorithm.

$$\lambda_{l_1} = 0.1; \lambda_{l_2} = 0.2; \lambda_{l_3} = 0.1; \lambda_{l_4} = 0.3;\ \lambda_{l_5} = 0.2;$$

$$\lambda_{l_6} = 0.1; \lambda_{l_7} = 0.2; \lambda_{l_8} = 0.1 \text{ (packets/slot)} \qquad (3.27)$$

From (3.22), we also rewrite in more details the total throughput λ made by our algorithm including traffic loads of five flows f_1, f_2, f_3, f_4 and f_5.

$$\lambda = \lambda_{l_1} + \lambda_{l_3} + \lambda_{l_6} + \lambda_{l_7} + \lambda_{l_8} = 0.5 \text{ (packets/slot)} \qquad (3.28)$$

From the results in Figure 3.23, we have the traffic loads λ'_l scheduled on link l by the Gronkvist's algorithm.

$$\lambda'_{l_1} = 1/9; \lambda'_{l_2} = 2/9; \lambda'_{l_3} = 1/9; \lambda'_{l_4} = 2/9; \lambda'_{l_5} = 2/9;$$

$$\lambda'_{l_6} = 1/9; \lambda'_{l_7} = 1/9; \lambda'_{l_8} = 1/9 \text{ (packets/slot)} \tag{3.29}$$

From (3.29), we also have the total throughput λ' made by the Gronkvist's algorithm including traffic loads of five flows f_1, f_2, f_3, f_4 and f_5.

$$\lambda' = \lambda'_{l_2} + \lambda'_{l_4} = 4/9 = 0.4444 \text{ (packets/slot)} \tag{3.30}$$

From the results of the two schedules by our algorithm and the Gronkvist's algorithm, we have the following observations.

(1) Our schedules provide the total throughput, i.e., λ = **0.5** (packets/slot), greater than the Gronkvist's algorithm, i.e., λ'= **0.4444** (packets/slot);

(2) The Gronkvist's schedule does not perfectly fit to the actual traffic loads demands on the links, e.g., traffic load demand on link l_4 is **0.3** (packets/slot) but this link is allocated just **0.2222** (packets/slot).

The main reason for the poor performance of Gronkvist's algorithm results is the relative and average assumption used in the definition of relative traffic Λ_l and hence the average relative traffic $\overline{\Lambda}$. Consequently, the guaranteed number of slots in the schedule for a link $\left\lceil \frac{\Lambda_l}{\overline{\Lambda}} \right\rceil$ does not always fit the actual traffic load.

3.4 CHAPTER CONCLUSIONS

It is widely accepted that wireless medium scheduling is a hard problem even if several simplifications are performed. In this study we opted for situations where the topology is fixed and known but the actual link dynamics are dependent on the periodic DC of the nodes. The time-varying nature of the underlying communication graph compounds the complexity of determining a single periodic TDMA-like schedule. We attempted to solve the scheduling problem by performing an off-line simulation-based process of the system for which we have strong evidence that, due to the periodicity of all input factors, a similar periodicity should be exhibited by the steady state. Part of this periodic steady state behavior is the transmission decisions. We excise the transmissions scheduled during the simulation to create a template which can then be subsequently downloaded to the nodes.

We evaluated our approach based on a number of regular topologies and duty cycling behaviors. Even though we restricted the evaluation to homogeneous (same duty cycling period) nodes, the process outlined here can be extended to situations with different duty cycle characteristics albeit at the cost of computing the combined (based on the least common multiple of the different cycles) repeated pattern across all nodes. Furthermore, our technique does not need to comply to rate allocation derived under lexmax/maxmin criteria. Other objectives can be used instead. The rate allocations, regardless of how they are calculated, as long as they are feasible, simply act as a target for the simulation-based process.

The most evident shortcoming of the excision process is that it ignores the state of the queues whose influence is only accounted for indirectly by excising scheduling patterns that are multiples of the duty cycle period. The reader can question whether the state of queues ought to have been a component of the state comparison/matching. We have, so far, avoided matching the queue state in addition to the transmissions because it is unclear whether matching the exact number of packets in the queue is necessary (or just that there are some) and for the added complexity that this would bring about (the buffers can be as many as the flows times the number of links in the network). A possible future direction is to elaborate the inter-dependency between queues and transmissions and how this is manifested in the schedule template excision. In the next chapter, we introduce a potent extension, which is the inclusion of NC (and scheduling decisions about when to code and when not). This is to be a means to improve the throughput when, due to duty cycling, throughput is “lost” because nodes are in OFF state.

CHAPTER 4
COMBINING DUTY-CYCLING AND NETWORK CODING

4.1 PROBLEM FORMULATION

This chapter provides the NC extension of the schedule construction process, which is detailed in Chapter 3. For reasons of consistency with Chapter 3, we demonstrate how the rate optimization formulation can be restated to take into account NC. In this thesis we use a particular form of pairwise NC, i.e., at each node, flows can be coded only in a pairwise fashion. This places a restriction on possible coding options, compared e.g., to the early results of section 2.6 in Chapter 2, and in particular in this chapter we assume that which pairs are to be coded together at each node is given. Additionally, a discussion about other coding options will be presented in Chapter 5. We forewarn the reader that the notion of max-min fairness we used in Chapter 3, when applied to networks where NC is added, does not lead to the same results, hence even though we insist on a max-min formulation for NC, the per-flow allocations will be different, hindering a flow-by-flow comparison. Instead, when evaluating the performance of adding NC, the comparison is going to be on the basis of total throughput. Finally, whereas most of the work in this chapter are extensions/adaptations of the techniques of Chapter 3, an additional technique unique to NC, that of *delay coding* will be introduced in section 4.2. Its purpose is to ensure that, as pointed in the conclusions of Chapter 2, there exist sufficient data packets available to the nodes in order to perform NC. Essentially this chapter provides the techniques to create STDMA schedules which not only determine when a node transmits but also when it can transmit network coded packets.

4.1.1 EXTENDING THE MAX-MIN RATE COMPUTATION TO THE NC CASE

In a manner similar to that of section 3.1 we can re-formulate the relevant optimization problems. For example, Figure 4.1 shows the $LP(r_f, NC)$ formulation, which is used with the WF algorithm with NC in Figure 4.2 to compute the numerical rates of all the flows $f \in \mathcal{F}$. Obviously, the formulation is different than that without NC, i.e., $LP(r_f)$, in terms of the constraints, since it is assumed that there exist any predetermined coding combination of two flows at a node, which need to be described as follows (using the notation introduced in section 1.3).

Constraint (4.2) specifies flow conservation constraints which require the rate of each flow summed-up across stages be conserved at a node n. Note the inclusion of the combined rates $r_{f_1,f_2}^{(k)}(l_1, l_2)$ for pairs of coded flows. Constraint (4.3) specifies, for each stage, the link capacity restrictions that apply to contending sub-flows which are active in that stage. Note that the coefficient of (1/2) before a coding combination of two flows in the constraints is to prevent double counting of the same combination of the two flows, since effectively a pair-wise combination of flows halves the number of transmissions needed for the packets that are coded together (keeping in mind that there are also packets that are not coded and are accounted for separately from the coded ones).

$$max \sum_{f \in \mathcal{F}} r_f \qquad (4.1)$$

Subject to

$$\sum_{\substack{k \in \mathcal{K};\\ l_1 \in \mathcal{L}:\\ o(l_1)=n}} r_{f_1}^{(k)}(l_1) + \sum_{\substack{k \in \mathcal{K};\\ f_2 \in \mathcal{F};\\ l_1, l_2 \in \mathcal{L}:\\ o(l_1)=n;\\ o(l_2)=n;}} r_{f_1,f_2}^{(k)}(l_1, l_2) - \sum_{\substack{k \in \mathcal{K};\\ l_1' \in \mathcal{L}:\\ d(l_1')=n}} r_{f_1}^{(k)}(l_1') -$$

$$- \sum_{\substack{k\in\mathcal{K};\\ f_2'\in\mathcal{F};\\ l_1',l_2'\in\mathcal{L}:\\ d(l_1')=n;\\ o(l_1')=o(l_2');}} r_{f_1,f_2'}^{(k)}(l_1',l_2') = \begin{cases} r_{f_1} & , n = o(f_1) \\ -r_{f_1} & , n = d(f_1) \\ 0 & , else \end{cases} \qquad \forall n \in \mathcal{N}, f_1 \in \mathcal{F} \tag{4.2}$$

$$\sum_{\substack{f\in\mathcal{F}\\ l\in\mathcal{L}\\ r_f^{(k)}(l)\in q}} r_f^{(k)}(l) + \frac{1}{2}\sum_{\substack{f_1,f_2\in\mathcal{F}\\ l_1,l_2\in\mathcal{L}:\, o(l_1)=o(l_2)\\ r_{f_1,f_2}^{(k)}(l_1,l_2)\in q}} r_{f_1,f_2}^{(k)}(l_1,l_2) \le r.\mathcal{T}^{(k)} \quad \forall k \in \mathcal{K}, \forall q \in Q \tag{4.3}$$

$$\sum_{k\in\mathcal{K}} r_{f_1,f_2}^{(k)}(l_1,l_2) \le \sum_{k\in\mathcal{K}} r_{f_2}^{(k)}(\overline{l_1}) + \sum_{k\in\mathcal{K}} r_{f_2,f_3}^{(k)}(\overline{l_1},l_3)$$

$$f_1,f_2,f_3 \in \mathcal{F}: f_1 \ne f_2, f_2 \ne f_3;\ l_1,l_2,\overline{l_1},l_3 \in \mathcal{L}: o(l_2) = o(l_1),\ o(l_3) = o(\overline{l_1}) \tag{4.4}$$

$$r_{f_1,f_2}^{(k)}(l_1,l_2) = \begin{cases} 0, & o(l_1) = o(f_1) \\ 0, & o(l_1) = d(f_1) \\ r_{f_2,f_1}^{(k)}(l_2,l_1), & else \end{cases} \qquad \forall f_1,f_2 \in \mathcal{F},\ l_1,l_2 \in \mathcal{L} \tag{4.5}$$

$$r_f = R_f \qquad S_f = FIXED, \forall f \in \mathcal{F} \tag{4.6}$$

$$r_f,\ r_f^{(k)}(l),\ r_{f_1,f_2}^{(k)}(l_1,l_2) \ge 0 \qquad \forall f,f_1,f_2 \in \mathcal{F},\ l,l_1,l_2 \in \mathcal{L},\ k \in \mathcal{K} \tag{4.7}$$

Figure 4.1. Formulation $LP(r_f, NC)$

Constraints (4.4) state that the rate coded from a pair of flows at a node $o(l_1)$ (if it exists) over active link l_1 cannot be larger than that from one of the two flows traversing over inverse link $\overline{l_1}$, which is generally created from two components, the non-coded and the coded, received at the node $o(l_1)$ across stages. Constraints (4.5) on the variable $r_{f_1,f_2}^{(k)}(l_1,l_2)$ specify that the coding from any two flows f_1 and f_2 at a node $n = o(l_1) = o(l_2)$ is not allowed if the node n is the source or the destination of flow $f_1 \in \mathcal{F}$ during each stage $\mathcal{T}^{(k)}$, and the constraints also indicate the symmetry of coding the two flows.

The solution process is similar to the WF of Chapter 3. For the sake of completeness we provided it here. The WF algorithm with NC in Figure 4.2 is the same as that without NC except $LP(r_f)$ is replaced by $LP(r_f, NC)$. In other words, the WF algorithm with NC works exactly the same as that without NC because their correctness is independent of the predetermined coding combinations of two flows.

```
Input: Formulation $LP(r_f, NC)$ with $f \in \mathcal{F}$;
Output: Numerical max-min rates $R_f$ of all the multi-hop flows $f \in \mathcal{F}$;
Begin
01.  Set $S_f = UNFXD$ and $R_f = 0$ for all the flows $f \in \mathcal{F}$;
02.  Repeat
03.      Select flow $f$ with $S_f = UNFXD$ in a round robin manner;
04.      Increase $R_f$ by a minuscule value $\varepsilon$, i.e., $R_f = R_f + \varepsilon$;
05.      Submit $LP(r_f, NC)$ to LP solver;
06.      If  LP solver indicates $LP(r_f, NC)$ is infeasible  Then
07.          Restore $R_f$ to the previous value, i.e., $R_f = R_f - \varepsilon$;
08.          Change $S_f = FIXED$;
09.      End If
10.  Until $S_f = FIXED$ for all the flows $f \in \mathcal{F}$;
End
```

Figure 4.2. The WF algorithm with NC

We can similarly restate the MP algorithm with NC to accelerate the computation process the WF algorithm with NC.

```
Input: Formulations $LP\ (f, NC)$ and $LP\ (f, R_{min}, NC)$ with $f \in \mathcal{F}$;
Output: Numerical max-min rates $R_f$ of all the multi-hop flows $f \in \mathcal{F}$;
Begin
01.  Set $S_f = UNFXD$ for all the flows $f \in \mathcal{F}$;
```

```
02. Repeat
03.     Select flow $f \in \mathcal{F}$ with $S_f = UNFXD$;
04.     Submit $LP\ (f, NC)$ to LP solver;
05.     Store rate $R_{min}$ of flow $f$ returned from LP solver;
06.     For each flow $f \in \mathcal{F}$ with $S_f = UNFXD$ Do
07.         Submit $LP\ (f, R_{min}, NC)$ to LP solver;
08.         Get rate $r_f$ of flow $f$ from LP solver;
09.         If $r_f = R_{min}$ Then
10.             Set $S_f = FIXED$ for flow $f$;
11.         End If
12.     End For
13. Until $S_f = FIXED$ for all the flows $f \in \mathcal{F}$;
End
```

Figure 4.3. The MP algorithm with NC

The MP algorithm with NC in Figure 4.3 is associated with $LP(g, NC)$ in Figure 4.4 and $LP\ (g, R_{min}, NC)$ in Figure 4.5 to compute the max-min rates for multi-hop flows in a DC-WSN. The MP algorithm with NC is exactly the same as that without NC, in which $LP(g)$ and $LP\ (g, R_{min})$ are replaced by $LP(g, NC)$ and $LP\ (g, R_{min}, NC)$.

$$max\ (r_g) \qquad\qquad g \in \mathcal{F}, S_g = UNFXD \qquad (4.8)$$

Subject to

$$\sum_{\substack{k\in\mathcal{K};\\ l_1\in\mathcal{L}:\\ o(l_1)=n}} r_{f_1}^{(k)}(l_1) + \sum_{\substack{k\in\mathcal{K};\\ f_2\in\mathcal{F};\\ l_1, l_2\in\mathcal{L}:\\ o(l_1)=n;\\ o(l_2)=n;}} r_{f_1,f_2}^{(k)}(l_1, l_2) - \sum_{\substack{k\in\mathcal{K};\\ l_1'\in\mathcal{L}:\\ d(l_1')=n}} r_{f_1}^{(k)}(l_1') -$$

$$- \sum_{\substack{k\in\mathcal{K};\\ f_2'\in\mathcal{F};\\ l_1',l_2'\in\mathcal{L}:\\ d(l_1')=n;\\ o(l_1')=o(l_2');}} r_{f_1,f_2'}^{(k)}(l_1',l_2') = \begin{cases} r_{f_1} & , n = o(f_1) \\ -r_{f_1} & , n = d(f_1) \\ 0 & , else \end{cases} \qquad \forall n \in \mathcal{N}, f_1 \in \mathcal{F} \qquad (4.9)$$

$$\sum_{\substack{f\in\mathcal{F}\\ l\in\mathcal{L}\\ r_f^{(k)}(l)\in q}} r_f^{(k)}(l) + \frac{1}{2}\sum_{\substack{f_1,f_2\in\mathcal{F}\\ l_1,l_2\in\mathcal{L}:\, o(l_1)=o(l_2)\\ r_{f_1,f_2}^{(k)}(l_1,l_2)\in q}} r_{f_1,f_2}^{(k)}(l_1,l_2) \le r.\mathcal{T}^{(k)} \ \forall k \in \mathcal{K}, \forall q \in \mathcal{Q} \qquad (4.10)$$

$$\sum_{k\in\mathcal{K}} r_{f_1,f_2}^{(k)}(l_1,l_2) \le \sum_{k\in\mathcal{K}} r_{f_2}^{(k)}(\overline{l_1}) + \sum_{k\in\mathcal{K}} r_{f_2,f_3}^{(k)}(\overline{l_1},l_3)$$

$$f_1,f_2,f_3 \in \mathcal{F}: f_1 \neq f_2, f_2 \neq f_3;\ l_1,l_2,\overline{l_1},l_3 \in \mathcal{L}: o(l_2) = o(l_1),\ o(l_3) = o(\overline{l_1}) \qquad (4.11)$$

$$r_{f_1,f_2}^{(k)}(l_1,l_2) = \begin{cases} 0\,, & o(l_1) = o(f_1) \\ 0\,, & o(l_1) = d(f_1) \\ r_{f_2,f_1}^{(k)}(l_2,l_1)\,, & else \end{cases} \qquad \forall f_1,f_2 \in \mathcal{F},\ l_1,l_2 \in \mathcal{L} \qquad (4.12)$$

$$r_f = R_f \qquad S_f = FIXED, \forall f \in \mathcal{F} \qquad (4.13)$$

$$r_f = r_g \qquad S_f = UNFXD, \forall f \in \mathcal{F} \qquad (4.14)$$

$$r_f,\ r_f^{(k)}(l),\ r_{f_1,f_2}^{(k)}(l_1,l_2) \ge 0 \qquad \forall f,f_1,f_2 \in \mathcal{F},\ l,l_1,l_2 \in \mathcal{L},\ k \in \mathcal{K} \qquad (4.15)$$

Figure 4.4. Formulation $LP(g, NC)$

$$max\ (r_g) \qquad g \in \mathcal{F}, S_g = UNFXD \quad (4.16)$$

Subject to

$$\sum_{\substack{k\in\mathcal{K};\\ l_1\in\mathcal{L}:\\ o(l_1)=n}} r_{f_1}^{(k)}(l_1) + \sum_{\substack{k\in\mathcal{K};\\ f_2\in\mathcal{F};\\ l_1,l_2\in\mathcal{L}:\\ o(l_1)=n;\\ o(l_2)=n;}} r_{f_1,f_2}^{(k)}(l_1,l_2) - \sum_{\substack{k\in\mathcal{K};\\ l_1'\in\mathcal{L}:\\ d(l_1')=n}} r_{f_1}^{(k)}(l_1') -$$

$$-\sum_{\substack{k\in\mathcal{K};\\ f_2'\in\mathcal{F};\\ l_1',l_2'\in\mathcal{L}:\\ d(l_1')=n;\\ o(l_1')=o(l_2');}} r_{f_1,f_2'}^{(k)}(l_1',l_2') = \begin{cases} r_{f_1} & , n=o(f_1) \\ -r_{f_1} & , n=d(f_1) \\ 0 & , else \end{cases} \qquad \forall n\in\mathcal{N}, f_1\in\mathcal{F} \quad (4.17)$$

$$\sum_{\substack{f\in\mathcal{F}\\ l\in\mathcal{L}\\ r_f^{(k)}(l)\in q}} r_f^{(k)}(l) + \frac{1}{2}\sum_{\substack{f_1,f_2\in\mathcal{F}\\ l_1,l_2\in\mathcal{L}:\, o(l_1)=o(l_2)\\ r_{f_1,f_2}^{(k)}(l_1,l_2)\in q}} r_{f_1,f_2}^{(k)}(l_1,l_2) \le r.\mathcal{T}^{(k)} \ \forall k\in\mathcal{K}, \forall q\in\mathcal{Q} \quad (4.18)$$

$$\sum_{k\in\mathcal{K}} r_{f_1,f_2}^{(k)}(l_1,l_2) \le \sum_{k\in\mathcal{K}} r_{f_2}^{(k)}(\overline{l_1}) + \sum_{k\in\mathcal{K}} r_{f_2,f_3}^{(k)}(\overline{l_1},l_3)$$

$$f_1,f_2,f_3\in\mathcal{F}: f_1\ne f_2, f_2\ne f_3;\ l_1,l_2,\overline{l_1},l_3\in\mathcal{L}: o(l_2)=o(l_1),\ o(l_3)=o(\overline{l_1}) \quad (4.19)$$

$$r_{f_1,f_2}^{(k)}(l_1,l_2) = \begin{cases} 0, & o(l_1)=o(f_1) \\ 0, & o(l_1)=d(f_1) \\ r_{f_2,f_1}^{(k)}(l_2,l_1), & else \end{cases} \qquad \forall f_1,f_2\in\mathcal{F},\ l_1,l_2\in\mathcal{L} \quad (4.20)$$

$$r_f = R_f \qquad S_f = FIXED, \forall f\in\mathcal{F} \quad (4.21)$$

$$r_f \ge R_{min} \qquad S_f = UNFXD, \forall f\in\mathcal{F} \quad (4.22)$$

$$r_f,\ r_f^{(k)}(l),\ r_{f_1,f_2}^{(k)}(l_1,l_2) \ge 0 \qquad \forall f,f_1,f_2\in\mathcal{F},\ l,l_1,l_2\in\mathcal{L},\ k\in\mathcal{K} \quad (4.23)$$

Figure 4.5. Formulation $LP(g, R_{min}, NC)$

Example 4.1. To illustrate how the MP algorithm with NC calculates the per-flow rates, we first consider a simple example in Figures 4.6 and 4.7, in which Figure 4.6 (or Figure 4.7) includes all possible maximal cliques without NC (or with NC) that created from contending sub-flows in each stage. Then we walk through the algorithm step-by-step, first running it without NC and next with NC.

The duty-cycled network has four nodes $n\in\mathcal{N}=\{1,2,3,4\}$, six directed wireless links in the link set $\mathcal{L}=\{l_1,l_2,l_3,l_4,l_5,l_6\}$ shown in Figure 4.6.a.1, and three flows $f_1,f_2,f_3\in\mathcal{F}$ traversing, respectively, the paths $\{l_1,l_2,l_3\}$, $\{l_4,l_5\}$ and $\{l_6\}$ indicated in Figure 4.6.b. The DC-configurations, i.e., $\langle\phi_n,\alpha_n,T_n\rangle$, of the nodes is as shown in Figure 4.6.a.2, in which period T is divided into two stages

$\mathcal{T}^{(k)}$, in units of slots, with $k \in \mathcal{K} = \{1,2\}$, such that in each of these stages the ON/OFF state of the nodes is unchanged and there exists at least an active link (both its ends are ON).

In Figure 4.6.b, during each stage k, flow f_1 without NC is represented by three sub-flow components $r_{f_1}^{(k)}(l_1)$, $r_{f_1}^{(k)}(l_2)$ and $r_{f_1}^{(k)}(l_3)$, flow f_2 without NC is represented by two sub-flow components $r_{f_2}^{(k)}(l_4)$ and $r_{f_2}^{(k)}(l_5)$, and flow f_3 without NC is represented by one sub-flow component $r_{f_3}^{(k)}(l_6)$. Hence, the maximal cliques of the sub-flow contention graph (CG) which is made up from all sub-flows competing in each stage, i.e., $\mathcal{T}^{(1)}$ and $\mathcal{T}^{(2)}$, are shown in Figures 4.6.c.1 and 4.6.c.2, respectively.

In Figure 4.7.b, during each stage k, flow f_1 with NC is represented by four sub-flow components $r_{f_1}^{(k)}(l_1)$, $r_{f_1}^{(k)}(l_2)$, $r_{f_1}^{(k)}(l_3)$ and $r_{f_1,f_2}^{(k)}(l_3, l_5)$, flow f_2 with NC is represented by three sub-flow components $r_{f_2}^{(k)}(l_4)$, $r_{f_2}^{(k)}(l_5)$, and $r_{f_1,f_2}^{(k)}(l_3, l_5)$, and flow f_3 with NC is still represented by one sub-flow component $r_{f_3}^{(k)}(l_6)$. Hence, the maximal cliques of the sub-flow contention graph (CG) which is made up from all sub-flows competing in each stage, i.e., $\mathcal{T}^{(1)}$ (from slots TS_4 to TS_{12}) and $\mathcal{T}^{(2)}$ (from slots TS_{12} to TS_{16}), are shown in Figures 4.7.c.1 and 4.7.c.2, respectively. Note that coded sub-flows, e.g., $r_{f_1,f_2}^{(k)}(l_3, l_5)$, are not manually specified but they are created algorithmically from non-coded sub-flows, e.g., $r_{f_1}^{(k)}(l_3)$ and $r_{f_2}^{(k)}(l_5)$, at intermediate nodes under the certain conditions that will be described later in Section 4.2.1. Note that in case with more than one possibility of pairing non-coded sub-flows at an intermediate node, we use the heuristic approach proposed in Chapter 5.

When the MP with NC runs on the network without NC, i.e., Figure 4.6, there are only two iterations of loop *Repeat Until* in steps from 2 to 13.

In the 1st iteration, after submitting $LP\ (f)$ to LP solver in step 4, we get rate R_{min} returned from LP solver in step 5, which is **0.03125** (packets/slot). It is the max-min rate, which is feasible to allocate to flows with undetermined rates, i.e. flows f_1, f_2 and f_3. This is because among the two cliques in two Figures 4.6.c.1 and 4.6.c.2, the clique in Figure 4.6.c.2 is bottlenecked first based on the respective capacity constraint. Since there are only two flows f_1 and f_2 with undetermined rate in the bottlenecked clique, the rate of flows f_1 and f_2 are determined first by loop *For* in steps from 6 to 11. Note that from now on the rate of flow f_1 and f_2 are determined and unchanged, i.e., with status $FIXED$. More specifically, the flow and sub-flow rates of all the flows on each stage are determined as follows.

(1) **Flow $\boldsymbol{f_1}$ -** Flow rate: $r_1 = 0.03125;$

Stage $\mathcal{T}^{(1)}$: $r_{f_1}^{(1)}(l_1) = 0.03125;\ r_{f_1}^{(1)}(l_2) = 0;\ r_{f_1}^{(1)}(l_3) = 0;$

Stage $\mathcal{T}^{(2)}$: $r_{f_1}^{(2)}(l_1) = 0;\ r_{f_1}^{(2)}(l_2) = 0.03125;\ r_{f_1}^{(2)}(l_3) = 0.03125;$

(2) **Flow $\boldsymbol{f_2}$ -** Flow rate: $r_2 = 0.03125;$

Stage $\mathcal{T}^{(1)}$: $r_{f_2}^{(1)}(l_5) = 0;\ r_{f_2}^{(1)}(l_4) = 0;$

Stage $\mathcal{T}^{(2)}$: $r_{f_2}^{(2)}(l_5) = 0.03125;\ r_{f_2}^{(2)}(l_4) = 0.03125;$

(3) **Flow $\boldsymbol{f_3}$ -** Flow rate: $r_3 = 0.03125;$

Stage $\mathcal{T}^{(1)}$: $r_{f_3}^{(1)}(l_6) = 0.03125;$

Stage $\mathcal{T}^{(2)}$: $r_{f_3}^{(2)}(l_6) = 0;$

In the 2nd iteration, after submitting $LP\ (f)$ to LP solver in step 4, we get rate R_{min} returned from LP solver in step 5, which is **0.21875** (packets/slot). It is the max-min rate, which is to indicate the max-min rate feasible to allocate to flows with undetermined rates, i.e. flow f_3. This is because among the two cliques in two Figures 4.6.c.1 and 4.6.c.2, the clique in Figure 4.6.c.1 is bottlenecked next based on the respective capacity constraint. Since there is only flow f_3 with undetermined rate in the bottlenecked clique, the rate of flow f_3 is determined next by loop *For* in steps from 6 to 11. More specifically, the flow and sub-flow rates of all the flows on each stage are determined as follows.

(1) **Flow $\boldsymbol{f_1}$ -** Flow rate: $r_1 = 0.03125;$

$Stage\ \mathcal{T}^{(1)}$: $r_{f_1}^{(1)}(l_1) = 0.03125;\ r_{f_1}^{(1)}(l_2) = 0;\ r_{f_1}^{(1)}(l_3) = 0;$

$Stage\ \mathcal{T}^{(2)}$: $r_{f_1}^{(2)}(l_1) = 0;\ r_{f_1}^{(2)}(l_2) = 0.03125;\ r_{f_1}^{(2)}(l_3) = 0.03125;$

(2) **Flow $\boldsymbol{f_2}$ -** Flow rate: $r_2 = 0.03125;$

$Stage\ \mathcal{T}^{(1)}$: $r_{f_2}^{(1)}(l_5) = 0;\ r_{f_2}^{(1)}(l_4) = 0;$

$Stage\ \mathcal{T}^{(2)}$: $r_{f_2}^{(2)}(l_5) = 0.03125;\ r_{f_2}^{(2)}(l_4) = 0.03125;$

(3) **Flow $\boldsymbol{f_3}$ -** Flow rate: $r_3 = 0.21875;$

$Stage\ \mathcal{T}^{(1)}$: $r_{f_3}^{(1)}(l_6) = 0.21875;$

$Stage\ \mathcal{T}^{(2)}$: $r_{f_3}^{(2)}(l_6) = 0;$

Turning out attention to the execution of MP with NC, i.e., Figure 4.7, there are also only two iterations of loop *Repeat Until* in steps from 2 to 13.

In the 1st iteration, after submitting $LP\ (f)$ to LP solver in step 4, with NC we get rate R_{min} returned from LP solver in step 5, which is **0.04166** (packets/slot). Note that this rate is 30% greater than it was without NC. It is the max-min rate, which is feasible to allocate to flows with undetermined rates, i.e. flows f_1, f_2 and f_3. This is because among the two cliques in two Figures 4.6.c.1 and 4.6.c.2, the clique in Figure 4.6.c.2 is bottlenecked first based on the respective capacity constraint. Since there are only two flows f_1 and f_2 with undetermined rate in the bottlenecked clique, the rates of flow f_1 and f_2 are determined first by loop For in steps from 6 to 11. Note that from now on the rate of flow f_1 and f_2 are determined and unchanged, i.e., with status $FIXED$. More specifically, the flow and sub-flow rates of all the flows on each stage are determined as follows.

(1) **Flow $\boldsymbol{f_1}$ -** Flow rate: $r_1 = 0.04166;$

$Stage\ \mathcal{T}^{(1)}$: $r_{f_1}^{(1)}(l_1) = 0.04166;\ r_{f_1}^{(1)}(l_2) = 0;$

$r_{f_1}^{(1)}(l_3) = 0;\ r_{f_1,f_2}^{(1)}(l_3, l_5) = 0;$

$Stage\ \mathcal{T}^{(2)}$: $r_{f_1}^{(2)}(l_1) = 0;\ r_{f_1}^{(2)}(l_2) = 0.04166;$

$r_{f_1}^{(2)}(l_3) = 0;\ r_{f_1,f_2}^{(2)}(l_3, l_5) = 0.04166;$

(2) **Flow $\boldsymbol{f_2}$ -** Flow rate: $r_2 = 0.04166;$

$Stage\ \mathcal{T}^{(1)}$: $r_{f_2}^{(1)}(l_5) = 0;\ r_{f_1,f_2}^{(1)}(l_3, l_5) = 0;$

$r_{f_2}^{(1)}(l_4) = 0;$

$Stage\ \mathcal{T}^{(2)}$: $r_{f_2}^{(2)}(l_5) = 0;\ r_{f_1,f_2}^{(2)}(l_3, l_5) = 0.04166;$

$r_{f_2}^{(2)}(l_4) = 0.04166;$

(3) **Flow $\boldsymbol{f_3}$ -** Flow rate: $r_3 = 0.04166;$

Stage $\mathcal{T}^{(1)}$: $r_{f_3}^{(1)}(l_6) = 0.04166;$

Stage $\mathcal{T}^{(2)}$: $r_{f_3}^{(2)}(l_6) = 0;$

In the 2nd iteration, after submitting $LP\ (f)$ to LP solver in step 4, with NC we get rate R_{min} returned from LP solver in step 5, which is **0.20833** (packets/slot). Note that this rate is less than the one without NC because of the increase of flow f_1's rate by NC. It is the max-min rate, which is to indicate the max-min rate feasible to allocate to flows with undetermined rates, i.e. flow f_3. This is because among the two cliques in two Figures 4.7.c.1 and 4.7.c.2, the clique in Figure 4.7.c.1 is bottlenecked next based on the respective capacity constraint. Since there is only flow f_3 with undetermined rate in the bottlenecked clique, the rate of flow f_3 is determined next by loop For in steps from 6 to 11. More specifically, the flow and sub-flow rates of all the flows on each stage are determined as follows.

(1) **Flow f_1 -** Flow rate: $r_1 = 0.04166;$

Stage $\mathcal{T}^{(1)}$: $r_{f_1}^{(1)}(l_1) = 0.04166;\ r_{f_1}^{(1)}(l_2) = 0;$

$r_{f_1}^{(1)}(l_3) = 0;\ r_{f_1,f_2}^{(1)}(l_3, l_5) = 0;$

Stage $\mathcal{T}^{(2)}$: $r_{f_1}^{(2)}(l_1) = 0;\ r_{f_1}^{(2)}(l_2) = 0.04166;$

$r_{f_1}^{(2)}(l_3) = 0;\ r_{f_1,f_2}^{(2)}(l_3, l_5) = 0.04166;$

(2) **Flow f_2 -** Flow rate: $r_2 = 0.04166;$

Stage $\mathcal{T}^{(1)}$: $r_{f_2}^{(1)}(l_5) = 0;\ r_{f_1,f_2}^{(1)}(l_3, l_5) = 0;$

$r_{f_2}^{(1)}(l_4) = 0;$

$Stage\ \mathcal{T}^{(2)}$: $r_{f_2}^{(2)}(l_5) = 0$; $r_{f_1,f_2}^{(2)}(l_3, l_5) = 0.04166$;

$r_{f_2}^{(2)}(l_4) = 0.04166$;

(3) **Flow $\boldsymbol{f_3}$ -** Flow rate: $r_3 = 0.20833$;

$Stage\ \mathcal{T}^{(1)}$: $r_{f_3}^{(1)}(l_6) = 0.20833$;

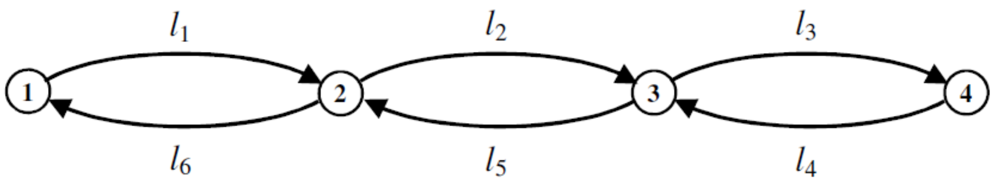
$Stage\ \mathcal{T}^{(2)}$: $r_{f_3}^{(2)}(l_6) = 0$;

From the above example and its results when we run the MP algorithm with NC on the network, we have the following observations of the impact of NC:

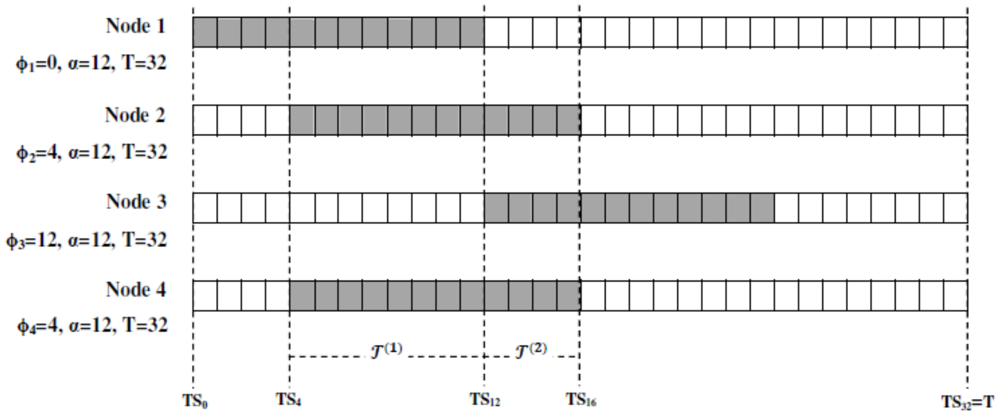
(1) NC can improve the rates of two flows, e.g., flows f_1 and f_2, traversing a node (greater than those without NC), at which the two flows can be combined and transmitted together for saving the flow transmissions;

(2) NC may indirectly cause the decrease of the rates of flows, e.g., flow f_3, (less than those without NC), which cannot be combined with any of the other flows, because of the increase of the rates of flows by NC;

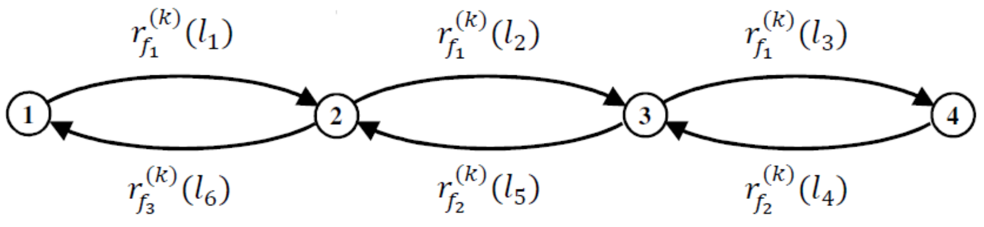
(3) NC may change the max-min allocation to flows compared to the case when no NC is applied because it may increase some flows' rates but also simulraneously decrease the other flows' rates as observed from (1) and (2), however leading to higher overall throughput (the total prior without NC is 0.28125 while with NC it is 0.29165).



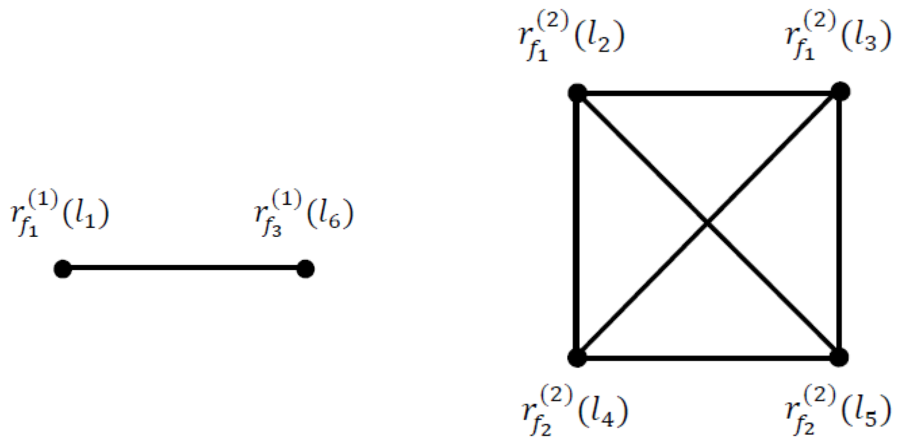
(a.1) Link topology



(a.2) States ON/OFF of nodes (ON slots are gray)



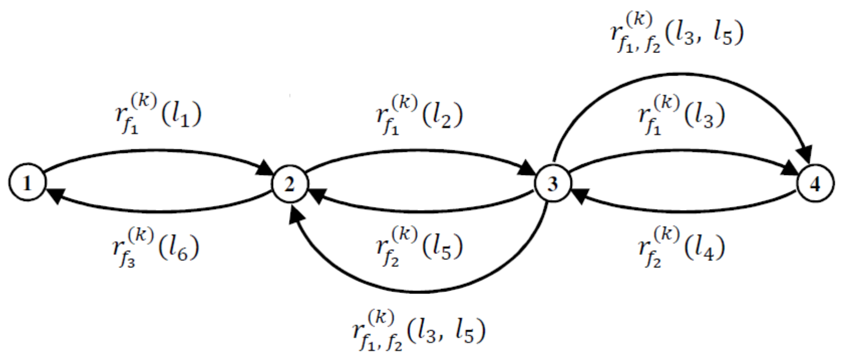
(b) Multi-hop flows



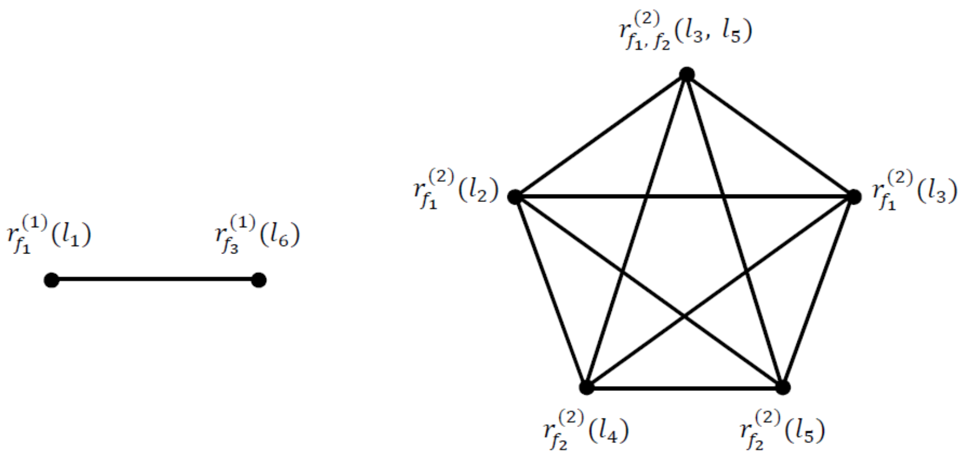
(c.1) CG in $\mathcal{T}^{(1)}$

(c.2) CG in $\mathcal{T}^{(2)}$

Figure 4.6. An illustrative example for the MP algorithm *without* NC (CG = Contention Graph)



(b) Multi-hop flows



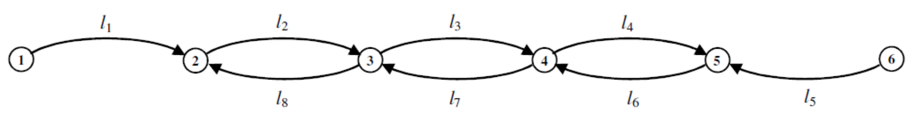
(c.1) CG in $\mathcal{T}^{(1)}$ (c.2) CG in $\mathcal{T}^{(2)}$

Figure 4.7. An illustrative example for the MP algorithm *with* NC (CG = Contention Graph)

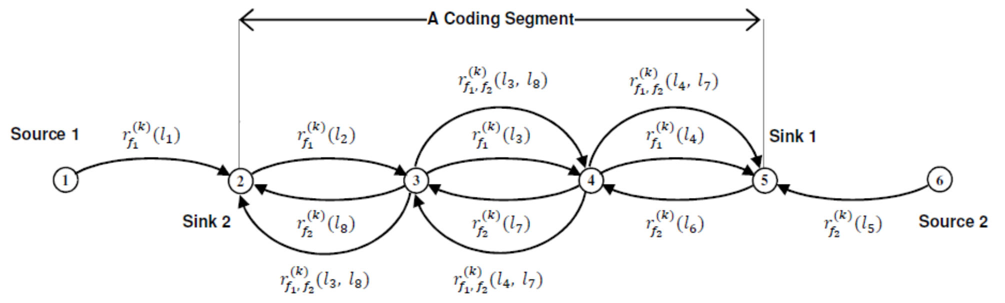
4.2 SIMULATION-BASED PERIODIC PATTERN EXCISION

4.2.1 DELAY CODING

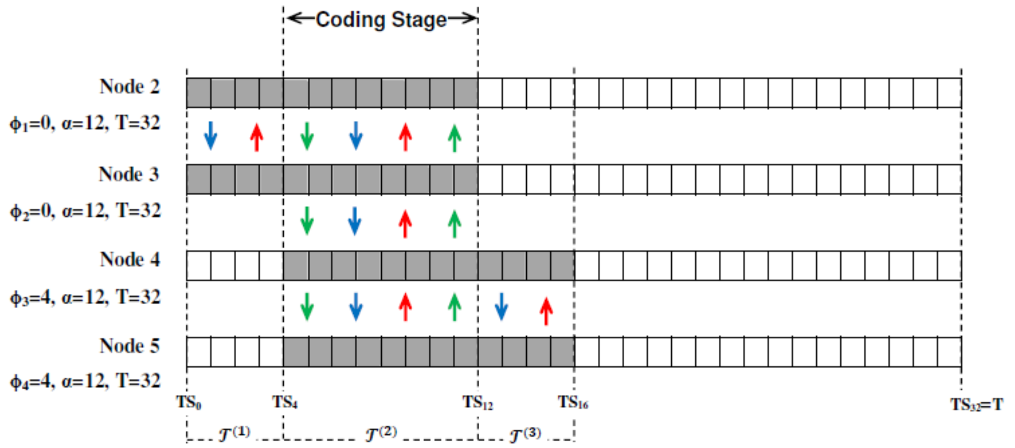
To improve the throughput that is generally reduced due to DC, our work uses the XOR-pairwise NC, in which there exist two flows over a common part (at least two hops), or a *coding segment* for short, of their routing paths. The duty-cycle of each intermediate node of the segment must simultaneously overlap the duty-cycles of its two neighbors along the segment so that the coded sub-flow from the two flows transmitted at the intermediate node can be received by the two neighbors.



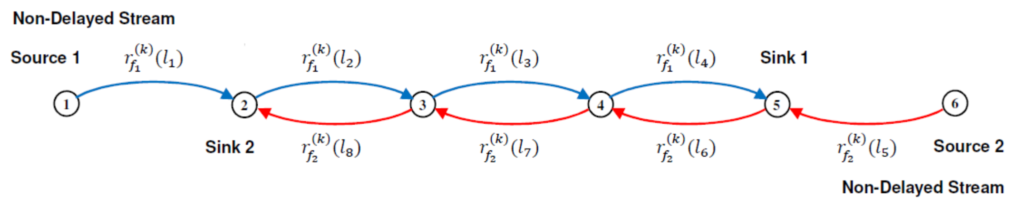
(a) Link topology



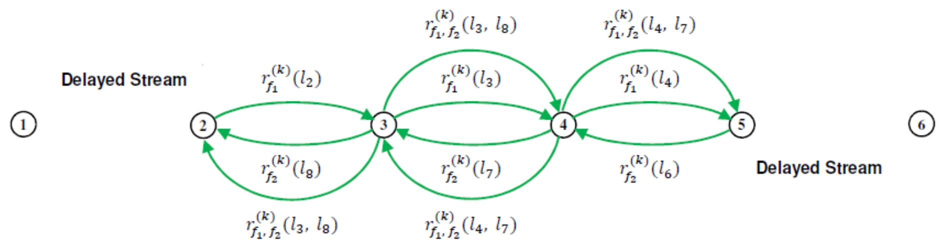
(b) Multi-hop flows



(c) States ON/OFF of duty-cycled nodes (ON slots are gray)



(d) Flow pattern of non-delayed streams



(e) Flow pattern of delayed streams

Figure 4.8. An illustrative example of delay coding process

Given a coding segment, we know where to apply the network coding, i.e., at the intermediate nodes along the coding segment, and when to apply the network coding, i.e., during the overlap period of the three cycles of an intermediate node and its two neighbors of the same segment, which is called a *coding stage* for short.

Example 4.2. For illustration, we use the network with link topology in Figure 4.8.a and flow pattern in Figure 4.8.b, which includes two flows, flow f_1 from source node 1 to sink node 5 and flow f_2 from source node 6 to sink node 2, and a coding segment (from nodes 2 to 5) of the routing paths of the two flows. Figure 4.8.c shows the duty-cycles of nodes including intermeadiate nodes of the coding segment, e.g., nodes 3 and 4, at which the two flows f_1 and f_2 can be combined or coded only during the coding stage $\mathcal{T}^{(2)}$, where exists the overlap between the nodes'duty-cycles and their neighbors'.

To produce more coding opportunities, we introduce the delay coding scheme, in which packets from one flow have to delay at an intermediate node until they are coded with those from the other flow. However, the packets should be delayed only in a coding stage, where the coding can take place. Outside the coding stage, if the link connection is still available, then packets should be scheduled for transmissions to utilize the available bandwidth. To implement the idea, we create two streams of packets scheduled only along a coding segment in the delay coding scheme, in which the delayed stream, e.g., green arrows in Figures 4.8.c and 4.8.e, exists only in the coding stage, e.g., $\mathcal{T}^{(2)}$ in Figure 4.8.c, while the non-delayed stream, e.g., blue and red arrows in Figures 4.8.c and 4.8.d, exists both inside and outside the coding stage, e.g., $\mathcal{T}^{(1)}$ and $\mathcal{T}^{(3)}$ in Figure 4.8.c.

Specifically, given a coding segment with two flows traversing in opposite directions, the delay coding scheme has two stages, the *initial stage* and the *steady stage*. In the initial stage, the packets from one flow with smaller id, e.g., f_1, go forward while those from the inverse flow with larger id, e.g., f_2, wait for coding at the intermediate node next to the segment's end, e.g., node 4 in Figure 4.8.e, until the packets of the two flows meet for the first time. In the steady stage, which follows the initial stage, packets from the two inverse flows have to wait for each other at intermediate nodes for coding. Note that non-delayed packets from the non-delayed stream are regular packets while delayed packets from the delayed stream are created by converting from regular packets at the two ends of the coding segment during the coding stage.

```
Input:  Coded sub-flow $r^{(k)}_{f_1,f_2}(l_1, l_2)$ from node $n$ during a coding stage;
        Transmission queue $TQ\ [0..\mathrm{L}-1]$ of node $n$;
Output: A coded packet, i.e., $P \neq NULL$, or none, i.e., $P = NULL$;
Begin
01.  $P \leftarrow get\ (TQ\ [0..\mathrm{L}-1], DELAY)$;
02.  If $(P \neq NULL)\ and\ (paired\ (P, TQ\ [0..\mathrm{L}-2], DELAY) \neq NULL)$ Then
03.      $P \leftarrow P \oplus paired\ (P, TQ\ [0..\mathrm{L}-2], DELAY)$;
04.  Else
05.      If $(P \neq NULL)\ and\ (paired\ (P, TQ\ [0..\mathrm{L}-2], NODEL) \neq NULL)$ Then
06.          $P \leftarrow P \oplus paired\ (P, TQ\ [0..\mathrm{L}-1], NODEL)$;
07.      Else
08.          If $(P \neq NULL)$ Then
09.              $insert\ (P, TQ\ [0..\mathrm{L}-2])$; $P \leftarrow NULL$;
10.          End If
11.      End If
12.  End If
End
```

Figure 4.9. Pseudo-code of the delay coding algorithm

Figure 4.9 shows the pseudo-code of the delay coding scheme, which outputs a coded packet, i.e., $P \neq NULL$, or none, i.e., $P \leftarrow NULL$. Suppose we have coded sub-flow $r^{(k)}_{f_1,f_2}(l_1, l_2)$ combined from two flows f_1, f_2 in a coding stage at an intermediate node n with transmission queue $TQ\ [0..\mathrm{L}-1]$ with length L or $TQ\ [\,]$

for short. From steps 1 to 4, only the delayed stream is chosen and a coded packet is output if the coding condition in step 2 is satisfied during the coding stage. Note that to further improve the throughput, the delayed stream, i.e., steps 2 to 4, is treated with higher priority than the non-delayed stream, i.e., steps 5 to 7. To provide even more opportunities for coding, the delayed stream's packets are allowed to code with those of the non-delayed stream, i.e., steps 5 to 7. The following functions are used in the pseudo-code: (1) $get\ (TQ\ [\,], DELAY)$ or $get\ (TQ\ [\,], NODEL)$ is to get a packet of the delayed stream or the non-delayed stream from transmission queue $TQ\ [\,]$ of node n; (2) $paired\ (P, TQ\ [\,], DELAY)$ or $paired\ (P, TQ\ [\,], NODEL)$ is to pair packet P with another packet of the delayed stream or the non-delayed stream from queue $TQ\ [\,]$ of node n; (3) $insert\ (P, TQ\ [\,], DELAY)$ is to insert packet P back into queue $TQ\ [\,]$ of node n.

4.2.2 EXTENDING THE SCHEDULING ALGORITHM TO NC

The scheduling algorithm with NC works exactly the same as that without NC does except non-coded sub-flows are extended to both non-coded and coded sub-flows. Without NC, the transmission of an ON non-coded sub-flow of flow f over link l in timeslot i by invoking the Maximum Weighted Independent Set (MWIS) approximation algorithm is determined by the weight $weight(f, l, i)$ of the sub-flow as computed by (3.22). With NC, the weight assigned to an ON coded sub-flow from two flows f_1 and f_2 over two respective links l_1 and l_2 during timeslot i is calculated as follows.

$$weight(f_1, f_2, l_1, l_2, i) = max(weight(f_1, l_1, i), weight(f_2, l_2, i)) \quad (4.30)$$

Note that the $max()$ function in (4.30) is to maximize the weight assignment of the coded sub-flow from the two component weights, $weight\ (f_1, l_1, i)$ and $weight\ (f_2, l_2, i)$, so that the scheduling algorithm can get more opportunities to benefit from NC.

Figure 4.10 shows the pseudo-code of the scheduling algorithm with NC in a given slot. Note that the only differences between the algorithm and the one without NC are coded sub-flows and their weight assignment to the MWIS algorithm as described in the previous paragraph.

```
Input:   Contention matrix $M\_SF$ [0..M-1][0..M-1] of all the M ON sub-flows;
         Weight of each of the M$'$ ON sub-flows with available packets in slot $i$;
Output:  Array $S\_SF$ [0..M$''$-1] of M$''$ of ON sub-flows to be scheduled in slot $i$;
Begin
01.   For Each ON sub-flow, $(f, l)$ or $(f_1, f_2, l_1, l_2)$, with packets in queue Do
02.       If It is an ON non-coded sub-flow $(f, l)$ Then
03.           $w \leftarrow weight\ (f, l, i)$;
04.       Else
05.           $w \leftarrow max\ (weight\ (f_1, l_1, i),\ weight\ (f_2, l_2, i))$;
06.       End If
07.       $G\_SF\ [0..M'-1][0..M'-1] \leftarrow graph(f, l, w, M\_SF\ [0..M-1][0..M-1])$;
08.   End For
09.   $S\_SF\ [0..M''-1] \leftarrow MWIS\ (G\_SF\ [0..M'-1][0..M'-1])$;
End
```

Figure 4.10. The scheduling algorithm with NC during slot i

4.2.3 SCHEDULE EXCISION PROCESS WITH NC

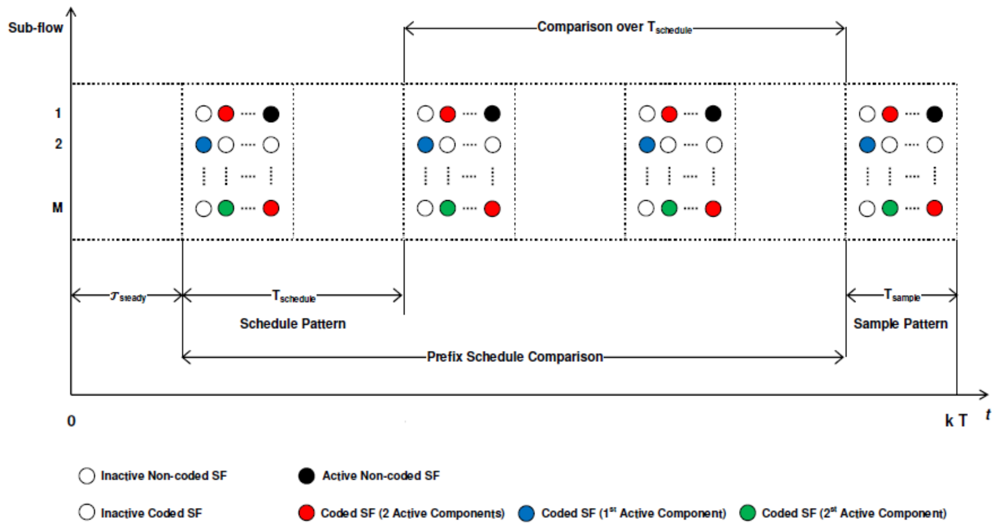

Sub-flow
Comparison over $T_{schedule}$
1
2
M
T_{steady}
$T_{schedule}$
Schedule Pattern
T_{sample}
Sample Pattern
Prefix Schedule Comparison
0
k T
t
Inactive Non-coded SF
Active Non-coded SF
Inactive Coded SF
Coded SF (2 Active Components)
Coded SF (1st Active Component)
Coded SF (2nd Active Component)

Figure 4.11. Illustration of the simulation-based schedule excision process with NC

The schedule excision process without NC includes two main comparisons, the prefix schedule comparison and the entire schedule comparison, in which the state of whether a sub-flow is scheduled or not is considered in each timeslot. Therefore, in order for the schedule excision process with NC to properly function like that without NC, we need to add more scheduling states to a coded sub-flow. These states capture the fact that an NC coded transmission introduced in the schedule may have up to two simulataneous possible recipients.

For a non-coded sub-flow determined by two ends, one sender and one receiver for only one flow, we only need the two following states: (1) the active state that indicates both the sender end and the receiver end are active represented by black circles in Figure 4.11, or ACTIVE for short; (2) the inactive state that indicates either the sender end or the receiver end is inactive represented by white circles in Figure 4.11, or INACTIVE.

However, for a coded sub-flow determined by three ends, one sender and two receivers for at most two flows, we need the four following states: (1) the active state that indicates the sender end and both the receiver ends are active represented by red circles in Figure 4.11, or BT_ACT for short; (2) the active state that indicates the sender end and the 1st receiver end are active represented by blue circles in Figure 4.11, or F1_ACT; (3) the active state that indicates the sender end and the 2nd receiver end are active represented by green circles in Figure 4.11, or F2_ACT; (4) the inactive state that indicates either the sender end or both the receiver ends are inactive represented by white circles in Figure 4.11, or INACTIVE.

4.2.4 FLOW BALANCE APPROACHES WITH NC

As mentioned in Chapter 3, we have the two following options to get a template without NC, which has flow balance guaranteed: (1) by making it a criterion during the pattern matching process (option A); (2) by enforcing it after the excision has taken place (option B) by trimming away the transmissions that result in the imbalance.

To implement options A and B with NC, we use two algorithms **CheckBalanceNC ()** and **MakeBalanceNC ()** presented by pseudo-codes in Figures 4.12 and 4.14, respectively. Algorithm **CheckBalanceNC ()** is to check if all the flows in a given schedule template are balanced, which is used during the pattern matching process. Meanwhile, algorithm **MakeBalanceNC ()** is to enforce the flow balance on a given imbalanced schedule template, which is used after the pattern matching process.

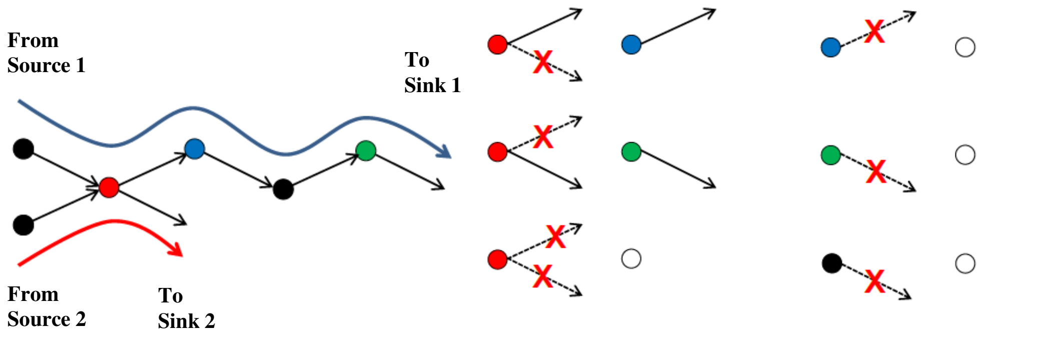
With the pseudo-code of algorithm **CheckBalanceNC ()** in Figure 4.12, we suppose a given schedule template with length $T_{schedule}$ and M sub-flows is stored in array *ScheduleArray* [0..$T_{schedule}$ – 1][0..M – 1]. Note that the array can contain five possible values: ACTIVE, BT_ACT, F1_ACT, F2_ACT, and INACTIVE, in which INACTIVE can be used for both non-coded and coded sub-flows. We also assume that the minimum value, i.e., $min\ (f)$, which is among the numbers of transmissions of a given flow f over each of its links l along its routing path, is predetermined. In addition, we use variable $\#_f(l)$ to store the number of transmissions of each flow f over link l, which is initialized to zero in step 1. The variable is increased by one whenever f over link l is scheduled, i.e., ACTIVE (step 4), or BT_ACT or F1_ACT (step 10), or BT_ACT or F2_ACT (step 16), during traversing the template (steps 2 to 21). Note that notation (f, l) is to indicate flow f goes over link l. The algorithm returns imbalance indication (steps 6, 12 or 18) whenever the variable is greater than the minimum respective value $min\ (f)$ or balance indication otherwise (step 22).

```
Input:  Schedule array ScheduleArray [0..T_schedule – 1][0..M – 1];
        Number $min(f)$ of each flow $f \in \mathcal{F}$ in the array;
Output:  YES if the array is balanced or NO otherwise;
Begin
01.   Set $\#_f(l)$ to zero for all the flows $f \in \mathcal{F}$ over link $l \in \mathcal{L}$;
02.   For  ($i$ from 0 to T_schedule – 1, $j$ from 0 to M – 1)  Do
03.       If  ((ScheduleArray [$i$][$j$] = ACTIVE) and ($j = (f, l)$))  Then
04.           $\#_f(l) = \#_f(l) + 1$;
05.           If  ($\#_f(l) > min(f)$)  Then
06.               Return NO;
07.           End If
08.       End If
09.       If  ((ScheduleArray [$i$][$j$] = F1_ACT) or
              (ScheduleArray [$i$][$j$] = BT_ACT)) and ($j = (f, l_1)$))  Then
10.           $\#_f(l_1) = \#_f(l_1) + 1$;
11.           If  ($\#_f(l_1) > min(f)$)  Then
12.               Return NO;
13.           End If
```

```
14.         End If
15.         If ((ScheduleArray [i][j] = F2_ACT) or
                (ScheduleArray [i][j] = BT_ACT)) and (j = (f, l_2))) Then
16.             $\#_f(l_2) = \#_f(l_2) + 1$;
17.             If ($\#_f(l_2) > min(f)$) Then
18.                 Return NO;
19.             End If
20.         End If
21.     End For
22.     Return YES;
End
```

Figure 4.12. Pseudo-code of algorithm **CheckBalanceNC ()**

The idea of the enforcement of flow balance (option B) upon the template is that for each flow we walk through the active spots of sub-flows of the flow along its routing path from the source to the sink in the template. We then mark the spots as visited only when we have walked through the entire path, i.e., we have reached the sink. Hence, the active spots, which have not marked yet, are the imbalanced spots, which need to be removed from the template, i.e., turned to inactive spots.



(a) A common spot between two routing paths (b) FBE for fully-coded spots (c) FBE for partially or non-coded spots

Figure 4.13. The process of flow balance enforcement (FBE)

The idea is effective only with a template without NC, in which all the routing paths are separated. However, in a schedule template generated to include NC, the paths may have common active spots of coded sub-flows (in red circles) as presented in Figure 4.13.a and hence we need a more elaborate enforcement of flow balance for non-coded and coded spots.

(1) Assume we have performed the marking process for all the flows as described earlier. If the active imbalanced spot is a coded sub-flow with its active sender end and two receiver ends, i.e., in red circle, we turn the spot from red to blue or green (or white) if it is marked once (or none) by walking through the routing path of its 1st, i.e., indicated as YES_F1, or 2nd, i.e., indicated as YES_F2, receiver end (or none, i.e., indicated as NO). This scenario is illustrated by Figure 4.13.b. Noting that by doing so, the trimming of one flow's active imbalanced spot, i.e., to turn from red to blue or green or white, does not impact the other flow's active balanced spot combined in the coded sub-flow;

(2) If the active imbalanced spot is a coded sub-flow with just one active receiver end, i.e., in blue or green circle, or the imbalanced spot is a non-coded sub-flow in the active state, i.e., in black circle, we turn the spot from blue or green or black into white, if it has not been marked at all. This scenario is illustrated by Figure 4.13.c.

The idea is demonstrated in the pseudo-code of algorithm **MakeBalanceNC ()** in Figure 4.14. We assume that we have walked through the active spots of sub-flows of each flow along its routing path from the source to the sink in the template and marked the spots in array *VisitedArray* [0..$T_{schedule}$ – 1][0..M – 1]. Note that only an active spot is marked as visited only when we have walked through the entire path, i.e., we have reached the sink. The array can contain the following values, NO (unmarked), YES (marked on an active spot of a non-coded sub-flow), YES_F1 (marked once on an active spot via the 1st receiver end of a coded sub-flow), YES_F2 (marked once on an active spot via the 2nd receiver end of a coded sub-flow), YES_BT (marked twice on an active spot via both the receiver ends of a coded sub-flow).

Hence, active spots, i.e., ACTIVE or F1_ACT or F2_ACT or BT_ACT, which have not been marked yet, i.e., NO visited, are considered as imbalanced spots in steps 2 and 3. The active imbalanced spots are illustrated in Figures 4.13.b and 4.13.c (mostly). Hence, they need to be removed from the template, i.e., turned to inactive spots, i.e., INACTIVE, by step 4. There are also other active imbalanced spots, i.e., BT_ACT, which have marked only once, i.e., YES_F1 or YES_F2, are considered as imbalanced spots in steps 8 and 11, respectively. The active imbalanced spots are illustrated in Figure 4.13.b, which need to be trimmed into F1_ACT or F2_ACT for enforcement of flow balance. Eventually, after traversing the whole template, the algorithm turns the given schedule array from an imbalanced template to a balanced one.

```
Input:  Imbalanced array ScheduleArray [0..$T_{schedule}$ – 1][0..M – 1];
        Visited array VisitedArray [0..$T_{schedule}$ – 1][0..M – 1];
Output:  Balanced array ScheduleArray [0..$T_{schedule}$ – 1][0..M – 1];
Begin
01.   For  ($i$ from 0 to $T_{schedule}$ – 1, $j$ from 0 to M – 1)  Do
02.       If  (VisitedArray [$i$][$j$] = NO)  Then
03.           If  ((ScheduleArray [$i$][$j$] = ACTIVE) or
                  (ScheduleArray [$i$][$j$] = F1_ACT) or
                  (ScheduleArray [$i$][$j$] = F2_ACT) or
                  (ScheduleArray [$i$][$j$] = BT_ACT))  Then
04.               ScheduleArray [$i$][$j$] = INACTIVE;
05.           End If
06.       Else
07.           If  (ScheduleArray [$i$][$j$] = BT_ACT)  Then
08.               If  (VisitedArray [$i$][$j$] = YES_F1)  Then
09.                   ScheduleArray [$i$][$j$] = F1_ACT;
10.               Else
11.                   If  (VisitedArray [$i$][$j$] = YES_F2)  Then
```

```
12.                     ScheduleArray [i][j] = F2_ACT;
13.                 End If
14.             End If
15.         End If
16.     End If
17. End For
End
```

Figure 4.14. Pseudo-code of algorithm **MakeBalanceNC ()**

To illustrate the schedule excision process with NC after flow balance is checked and guaranteed, we run the scheduling algorithm without and with NC on the network in Example 4.1. As a result, we construct the schedule without NC in Figure 4.15 and that with NC in Figure 4.16.

t	$r_{f_1}^{(k)}(l_1)$	$r_{f_1}^{(k)}(l_2)$	$r_{f_1}^{(k)}(l_3)$	$r_{f_2}^{(k)}(l_4)$	$r_{f_2}^{(k)}(l_5)$	$r_{f_3}^{(k)}(l_6)$
1	○	○	○	○	○	○
2	○	○	○	○	○	○
3	○	○	○	○	○	○
4	○	○	○	○	○	○
5	○	○	○	○	○	●
6	○	○	○	○	○	●
7	○	○	○	○	○	●
8	○	○	○	○	○	●
9	○	○	○	○	○	●
10	●	○	○	○	○	○
11	○	○	○	○	○	●
12	○	○	○	○	○	●
13	○	●	○	○	○	○
14	○	○	●	○	○	○
15	○	○	○	●	○	○
16	○	○	○	○	●	○
17	○	○	○	○	○	○
…	○	○	○	○	○	○
32	○	○	○	○	○	○

Figure 4.15. The schedule is constructed when we run the scheduling algorithm *without* NC on the network in Example 4.1

In Figure 4.15, timeslot $\boldsymbol{t}$ is within duration T$_{\text{schedule}}$ without NC, i.e., from 1 to 32, and non-coded sub-flow $\boldsymbol{r_f^{(k)}(l)}$ indicates flow f, i.e., from f_1 to f_3, over link l, i.e., from l_1 to l_6, across stages k, i.e., from 1 to 2. Note that white circles (black circles) mean the non-coded sub-flows are not scheduled (are scheduled). Meanwhile, in Figure 4.16, timeslot $\boldsymbol{t}$ is within duration T$_{\text{schedule}}$ with NC, i.e., from 1 to 96, and coded sub-flow $\boldsymbol{r_{f_1,f_2}^{(k)}(l_3,l_5)}$ indicates two flows f_1 and f_2 can be combined over two links l_3 and l_5 across stages k. Note that red circles (white circles) at column $r_{f_1,f_2}^{(k)}(l_3,l_5)$ mean both flows f_1 and f_2 are scheduled (not scheduled) over two links l_3 and l_5.

From the results in Figures 4.15 and 4.16, we observe that the constructed schedules are guaranteed flow balance. For example, the rates of flow f_1 over links l_1, l_2 and l_3 along the routing path are equal to 1/32 or **0.03125** in Figure 4.15 while they are 4/96 or **0.04166** in Figure 4.16. The results are consistent with those from the MP algorithm as described above. Paticularly, with NC in this example, two non-coded sub-flows, $r_{f_1}^{(k)}(l_3)$ and $r_{f_2}^{(k)}(l_5)$, are no longer scheduled (presented by white circles) because of scheduling coded-subflow $r_{f_1,f_2}^{(k)}(l_3,l_5)$ (presented by red circles). We observe that the length of the constructed schedule changes with the introduction of additional coded sub-flows in this example, e.g., the schedule length increases from 32 to 96 slots when coded sub-flow $r_{f_1,f_2}^{(k)}(l_3,l_5)$ is created. We will see later from the simulation results in Table 4.1, the introduction of additional coded sub-flows may result in the decrease of the schedule length. From the observations, we have no conclusive answer on whether adding NC reduces or expands the schedule length.

$\boldsymbol{t}$	$\boldsymbol{r_{f_1}^{(k)}(l_1)}$	$\boldsymbol{r_{f_1}^{(k)}(l_2)}$	$\boldsymbol{r_{f_1}^{(k)}(l_3)}$	$\boldsymbol{r_{f_1,f_2}^{(k)}(l_3,l_5)}$	$\boldsymbol{r_{f_2}^{(k)}(l_5)}$	$\boldsymbol{r_{f_2}^{(k)}(l_4)}$	$\boldsymbol{r_{f_3}^{(k)}(l_6)}$
1	○	○	○	○	○	○	○
2	○	○	○	○	○	○	○
3	○	○	○	○	○	○	○
4	○	○	○	○	○	○	○
5	○	○	○	○	○	○	●

6	○	○	○	○	○	○	●
7	○	○	○	○	○	○	●
8	○	○	○	○	○	○	●
9	○	○	○	○	○	○	●
10	○	○	○	○	○	○	●
11	●	○	○`	○	○	○	○
12	○	○	○	○	○	○	●
13	○	●	○	○	○	○	○
14	○	○	○	○	○	●	○
15	○	○	○	🔴	○	○	○
16	○	○	○	○	○	●	○
17	○	○	○	○	○	○	○
…	○	○	○	○	○	○	○
32	○	○	○	○	○	○	○
33	○	○	○	○	○	○	○
34	○	○	○	○	○	○	○
35	○	○	○	○	○	○	○
36	○	○	○	○	○	○	○
37	○	○	○	○	○	○	●
38	○	○	○	○	○	○	●
39	○	○	○	○	○	○	●
40	○	○	○	○	○	○	●
41	○	○	○	○	○	○	●
42	●	○	○	○	○	○	○
43	○	○	○`	○	○	○	●
44	○	○	○	○	○	○	●
45	○	●	○	○	○	○	○
46	○	○	○	🔴	○	○	○
47	○	○	○	○	○	●	○
48	○	○	○	○	○	●	○
49	○	○	○	○	○	○	○
…	○	○	○	○	○	○	○
64	○	○	○	○	○	○	○
65	○	○	○	○	○	○	○
66	○	○	○	○	○	○	○
67	○	○	○	○	○	○	○
68	○	○	○	○	○	○	○
69	○	○	○	○	○	○	●
70	○	○	○	○	○	○	●
71	○	○	○	○	○	○	●
72	○	○	○	○	○	○	●
73	○	○	○	○	○	○	●
74	●	○	○	○	○	○	○
75	○	○	○`	○	○	○	●
76	●	○	○	○	○	○	○
77	○	●	○	○	○	○	○
78	○	○	○	🔴	○	○	○
79	○	●	○	○	○	○	○

80	○	○	○	●	○	○	○
81	○	○	○	○	○	○	○
…	○	○	○	○	○	○	○
96	○	○	○	○	○	○	○

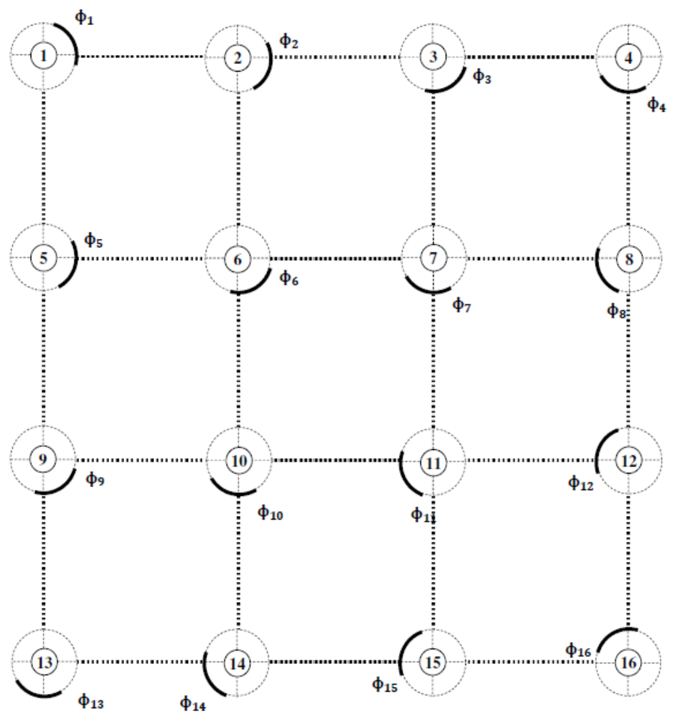
Figure 4.16. The schedule is constructed when we run the scheduling algorithm *with* NC on the network in Example 4.1

4.3 EVALUATION

4.3.1 THROUGHPUT IMPROVEMENT

In order to evaluate the ability of the proposed NC technique to improve the total throughput, we continue to use regular topologies as in Chapter 3, and experiment with different per-node phases. Like those without NC, there is no restriction to just regular topologies for the application of the proposed scheme, but the reader can follow easily the layout, phase relation, and flow paths on a regular topology.

We also consider a 4x4 topology as shown in Figure 4.17. Circular arcs are used to pictorially depict which nodes have the same phases (e.g., in the 4x4 topology nodes 3, 6, and 9 have the same phase). The duty cycle period is 32 slots and the ON intervals are all 12 slots long. Given that the phases can be staggered differently, we explore the impact staggering by various "phase gaps" (Figures 4.17.c) and indicate the phase (counted in slots) at which the corresponding node switches to ON. Additionally, a number of flows are simulated in the configuration following the pattern PT-5, which is shown in Figures 4.17.b. Note that the routes in pattern PT-5 were computed, based on single-shortest-path time-varying routing, i.e., the directions are along the sides of the grid. Also, to create opportunities for the XOR-pairwise NC in flow pattern PT-5, one flow (Source, Sink) always has the respective reverse flow (Sink, Source) with the same routing path. In our work, we only investigate whether the coding can be applied in DC-WSNs and how it can be applied. i.e., not to maximize the coding benefit when there are multiple choices of given flows for coding. This is a matter we briefly address in Chapter 5.



(a) 4x4 Grid topology

Flow Id	Routing Path
F_1	$1 \rightarrow 2 \rightarrow 3 \rightarrow 4$
F_2	$4 \rightarrow 3 \rightarrow 2 \rightarrow 1$
F_3	$5 \rightarrow 6 \rightarrow 7 \rightarrow 8$
F_4	$8 \rightarrow 7 \rightarrow 6 \rightarrow 5$
F_5	$9 \rightarrow 10 \rightarrow 11 \rightarrow 12$
F_6	$12 \rightarrow 11 \rightarrow 10 \rightarrow 9$
F_7	$13 \rightarrow 14 \rightarrow 15 \rightarrow 16$
F_8	$16 \rightarrow 15 \rightarrow 14 \rightarrow 13$
F_9	$1 \rightarrow 5 \rightarrow 9 \rightarrow 13$
F_{10}	$13 \rightarrow 9 \rightarrow 5 \rightarrow 1$
F_{11}	$2 \rightarrow 6 \rightarrow 10 \rightarrow 14$
F_{12}	$14 \rightarrow 10 \rightarrow 6 \rightarrow 2$
F_{13}	$3 \rightarrow 7 \rightarrow 11 \rightarrow 15$
F_{14}	$15 \rightarrow 11 \rightarrow 7 \rightarrow 3$
F_{15}	$4 \rightarrow 8 \rightarrow 12 \rightarrow 16$
F_{16}	$16 \rightarrow 12 \rightarrow 8 \rightarrow 4$

(b) Flow pattern PT-5

Phase Gaps	ϕ_1 ,ϕ_2 ,ϕ_3 ,ϕ_4 ϕ_5 ,ϕ_6 ,ϕ_7 ,ϕ_8 ϕ_9 ,ϕ_{10},ϕ_{11},ϕ_{12} ϕ_{13},ϕ_{14},ϕ_{15},ϕ_{16}
2	0 , 2 , 4 , 6 2 , 4 , 6 , 8 4 , 6 , 8 , 10 6 , 8 , 10 , 12
4	0 , 4 , 8 , 12 4 , 8 , 12 , 16 8 , 12 , 16 , 20 12 , 16 , 20 , 24
6	0 , 6 , 12 , 18 6 , 12 , 18 , 24 12 , 18 , 24 , 30 18 , 24 , 30 , 4
8	0 , 8 , 16 , 24 8 , 16 , 24 , 0 16 , 24 , 0 , 8 24 , 0 , 8 , 16
10	0 , 10 , 20 , 30 10 , 20 , 30 , 8 20 , 30 , 8 , 18 30 , 8 , 18 , 28

(c) Phase scheme for G4x4

Figure 4.17. Topologies, flows and phases used in the simulations

Another facet of the experiments is the phase relation between adjacent nodes. We again consider three schemes:

1. *Synchronized Phases* (SP), in which all the phases are synchronized to start at the same point in time. This is a benchmark value which essentially eliminates the time-varying nature of the communication graph.

2. *Fixed Ladder Phases* (FLP), in which any two adjacent nodes in the grid (vertically or horizontally) have their phases staggered 2 slots apart. This scheme is meant to test the impact of flow patterns and their interference on the schedule construction but with a fairly benign impact by the phase differences, i.e., adjacent ON periods overlap significantly over relatively long periods of time.

3. *Varied Ladder Phases* (VLP), in which we vary the staggering of the phases from 2 to 10 slots (Figures 4.17.c), creating increasingly "difficult" short periods over which links are active. Note that when the VLP scheme is used, the number of flows in each flow pattern is kept maximized and unchanged, i.e., in PT-5, the number of flows is 16.

In all cases, and as a matter of convention, instead of denoting the specific flows that comprise a certain mix, we create larger sets of flows by adding more flows in their numerical order. That is, sets of 1, 2, 3, etc. flows comprise correspondingly of the flows {F1}, {F1, F2}, {F1, F2, F3}, etc. Finally, note that because of the time varying nature of the underlying communication graph, the set of parameters used here that guides the duty cycling, generates a great deal of actual link topologies.

Representative results are summarized in Table 4.1 for pattern PT-5 with schemes SP, FLP and VLP. The **Coding** indicates the results done without NC, i.e., NC0, or with NC, i.e., NC1. The **#F or Gap** is the number of flows for phase schemes SP and FLP, or the stagger gap for phase scheme VLP. **Pre-Sim** is the total throughput of the constructed excised periodic schedule. **Numerical** is the total throughput as derived from the water filling algorithm. **Fairness** is a fairness index between what should be the achieved rates (as per water-filling) and what is achieved by the excised schedule. **Tschedule/T** expresses how many multiples of the duty cycle period is the length of the excised periodic schedule. **Mismatch** captures the fraction as percentage of sub-flows who did not match exactly during the prefix comparison. **Err** captures the fraction of sub-flows that did not match exactly during the complete schedule length comparison. The note field indicates whether Option A or Option B for correcting the flow balance was necessary in the corresponding case and which of the two options produced the best results.

With NC, the virtually identical numbers in the Pre-Sim and Numerical columns again demonstrate that the process of schedule excision is a reasonable approach to produce accurate schedules. Despite the fact that a key component of the simulation-based process is the invocation of an approximation to MWIS the fact is that a schedule is constructed via multiple (one per slot) invocation of MWIS, hence what matters is less the worst-case behavior of the approximation and more the average performance.

The process of matching the T_{sample} prefix and the entire $T_{schedule}$ with NC also cannot be expected to lead to perfect matches. For this reason, we allowed a maximum of mismatch (seen as difference in transmissions scheduled for a sub-flow between two compared patterns) of 10% which turned out to be very pessimistic, i.e., we did not reach that degree of mismatch in both the T_{sample} prefix and the entire $T_{schedule}$, which are presented by Mismatch and Err in the performance tables, respectively.

Another aspect of the results is that the produced schedule of duration $T_{schedule}$ with and without NC is a small multiple of T (the DC period). Without NC, the less loaded the network with flows, the smaller this multiple tends to be, but there are exceptions. With NC, the duration tends to be shorter than that without NC. This makes a lot of sense because with NC, particularly the delay coding, each pair of two non-coded sub-flows is possibly combined into a coded sub-flow, which leads to reduction of network load. As mentioned earlier, the relation

$T_{schedule}/T$ reveals the impact of the queue state. With NC, the queues still tend to behave periodically as without NC. Especially, due to the combination of two transmissions into one with NC, which results in the release of congestion in the network, the ratio $T_{schedule}/T$ with NC tends to be smaller than that without NC as shown in Table 4.1.

From the results with the three phase schemes, SP, FLP and VLP, in Table 4.1, we note that the set of scheme FLP resulted in excised schedule templates that exhibit flow imbalance more than the other two schemes. Moreover when flow balance correction had to be applied, it was always for schedules that were larger than a single T period, i.e., who had some queueing dynamics influencing successive T periods. Again, indirect evidence to this end is the fact that when the flow balance corrections (options A or B) were taken, after removing the flow imbalance, certain flows ended up being (unfairly) victimized, i.e. losing the little allocation they had. In addition, the less the overlap (the larger the phase stagger between nodes) the links remain active only briefly, resulting in smaller rates assigned to flows traversing them, as they become more "bottlenecked" leading to smaller (and hence problematic to schedule as just mentioned) rate allocations.

Table 4.1. Simulation results with PT-5

Coding	#F or Gap	Pre-Sim	Numerical	Fairness	$T_{schedule}/T$	Mismatch	Err	Scheme	Notes
NC0	2	0.125	0.125	1	1	0	0	SP	
NC1	2	0.1875	0.1875	1	1	0	0	SP	
NC0	4	0.1875	0.1875	1	2	0	0	SP	
NC1	4	0.25	0.25	1	1	0	0	SP	
NC0	6	0.28125	0.28125	1	2	0	0	SP	
NC1	6	0.375	0.375	1	1	0	0	SP	
NC0	8	0.375	0.375	1	2	0	0	SP	
NC1	8	0.5	0.5	1	1	0	0	SP	
NC0	10	0.3515625	0. 3515625	1	8	0	0	SP	
NC1	10	0.46875	0.46875	1	2	0	0	SP	
NC0	12	0.296875	0.328125	0.99	10	0	0	SP	Option B
NC1	12	0.375	0.40625	0.99	1	0	0	SP	Option A
NC0	14	0.328125	0.328125	1	4	0	0	SP	
NC1	14	0.4375	0.4375	1	1	0	0	SP	
NC0	16	0.375	0.375	1	4	0	0	SP	

NC1	16	0.5	0.5	1	1	0	0	SP	
NC0	2	0.145833	0.145833	1	3	0	0	FLP	
NC1	2	0.21875	0.21875	1	2	0	0	FLP	
NC0	4	0.21875	0.21875	1	4	0	0	FLP	
NC1	4	0.291666	0.291666	1	3	0	0	FLP	
NC0	6	0.28125	0.28125	1	2	0	0	FLP	
NC1	6	0.375	0.375	1	1	0	0	FLP	
NC0	8	0.375	0.375	1	2	0	0	FLP	
NC1	8	0.5	0.5	1	1	0	0	FLP	
NC0	10	0.390625	0.4296875	0.99	20	0	0	FLP	Option B
NC1	10	0.503125	0.520833	0.99	20	0	0	FLP	Option B
NC0	12	0.3515625	0.375	0.96	4	0	0	FLP	Option B
NC1	12	0.421875	0.421875	1	24	0	0	FLP	
NC0	14	0.328125	0.328125	1	4	0	0	FLP	
NC1	14	0.4375	0.4375	1	1	0	0	FLP	
NC0	16	0.375	0.375	1	16	0	0	FLP	
NC1	16	0.5	0.5	1	1	0	0	FLP	
NC0	2	0.375	0.375	1	16	0	0	VLP	
NC1	2	0.5	0.5	1	1	0	0	VLP	
NC0	4	0.375	0.375	1	8	0	0	VLP	
NC1	4	0.5	0.5	1	1	0	0	VLP	
NC0	6	0.5	0.5	1	1	0	0	VLP	
NC1	6	0.5	0.5	1	1	0	0	VLP	
NC0	8	0.3125	0.3125	1	32	0	0	VLP	
NC1	8	0.3125	0.3125	1	32	0	0	VLP	
NC0	10	0.125	0.125	1	4	0	0	VLP	
NC1	10	0.125	0.125	1	4	0	0	VLP	

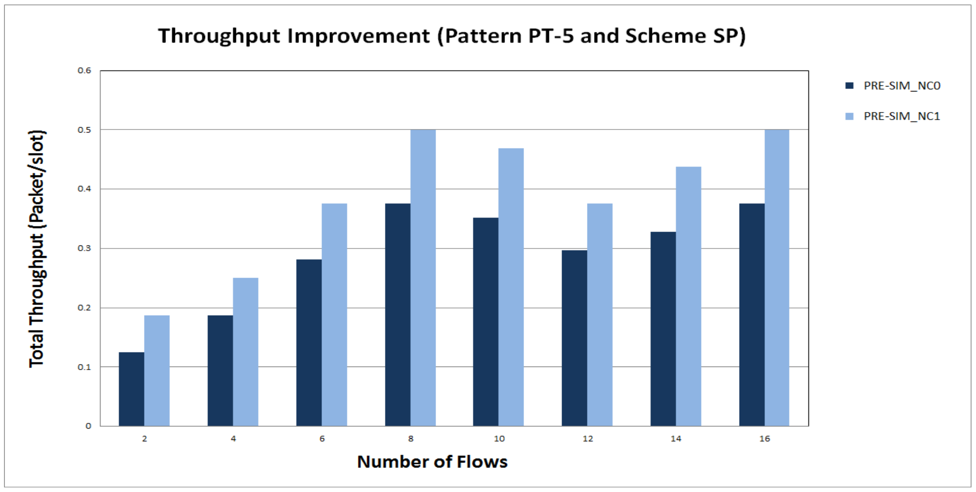
Based on the results in Table 4.1, we create Figures 4.18.a, 4.18.b and 4.18.c, which show the throughput improvement by NC in SP, FLP and VLP, respectively. In the bar charts, X-axis represents the number of flows (with SP and FLP) or the gaps between two neighbors' duty-cycles (with VLP), and Y-axis represents the total throughput (in packets/slot). The **PRE-SIM_NC0** and **PRE-SIM_NC1** indicate **Pre-Sim** values without and with NC, respectively, from which we capture the following.

For the sake of comparison, we define the coding gain as the ratio of the number of transmissions required by the non-coding approach to those by the delay coding approach to deliver the same set of packets. This coding gain can be approximately calculated by the ratio of the total throughput with NC to that without NC of the same network configuration.

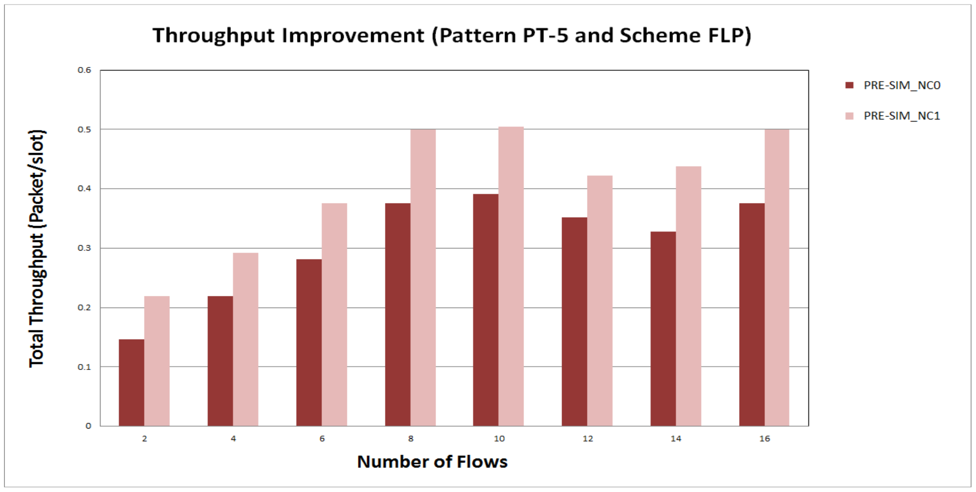
From results with schemes SP, FLP and VLP in Figures 4.18, the improvement from the case without NC to that with NC can increase up to 50%, which is corresponding to the coding gain of 1.5. This is in good agreement with the results by Katti et al. [22], in which the coding gain in a chain topology with two flows in reverse directions and N nodes is approximately calculated as 2N/(N+1). This gain is limited by the upper-bound of 2 when the chain length grows and the interference among simultaneous transmissions is not included in the calculation. Note that in Figure 4.18.c, due to no overlap among the duty-cycles of an intermediate node and its two neighbors and therefore NC not happening at the intermediate node, there is no coding gain when the gap between two neighbors' duty-cycles inceases from 6 to 10.

To investigate the impact of NC on the average packet delay, we performed more simulations and made Figures 4.19.a, 4.19.b and 4.19.c, which show the average packet delay when NC is applied in SP, FLP and VLP, respectively. In the bar charts, X-axis represents the number of flows (with SP and FLP) or the gaps between two neighbors' duty-cycles (with VLP), and Y-axis represents the average packet delay (in slots/packet). The **DELAY_NC0** and **DELAY_NC1** indicate the average packet delay without and with NC, respectively, from which we capture the following.

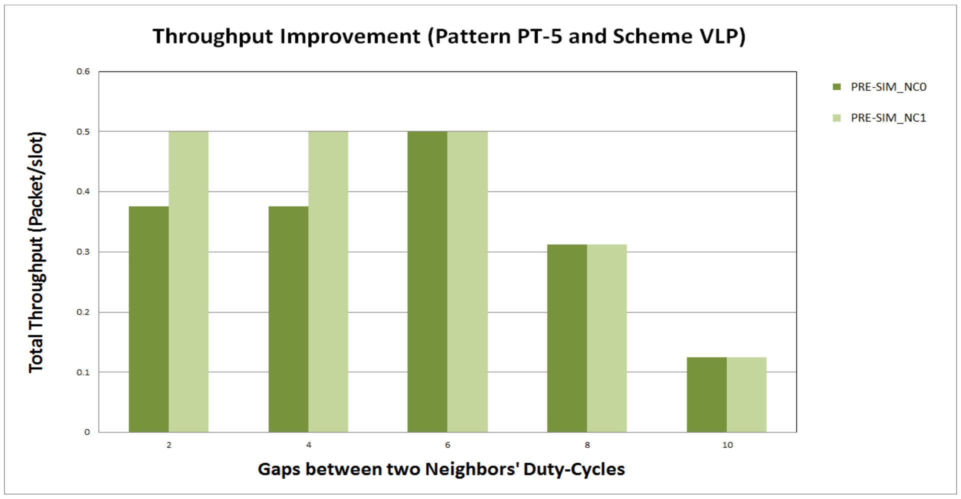
In Figures 4.19.a and 4.19.b, when the number of flows increases, the interference among simultaneous transmissions tends to increase, which results in more congestion and hence packet delay. Also, NC definitely helps to significantly reduce the average packet delay. This is because with greedy sources, the combination of two transmissions into one by NC helps to release the congestion. In Figure 4.19.c, when the gap between two neighbors' duty-cycles increases, without NC the congestion and delay also increase because of smaller overlap between the two neighbors' duty-cycles.



(a)

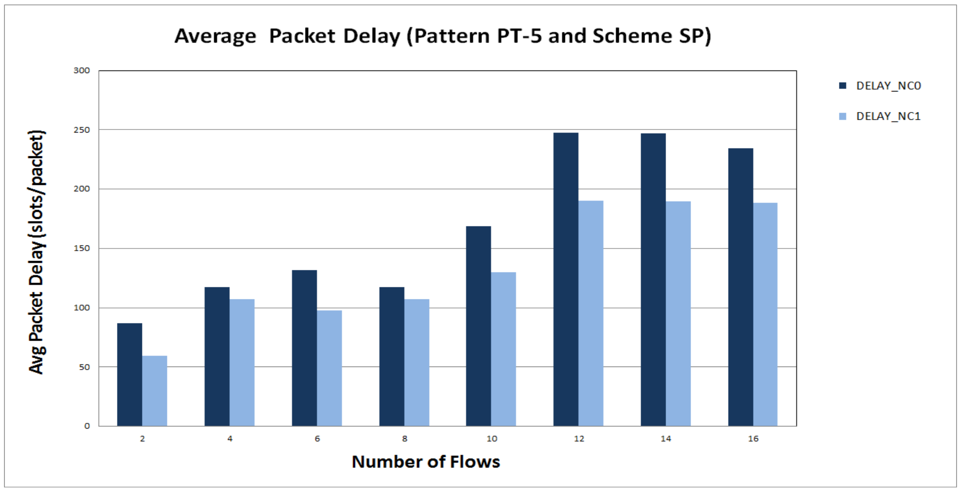


(b)

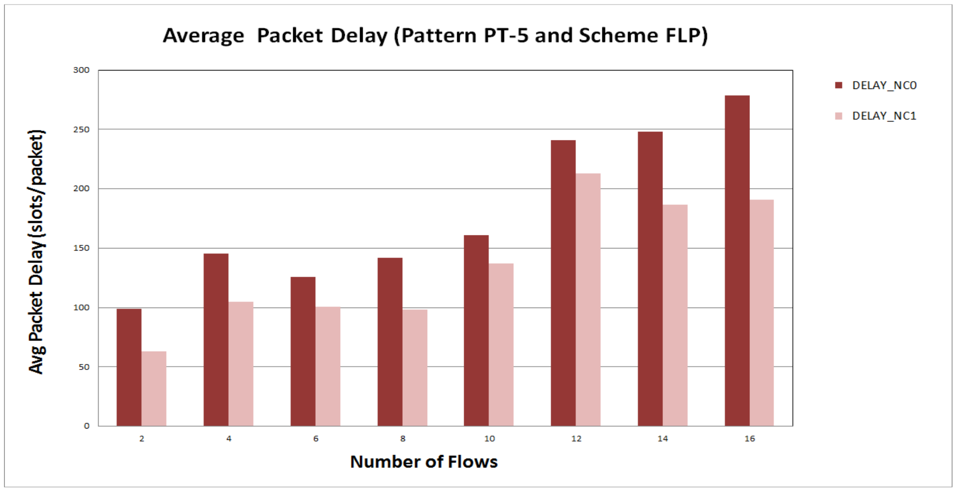


(c)

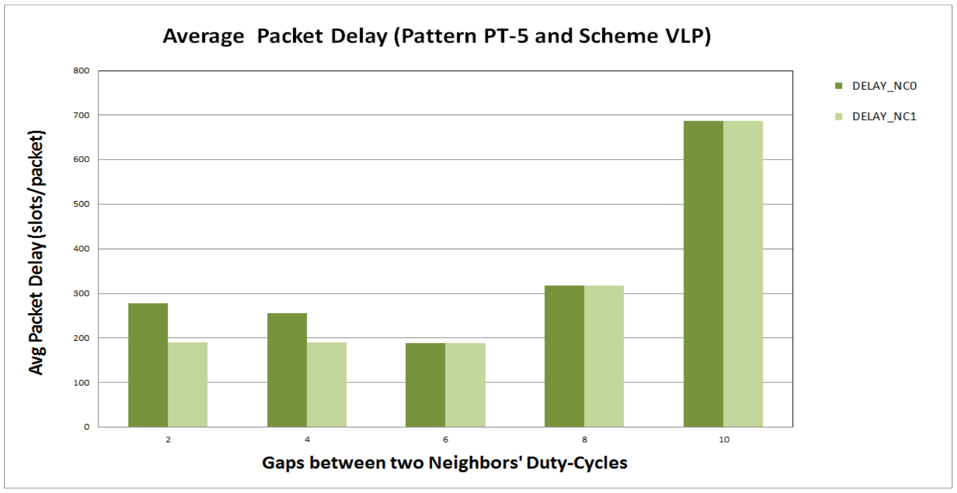
Figure 4.18. Throughput improvement in 4x4 grid with pattern PT-5



(a)



(b)



(c)

Figure 4.19. Average packet delay in 4x4 grid with pattern PT-5

4.3.2 ENERGY SAVINGS

To evaluate the energy saving, we calculate the energy cost per unit of throughput (packets/slot) for the same DC configuration of nodes in the network with NC and without NC, and then compare the two costs. Essentially, we want to know with one unit of throughput received at all the sinks whether energy cost spent in the network with NC is greater than that spent in the network without NC. We expect the normalized cost per throughput of the network with NC is less than that of the network without NC. The main reason is that in the network with NC one transmission can be received by up to two receiver nodes while in the network without NC one transmission is corresponding to one reception. Therefore, one transmission can be saved in the network with NC. Suppose we have

$$u_{w/\,NC} = \frac{\#del}{T_{schedule}} \tag{4.31}$$

which is the total throughput of the network with NC. It is calculated by the ratio of the number of packets delivered to all the destinations, i.e., $\#del$, during the constructed schedule, i.e., $T_{schedule}$.

If we represent by x and y the power consumption normalized with a energy unit, e.g., Joule (J), corresponding to a transmission and a reception then the definition of the normalized energy cost per unit time becomes

$$C_{w/\,NC} = \frac{x.\#tx + y.\#rx}{T_{schedule}} \tag{4.32}$$

in which $\#tx$ and $\#rx$ are the respective numbers of transmissions and receptions during the constructed schedule $T_{schedule}$ and *x:y* is the ratio of power consumption corresponding to *Transmission:Reception=1.5:1.0.*

Hence, we can approximately calculate the normalized energy cost per throughput of the network with NC from (4.31) and (4.32) as follows.

$$\frac{C_{w/\,NC}}{u_{w/\,NC}} = \left(\frac{x.\#tx + y.\#rx}{T_{schedule}}\right).\left(\frac{T_{schedule}}{\#del}\right) = \frac{x.\#tx + y.\#rx}{\#del} \tag{4.33}$$

Similarly, we can approximately calculate the normalized energy cost per throughput of the network without NC as follows.

$$\frac{C_{w/o\,NC}}{u_{w/o\,NC}} = \left(\frac{x.\#tx' + y.\#rx'}{T_{schedule}}\right).\left(\frac{T_{schedule}}{\#del'}\right) = \frac{x.\#tx' + y.\#rx'}{\#del'} \tag{4.34}$$

in which $u_{w/o\,NC}$ and $C_{w/o\,NC}$ are the total throughput and the normalized energy cost per unit time during the constructed schedule $T_{schedule}$ (i.e., without NC). Also, $\#tx'$ and $\#rx'$ are the respective numbers of transmissions and receptions, and $\#del'$ is the number of packets delivered to all the destinations during $T_{schedule}$.

Note that working under the scheduling algorithm, duty-cycled nodes are almost always transmitting or receiving packets during active periods. The nodes receiving or transmitting a sub-flow that are not scheduled in a timeslot are to be deactivated transceivers to save energy in the timeslot. Hence, the calculation of energy saving in (4.33) and (4.34) does not include the energy consumption during idling periods as such idle periods no longer exist.

Figures 4.20.a, 4.20.b and 4.20.c show the normalized energy costs per throughput unit with NC (legend EGY_COSTS_NC1) and without NC (legend EGY_COSTS_NC0) under the three phase schemes, SP, FLP and VLP, respectively. With two schemes SP and FLP, the energy consumption of the configuration with NC is reduced by 23% compared to without NC when the number of flows increases from 2 to 16. Meanwhile, with scheme VLP, the energy consumption of the configuration with NC decreases around 23% that wihout NC when the gap between two neighbors' duty-cycles increases from 2 to 4. When the gap increases from 6 to 10, the energy consumptions with and without NC are the same because of no NC. Note that we can approximately verify the gap from the simulation results by substituting the ratio *x:y=1.5:1* into (4.33) and (4.34) as follows.

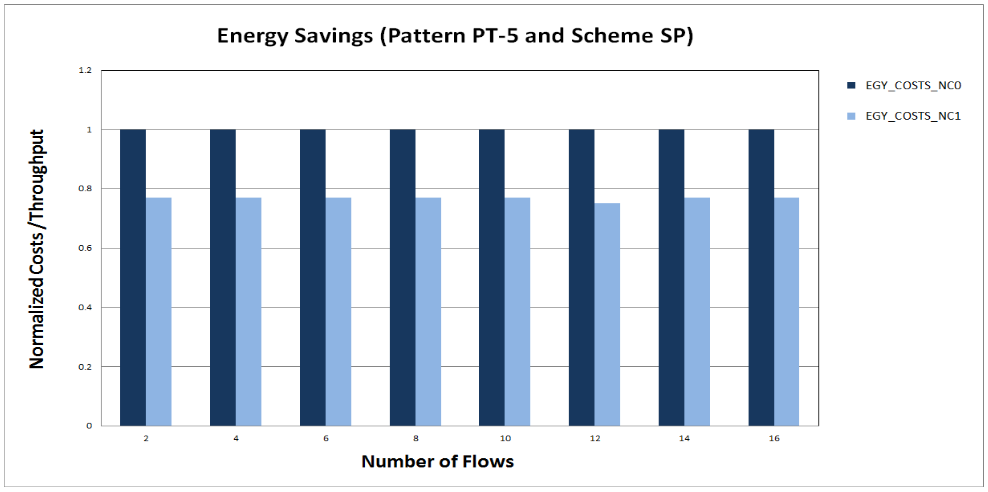
$$\frac{c_{w/\,NC}}{u_{w/\,NC}} = \frac{1.5.\#tx + \#rx}{\#del} \qquad (4.35)$$

$$\frac{c_{w/o\,NC}}{u_{w/o\,NC}} = \frac{1.5.\#tx' + \#rx'}{\#del'} \qquad (4.36)$$

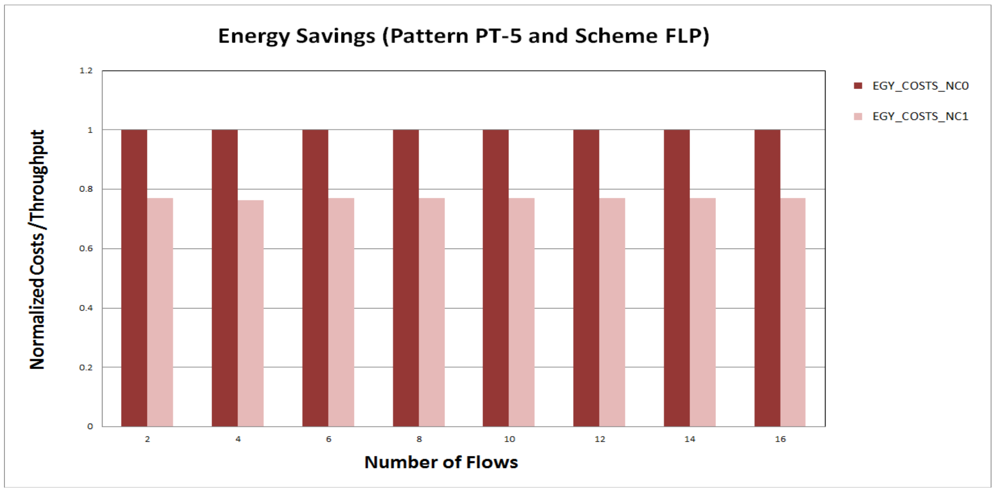
We observe that with the same number of packets delivered to the destinations per time unit in both cases with and without NC, i.e., ($\#del = \#del'$), the number of transmitted packets is equal to that of received packets in the case without NC, i.e., ($\#tx' = \#rx'$), while the number of transmitted packets is always greater than half that of received packets in the case with NC, i.e., ($\#tx > 0.5.\#rx$), thanks to the pairwise-XOR network coding. We also observe that the numbers of received packets per time unit in both cases with and without NC are equal, i.e., $\#rx = \#rx'$, so that the same thoughput can be attained. After applying the observations into (4.35) and (4.36) we have

$$\frac{c_{w/\,NC}}{u_{w/\,NC}} : \frac{c_{w/o\,NC}}{u_{w/o\,NC}} > 0.7 \qquad (4.37)$$

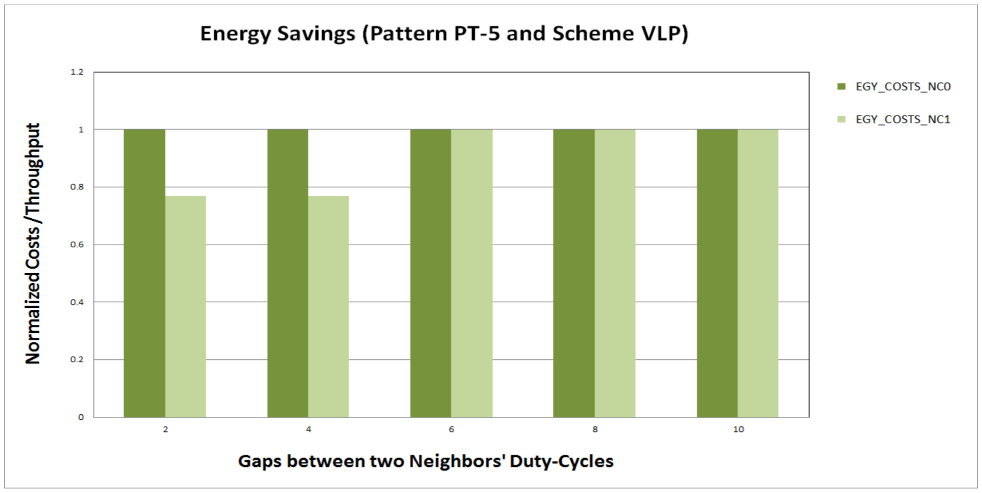
Obviouslly, the normalized energy costs per throughput unit with and without NC in Figures 4.20.a, 4.20.b and 4.20.c are conformable to (4.37).



(a)



(b)



(c)

Figure 4.20. Energy Savings from pattern PT-5 and schemes SP, FLP and VLP

4.4 CHAPTER CONCLUSIONS

In this chapter, we attempt to solve the problem of throughput reduction due to duty-cycling by introducing pairwise XOR NC. We introduce a delay coding scheme to ensure there are enough data packets available to perform NC. By applying NC we are aware that the max-min fair rate allocation of the flows changes as well. This is because additional constraints are introduced to reflect the fact that the data of a flow now have two options when it comes to be forwarded at an intermediate node: they can be forwarded as they are, or as coded packets with the packets of another flow.

As in Chapter 3, we perform the simulation-based process to determine that we have reached the periodic steady state. Particular to the case of NC is that we make use of the delay coding approach to further attain coding benefits, by giving coded sub-flows higher priority than non-coded sub-flows of the same weight. Even in the presence of NC, we still have strong evidence that, due to the periodicity of all input factors, part of which is the periodic behavior of the transmission decisions, a similar periodicity is exhibited by the steady state.

The process of manipulating the excised schedule template is more elaborate in the case of NC. Without NC, an active non-coded sub-flow or spot in the template is only used for a single routing path, whereas with NC, an active coded sub-flow or spot in the template is possibly used for up to two single routing paths. Hence, we introduce additional active states for an active coded sub-flow in the flow balance process, and we treat it accordingly if it needs to be trimmed during the flow balance adjustment performed on the excised schedule template.

A key assumption used in this chapter is that the pairs of flows that are coded together have been decided a-priori. In fact, there are multiple options for coding combinations, and this will be revisited in Chapter 5 where we will discuss extensions and generalizations of the techniques developed in Chapters 3 and 4.

CHAPTER 5
EXTENSIONS AND GENERALIZATIONS

5.1 NON-GREEDY TRAFFIC AND ENERGY SAVINGS

In Chapters 3 and 4, we described a methodology that relies on the assumption that the traffic sources are greedy. As part of the computation of the flow rates, a certain rate is eventually assigned to each flow and implemented by the schedule. A particular concern is what happens when the traffic generated by a source is lower than the rate implemented by the schedule. The intent of STDMA is to allow the nodes to know exactly when to transmit or receive, thus avoid idle listening. In the event of less traffic than the schedule assumes, it is possible that a node anticipates to receive but the reception does not happen, hence energy is wasted, as the corresponding slot becomes equivalently an idle listening slot; precisely what we wanted to avoid in the first place. (Note that a similar concern for the side of transmission does not exist.)

In this section we detail how an additional scheme can be used to avoid such idle listening in the event when traffic is less than what the schedule is constructed to carry. Effectively, the additional scheme involves the "deactiviation" of slots in the template, from the perspective of the receiver. We start with the assumption that all the nodes are synchronized, a requirement that is needed anyway for a slotted system. The schedule template is assumed to have been downloaded to all nodes, i.e., it is common knowledge. The proposed approach follows the three main steps:

(1) The source plays the role of a traffic regulator of the given flow f in the sense that it is responsible for collecting and measuring the actual traffic load λ_f. With the traffic load, the schedule template's length $\mathrm{T}_{\mathrm{Schedule}}$ and the flow's rate r_f as

calculated during the max-min fairness step (and represented by the schedule), the source can determine a duration T_{λ_f}, as follows.

$$\mathrm{T}_{\lambda_f} = r_f \mathrm{T}_{\mathrm{Schedule}} / \lambda_f \tag{5.1}$$

(2) Under the assumption that the traffic load λ_f is less than the flow's rate r_f, T_{λ_f} is greater than $\mathrm{T}_{\mathrm{Schedule}}$, the source will send to its downstream node an indication (described next) to turn off reception for duration T_{λ_f} while the source collects enough packets in order to form a burst of rate r_f over an entire schedule's length $\mathrm{T}_{\mathrm{Schedule}}$ before transmitting it to the downstream node. In other words, the source forms bursts that can sustain the rate r_f over limited time lengths (length equal to $\mathrm{T}_{\mathrm{Schedule}}$). The process is the inverse of a smoothing traffic regulator, since its explicit purpose is to make traffic bursty. The downstream node relays to its downstream nodes along the routing path the same indication and burst until the destination of the flow is reached. (Note that while transmitting to the downstream node, the source still collects packets from its traffic source.)

(3) After a duration $\mathrm{T}_{\mathrm{Schedule}}$, because the traffic is less than r_f all the downstream nodes along the routing path until the sink have to turn off reception for duration $(\mathrm{T}_{\lambda_f} - \mathrm{T}_{\mathrm{Schedule}})$ so that the source has more time for collecting enough packets from its traffic source to form the next burst. All the downstream nodes repeat this step (3) until they receive a new turning-off indication from the source.

A few clarifications are in order. First, the technique outlined here is to be applied to each flow separately. From the point of view of the receiver, the scheme dictates how long it should assume there will be no traffic from the upstream node of that subflow. Second, the means to indicate to downstream nodes that they should avoid idle listening, i.e., the indication sent from upstream/source node can be either implicit or explicit. For a simple implicit signaling works as follows: if the receiver has received no packet in the first (in $\mathrm{T}_{\mathrm{Schedule}}$ order) inbound slot of flow f, then all the following slots in $\mathrm{T}_{\mathrm{Schedule}}$ for flow f in the schedule will be assumed to also have no traffic, and hence the receiver will switch off reception during those slots. This signaling wastes one slot for idle listening. A better mechanism (which can be made more reliable in the presence of noise/interference using low rate coding) is that of explicit signaling, where a single slot is sacrificed to convey the burst characteristics constructed by the source. The advantage of explicit signaling is that it can contain more information, e.g., T_{λ_f}.

Example 5.1. To understand the slot deactivation scheme, we will consider the simple network of Figures 5.1 and 5.2, without and with NC, respectively and construct the corresponding schedules via simulation-based periodic steady state excision.

The duty-cycled network has four nodes $n \in \mathcal{N} = \{1, 2, 3, 4\}$, six directed wireless links in the link set $\mathcal{L} = \{l_1, l_2, l_3, l_4, l_5, l_6\}$ shown in Figure 5.1.a.1, and two flows $f_1, f_2 \in \mathcal{F}$ traversing, respectively, the paths $\{l_1, l_2, l_3\}$ and $\{l_5, l_6\}$ indicated in Figure 5.1.b. The DC-configurations, i.e., $\langle \phi_\mathrm{n}, \alpha_\mathrm{n}, \mathrm{T_n} \rangle$, of the nodes is as shown in Figure 5.1.a.2, in which period T is divided into two stages $\mathcal{T}^{(k)}$, in units of slots, with $k \in \mathcal{K} = \{1,2\}$, such that in each of these stages the ON/OFF state of the nodes is unchanged.

In Figure 5.1.b, during each stage k, flow f_1 without NC is represented by three sub-flow components $r_{f_1}^{(k)}(l_1)$, $r_{f_1}^{(k)}(l_2)$ and $r_{f_1}^{(k)}(l_3)$, and flow f_2 without NC is represented by two sub-flow components $r_{f_2}^{(k)}(l_5)$ and $r_{f_2}^{(k)}(l_6)$. Hence, the maximal cliques of the sub-flow contention graph (CG) which is made up from all sub-flows competing in each stage, i.e., $\mathcal{T}^{(1)}$ and $\mathcal{T}^{(2)}$, are shown in Figures 5.1.c.1 and 5.1.c.2, respectively.

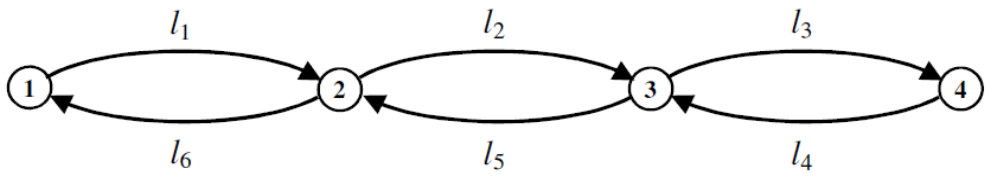
In Figure 5.2.b, during each stage k, flow f_1 with NC is represented by four sub-flow components $r_{f_1}^{(k)}(l_1)$, $r_{f_1}^{(k)}(l_2)$, $r_{f_1,f_2}^{(k)}(l_2, l_6)$, and $r_{f_1}^{(k)}(l_3)$, and flow f_2 with NC is represented by three sub-flow components $r_{f_2}^{(k)}(l_5)$, $r_{f_2}^{(k)}(l_6)$, and $r_{f_1,f_2}^{(k)}(l_2, l_6)$. Hence, the maximal cliques of the sub-flow contention graph (CG) which is made up from all sub-flows competing in each stage, i.e., $\mathcal{T}^{(1)}$ and $\mathcal{T}^{(2)}$, are shown in Figures 5.2.c.1 and 5.2.c.2, respectively.

Without NC, from the MP algorithm associated with lex-max formulations, we have the following flow rates.

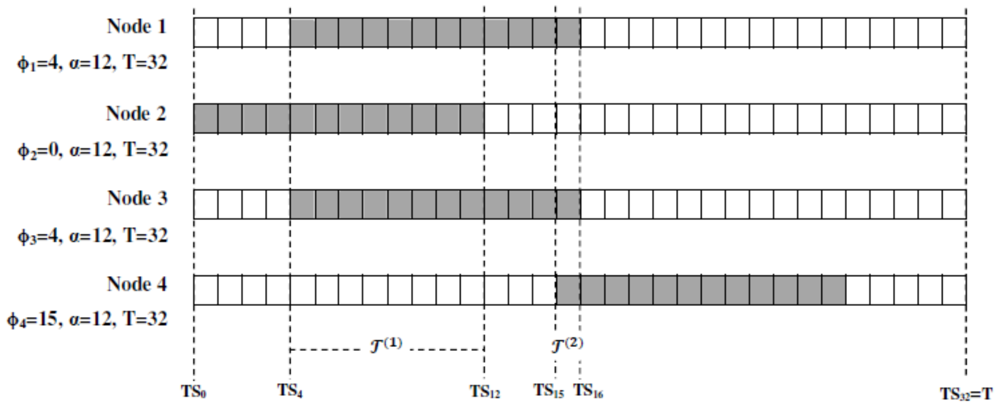
$$\begin{aligned}
&r_1 = 0.03125,\ r_2 = 0.09375;\\
&r_{f_1}^{(1)}(l_1) = 0.03125,\ r_{f_1}^{(1)}(l_2) = 0.03125,\ r_{f_1}^{(1)}(l_3) = 0,\\
&r_{f_2}^{(1)}(l_5) = 0.09375,\ r_{f_2}^{(1)}(l_6) = 0.09375;\\
&r_{f_1}^{(2)}(l_1) = 0,\ r_{f_1}^{(2)}(l_2) = 0,\ r_{f_1}^{(2)}(l_3) = 0.03125,\\
&r_{f_2}^{(2)}(l_5) = 0,\ r_{f_2}^{(2)}(l_6) = 0;
\end{aligned} \tag{5.2}$$

With NC, from the MP algorithm associated with lex-max formulations, we have the following flow rates.

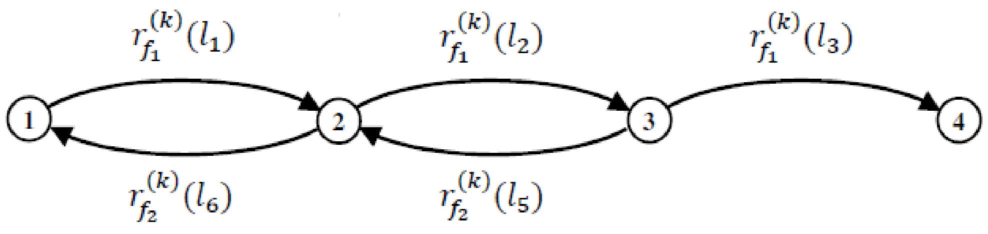
$$\begin{aligned}
&r_1 = 0.03125,\ r_2 = 0.109375;\\
&r_{f_1}^{(1)}(l_1) = 0.03125,\ r_{f_1}^{(1)}(l_2) = 0,\ r_{f_1,f_2}^{(1)}(l_2, l_6) = 0.03125,\ r_{f_1}^{(1)}(l_3) = 0,\ r_{f_2}^{(1)}(l_5) =\\
&0.109375,\ r_{f_2}^{(1)}(l_6) = 0.078125;\\
&r_{f_1}^{(2)}(l_1) = 0,\ r_{f_1}^{(2)}(l_2) = 0,\quad r_{f_1,f_2}^{(2)}(l_2, l_6) = 0,\quad r_{f_1}^{(2)}(l_3) = 0.03125,\\
&r_{f_2}^{(2)}(l_5) = 0,\ r_{f_2}^{(2)}(l_6) = 0;
\end{aligned} \tag{5.3}$$



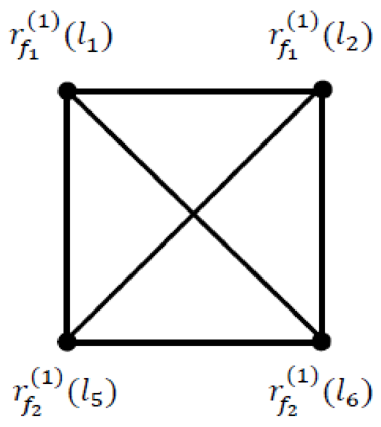
(a.1) Link topology


Node 1
ϕ_1=4, α=12, T=32
Node 2
ϕ_2=0, α=12, T=32
Node 3
ϕ_3=4, α=12, T=32
Node 4
ϕ_4=15, α=12, T=32
$\mathcal{T}^{(1)}$
$\mathcal{T}^{(2)}$
TS_0
TS_4
TS_{12}
TS_{15}
TS_{16}
TS_{32}=T

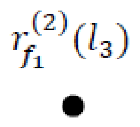
(a.2) States ON/OFF of nodes (ON slots are gray)



(b) Multi-hop flows

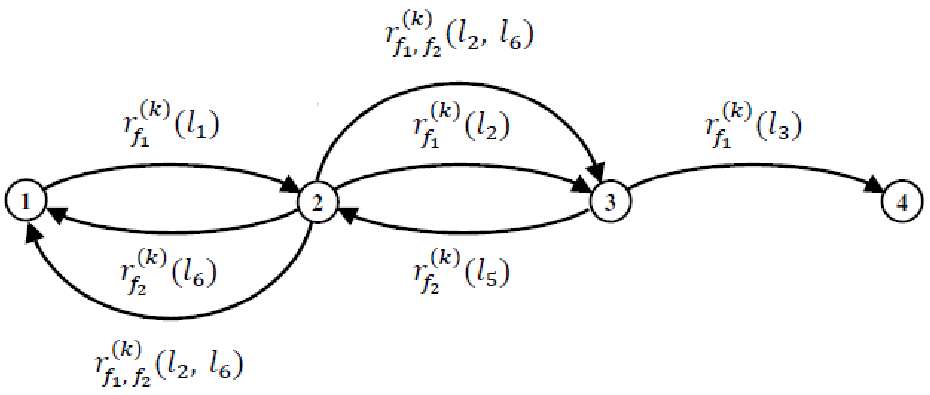


(c.1) CG in $\mathcal{T}^{(1)}$

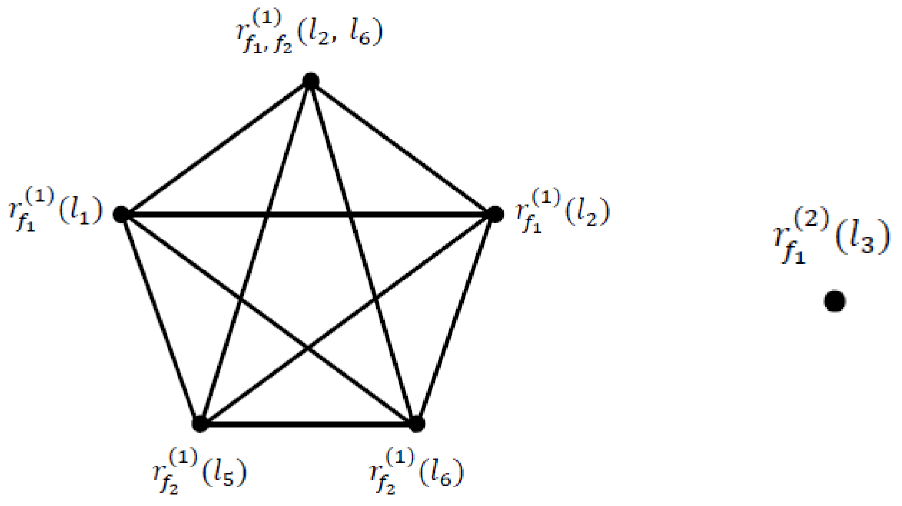


(c.2) CG in $\mathcal{T}^{(2)}$

Figure 5.1. An illustrative example for the non-greedy traffic *without* NC (CG = Contention Graph)



(b) Multi-hop flows



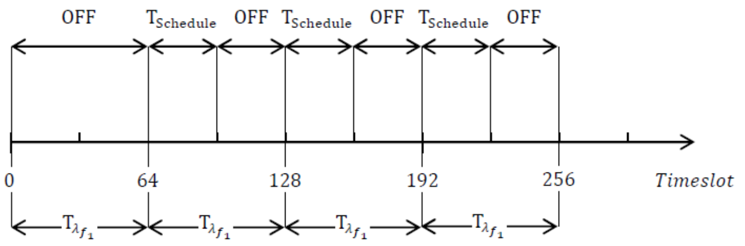
(c.1) CG in $\mathcal{T}^{(1)}$ (c.2) CG in $\mathcal{T}^{(2)}$

Figure 5.2. An illustrative example for the non-greedy traffic *with* NC (CG = Contention Graph)

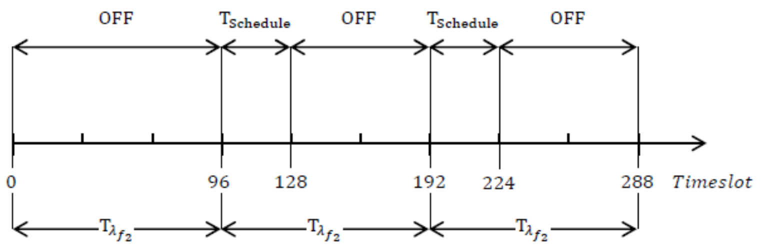
Assume now for the purpose of demonstrating the impact of the slot deactivation scheme, that the traffic load λ_{f_1} of flow f_1 equals to 50% of the flow's rate r_{f_1} shown by $r_{f_1}^{(k)}(l_1)$, $r_{f_1}^{(k)}(l_2)$ and $r_{f_1}^{(k)}(l_3)$ from (5.2) corresponding to the constructed schedule of length, $\mathrm{T}_{\mathrm{Schedule}} = 32$ slots, shown in Figure 5.3.a. From (5.1) and the above approach, we have the slot deactivation scheme shown in Figure 5.3.b for downstream nodes 2, 3 and 4 after receiving an explicit indication with information $\mathrm{T}_{\lambda_{f_1}} = 64$ slots from the source node 1 of flow f_1.

t	$r_{f_1}^{(k)}(l_1)$	$r_{f_1}^{(k)}(l_2)$	$r_{f_1}^{(k)}(l_3)$	$r_{f_2}^{(k)}(l_5)$	$r_{f_2}^{(k)}(l_6)$
1	○	○	○	○	○
2	○	○	○	○	○
3	○	○	○	○	○
4	○	○	○	○	○
5	○	○	○	●	○
6	○	○	○	○	●
7	○	○	○	●	○
8	○	○	○	○	●
9	●	○	○	○	○
10	○	●	○	○	○
11	○	○	○	●	○
12	○	○	○	○	●
13	○	○	○	○	○
14	○	○	○	○	○
15	○	○	○	○	○
16	○	○	●	○	○
17	○	○	○	○	○
18	○	○	○	○	○
19	○	○	○	○	○
…	○	○	○	○	○
32	○	○	○	○	○

(a) Schedule constructed in Example 5.1 (*without* NC)



(b) Turning-off reception of nodes 2, 3 and 4 with λ_{f_1} equal to 50% of r_{f_1}



(c) Turning-off reception of nodes 2 and 1 with λ_{f_2} equal to 33.3% of r_{f_2}

Figure 5.3. Illustration of the slot deactivation scheme (*without* NC)

Similarly, when the traffic load λ_{f_2} of flow f_2 equals to 33.3% of the flow's rate r_{f_2}, which is shown by $r_{f_2}^{(k)}(l_5)$ and $r_{f_2}^{(k)}(l_6)$ from (5.2) corresponding to the constructed schedule of length, $\mathrm{T_{Schedule}} = 32$ slots, shown in Figure 5.3.a. From (5.1) and the above approach, we have the slot deactivation scheme shown in Figure 5.3.c for downstream nodes 2 and 1 after receiving an explicite indication with information $\mathrm{T}_{\lambda_{f_2}} = 96$ slots from the source node 3 of flow f_2.

Note that in Figures 5.3.b and 5.3.c, we assume that two source nodes, 1 and 3, of two respective flows, f_1 and f_2, start their slot deactivation scheme at the same time, i.e., timeslot 0, as shown by the timeslot axis (in timeslots). Also, the OFF duration indicates that the receptions at the respective downstream nodes are off while the $\mathrm{T_{schedule}}$ indicates the receptions of the respective downstream nodes are active corresponding to the template.

We observe that without NC, the slot deactivation scheme for the downstream nodes along a flow, e.g., f_1, do not impact the operation of the scheme for the downstream nodes of another flow, e.g., f_2. This is because each scheduled spot in the template is dedicated just to a single sub-flow of a flow. In other words, active spots scheduled for nodes along a flow's path in the template guarantee the flow can transmit to its sink all the packets that have been generated from its source.

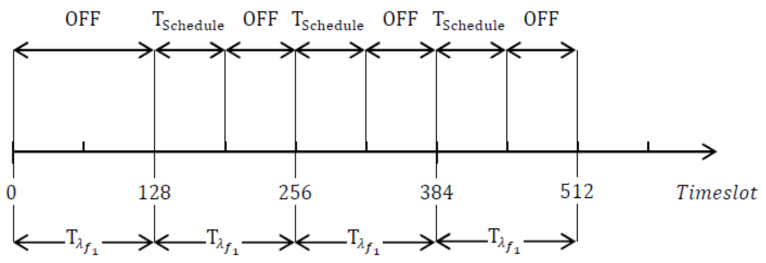
A question arises at this point is that whether the slot deactivation approach described above still works with NC, in which an active spot in the template may be scheduled for two flows, i.e., an active spot for a coded sub-flow. To answer this question, let us reconsider Example 5.1 with NC, which has the template with $\mathrm{T_{Schedule}} = 64$ slots shown in Figure 5.4. Note that the template with NC is different from that without NC in Figure 5.3.a that the former has an additional column for non-coded sub-flow $r_{f_1,f_2}^{(k)}(l_2, l_6)$ from two flows f_1 and f_2. In addition, $r_{f_1}^{(k)}(l_2)$ is not scheduled because f_1 (with smaller numerical rate) is entirely coded by f_2 (with greater numerical rate).

t	$r_{f_1}^{(k)}(l_1)$	$r_{f_1}^{(k)}(l_2)$	$r_{f_1}^{(k)}(l_3)$	$r_{f_1,f_2}^{(k)}(l_2,l_6)$	$r_{f_2}^{(k)}(l_5)$	$r_{f_2}^{(k)}(l_6)$
1	○	○	○	○	○	○
2	○	○	○	○	○	○
3	○	○	○	○	○	○
4	○	○	○	○	○	○
5	○	○	○	○	●	○
6	○	○	○	○	○	●
7	○	○	○	○	●	○
8	○	○	○	○	○	●
9	○	○	○	○	●	○
10	●	○	○	○	○	○
11	○	○	○	🔴	○	○
12	○	○	○	○	●	○
13	○	○	○	○	○	○
14	○	○	○	○	○	○
15	○	○	○	○	○	○
16	○	○	●	○	○	○
17	○	○	○	○	○	○
18	○	○	○	○	○	○
19	○	○	○	○	○	○
…	○	○	○	○	○	○
32	○	○	○	○	○	○
33	○	○	○	○	○	○
34	○	○	○	○	○	○
35	○	○	○	○	○	○
36	○	○	○	○	○	○
37	○	○	○	○	○	●
38	○	○	○	○	●	○
39	○	○	○	○	○	●
40	○	○	○	○	●	○
41	●	○	○	○	○	○
42	○	○	○	🔴	○	○
43	○	○	○	○	●	○
44	○	○	○	○	○	●
45	○	○	○	○	○	○
46	○	○	○	○	○	○
47	○	○	○	○	○	○
48	○	○	●	○	○	○
49	○	○	○	○	○	○
50	○	○	○	○	○	○
51	○	○	○	○	○	○
…	○	○	○	○	○	○
64	○	○	○	○	○	○

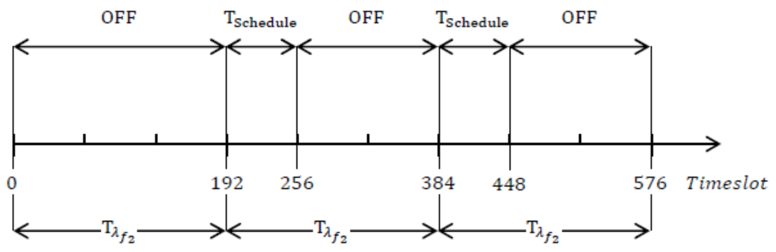
Figure 5.4. Schedule constructed in Example 5.1 (*with* NC)

Similar to the case without NC, we assume that the traffic load $\boldsymbol{\lambda_{f_1}}$ of flow $\boldsymbol{f_1}$ equals to 50% of the flow's rate $\boldsymbol{r_{f_1}}$and the traffic load $\boldsymbol{\lambda_{f_2}}$ of flow $\boldsymbol{f_2}$ equals to 33.3% of the flow's rate $\boldsymbol{r_{f_2}}$. From (5.1) and the above approach, we have the slot deactivation scheme shown in Figure 5.5.a for downstream nodes 2, 3 and 4 after receiving an explicit indication with information $\mathbf{T}_{\boldsymbol{\lambda_{f_1}}} = \mathbf{128}$ slots from the source node 1 of flow $\boldsymbol{f_1}$. Also, from (5.1) and the above approach, we have the slot deactivaton scheme shown in Figure 5.5.b for downstream nodes 2 and 1 after receiving an explicit indication with information $\mathbf{T}_{\boldsymbol{\lambda_{f_2}}} = \mathbf{192}$ slots from the source node 3 of flow $\boldsymbol{f_2}$.

With NC, we again see the slot deactivation scheme for the downstream nodes along a flow, e.g., f_1, does not impact the scheme for the downstream nodes along the other flow, e.g., f_2. This is because each scheduled spot of a coded sub-flow in the template can carry *both* flows, i.e., no matter whether one of the two receivers turns off reception. Namely, the active spots scheduled for nodes along a flow's path in the template guarantee the flow can transmit all its packets to the destination that have been generated from its source.



(a) Turning-off reception of nodes 2, 3 and 4 with λ_{f_1}equal to 50% of r_{f_1}



(b) Turning-off reception of nodes 2 and 1 with λ_{f_2}equal to 33.3% of r_{f_2}

Figure 5.5. Illustration of the slot deactivation scheme (*with* NC)

$\boldsymbol{t}$	$\boldsymbol{r^{(k)}_{f_1}(l_1)}$	$\boldsymbol{r^{(k)}_{f_1}(l_2)}$	$\boldsymbol{r^{(k)}_{f_1}(l_3)}$	$\boldsymbol{r^{(k)}_{f_1,f_2}(l_2, l_6)}$	$\boldsymbol{r^{(k)}_{f_2}(l_5)}$	$\boldsymbol{r^{(k)}_{f_2}(l_6)}$
1	○	○	○	○	○	○
…	○	○	○	○	○	○
5	○	○	○	○	○	○
6	○	○	○	○	○	○
7	○	○	○	○	○	○
8	○	○	○	○	○	○
9	○	○	○	○	○	○
10	●	○	○	○	○	○
11	○	○	○	🔵	○	○
12	○	○	○	○	○	○
13	○	○	○	○	○	○
14	○	○	○	○	○	○
15	○	○	○	○	○	○
16	○	○	●	○	○	○
…	○	○	○	○	○	○
32	○	○	○	○	○	○
33	○	○	○	○	○	○
…	○	○	○	○	○	○
37	○	○	○	○	○	○
38	○	○	○	○	○	○
39	○	○	○	○	○	○
40	○	○	○	○	○	○
41	●	○	○	○	○	○
42	○	○	○	🔵	○	○
43	○	○	○	○	○	○
44	○	○	○	○	○	○
45	○	○	○	○	○	○
46	○	○	○	○	○	○
47	○	○	○	○	○	○
48	○	○	●	○	○	○
…	○	○	○	○	○	○
64	○	○	○	○	○	○

Figure 5.6. Scheduled spots during slots 128 to 192 for flow $f_1$and f_2

For illustration, from Figures 5.5.a and 5.5.b we consider duration $\mathrm{T}_{\mathrm{Schedule}}$ from slots 128 to 192, during which downstream nodes along f_1's path, i.e., nodes 2, 3 and 4, are activated according to the template in Figure 5.4. Meanwhile, downstream nodes along f_2's path, i.e., nodes 2 and 1, turn off reception completely. The slot deactivation scheme for $f_1$and f_2 in this duration is equivalent to the template in Figure 5.6, in which the coded sub-flow's spots in blue indicate f_1's receiver is active while f_2's receiver is off.

t	$r_{f_1}^{(k)}(l_1)$	$r_{f_1}^{(k)}(l_2)$	$r_{f_1}^{(k)}(l_3)$	$r_{f_1,f_2}^{(k)}(l_2,l_6)$	$r_{f_2}^{(k)}(l_5)$	$r_{f_2}^{(k)}(l_6)$
1	○	○	○	○	○	○
…	○	○	○	○	○	○
5	○	○	○	○	●	○
6	○	○	○	○	○	●
7	○	○	○	○	●	○
8	○	○	○	○	○	●
9	○	○	○	○	●	○
10	○	○	○	○	○	○
11	○	○	○	🟢	○	○
12	○	○	○	○	●	○
13	○	○	○	○	○	○
14	○	○	○	○	○	○
15	○	○	○	○	○	○
16	○	○	○	○	○	○
…	○	○	○	○	○	○
32	○	○	○	○	○	○
33	○	○	○	○	○	○
…	○	○	○	○	○	○
37	○	○	○	○	○	●
38	○	○	○	○	●	○
39	○	○	○	○	○	●
40	○	○	○	○	●	○
41	○	○	○	○	○	○
42	○	○	○	🟢	○	○
43	○	○	○	○	●	○
44	○	○	○	○	○	●
45	○	○	○	○	○	○
46	○	○	○	○	○	○
47	○	○	○	○	○	○
48	○	○	○	○	○	○
…	○	○	○	○	○	○
64	○	○	○	○	○	○

Figure 5.7. Scheduled spots during slots 192 to 256 for flow $f_1$and f_2

Similarly, from Figures 5.5.a and 5.5.b we consider duration $\mathrm{T_{Schedule}}$ from slots 192 to 256, during which downstream nodes along f_1's path, i.e., nodes 2, 3 and 4, are turned off completely. Meanwhile, downstream nodes along f_2's path, i.e., nodes 2 and 1, are activated according to the template in Figure 5.4. The slot deactivation scheme for both $f_1$and f_2 in this duration is equivalent to the template in Figure 5.7, in which the coded sub-flow's spots in green indicate f_1's receiver is off while f_2's receiver is active.

5.2 DEALING WITH HETEROGENEOUS DUTY-CYCLES

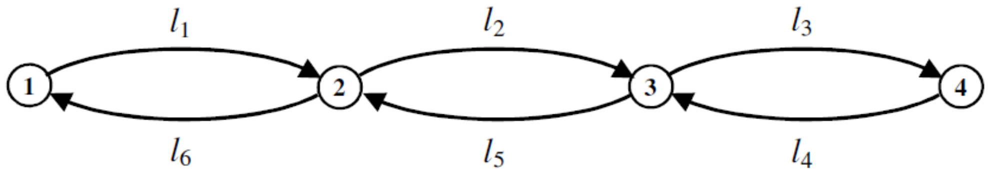
From the beginning of this thesis we have assumed that all nodes have the same cycle length $\mathrm{T_n}$ and the same active period α_n. This was done for reasons of convenience of presentation of our work. In reality, for multiple reasons as described at the beginning of the thesis, those values are expected to be different from one node to another. When this happens, our algorithms and models still apply because of the following reasons.

(1) The time-dependent shortest single-path routing we assumed was calculated in all cases to determine the routing paths for all the flows in our work, is deterministic and is described in [8,9,10] without any constraint on the particular DC structure. Hence, the routing paths can be equally well predetermined and provided as input to our algorithms and models. Note that the cycle period T is then calculated as the least common multiple (LCM) of the cycle periods $\mathrm{T_n}$ of all the nodes. In addition, for the reachability from the source to the sink of a given flow, we still assume that the duty-cycles of nodes are overlapped one after another along the routing path of the flow.

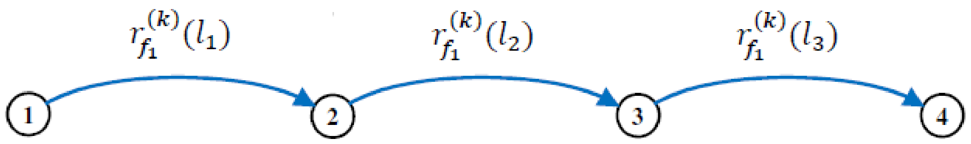
(2) The ON/OFF state of node n is deterministic in each timeslot and characterized by its triplet $\langle \phi_\mathrm{n}, \alpha_\mathrm{n}, \mathrm{T_n} \rangle$. Hence, the states of all the nodes during the LCM period T are also deterministic. As a result, the least number K of stages $\mathcal{T}^{(k)}$, in unit of slots with $k \in \mathcal{K}$, such that during each of these stages the states (ON/OFF) of nodes are unchanged, is also deterministic and trivial to calculate. Since our MP algorithms operate on $\mathrm{T}, \mathcal{K}$ and $\mathcal{T}^{(k)}$, input, all those inputs can be constructed even in the case of heterogeneous DC.

For illustration, Figure 5.8.a shows a DC network with line topology and a single flow as shown in Figure 5.8.b, in which nodes DC are different as shown in Figure 5.8.c. We observe that despite the heterogeneous DC, the time-varying shortest single-routing path (in blue) of flow f_1, including three sub-flows

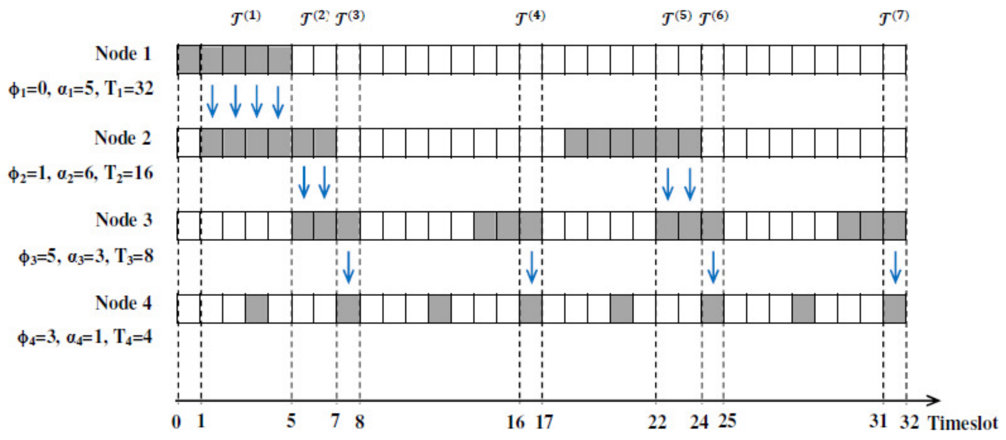
$r_{f_1}^{(k)}(l_1)$, $r_{f_1}^{(k)}(l_2)$ and $r_{f_1}^{(k)}(l_3)$, is deterministic from source 1 to sink 4 through overlaps of nodes 1, 2, 3 and 4. In addition, we find that 7 stages, from $\mathcal{T}^{(1)}$ to $\mathcal{T}^{(7)}$, are deterministic during T, which is the least common multiple of all the cycle periods, from T_1 to T_4.



(a) Link topology



(b) A multi-hop flow



$\mathcal{T}^{(1)}$ $\mathcal{T}^{(2)}$ $\mathcal{T}^{(3)}$ $\mathcal{T}^{(4)}$ $\mathcal{T}^{(5)}$ $\mathcal{T}^{(6)}$ $\mathcal{T}^{(7)}$
Node 1
$\phi_1=0$, $\alpha_1=5$, $T_1=32$
Node 2
$\phi_2=1$, $\alpha_2=6$, $T_2=16$
Node 3
$\phi_3=5$, $\alpha_3=3$, $T_3=8$
Node 4
$\phi_4=3$, $\alpha_4=1$, $T_4=4$
0 1 5 7 8 16 17 22 24 25 31 32
Timeslot

(c) States ON/OFF of nodes (ON slots are gray)

Figure 5.8. An illustrative duty-cycled network with different duty-cycles and active periods

5.3 MAXIMUM OCCUPANCY OF BUFFERS

A direct and fortunate side-effect of producing a deterministic schedule is that the buffer occupancy of all intermediate nodes can be determined in advance. First consider a simple situation without NC, in which each flow's packets are stored at a node in a separate buffer. The following algorithm calculates the maximum length of a buffer that stores packets for a given flow.

```
Input:  Schedule array ScheduleArray [0..$T_{schedule}$ – 1][0..M – 1];
        Flow $f \in \mathcal{F}$; Upstream link $l_{up} \in \mathcal{L}$; Downstream link $l_{dn} \in \mathcal{L}$;
Output: Maximum buffer length $max_f(l_{up}, l_{dn})$ for flow $f$ with links $l_{up}$ and $l_{dn}$;
Begin
01.  Set $\#_f(l_{up}, l_{dn})$ and $max_f(l_{up}, l_{dn})$ to zeroes;
02.  For ($i$ from 0 to $T_{schedule}$ – 1, $j$ from 0 to M – 1) Do
03.      If ((ScheduleArray [$i$][$j$] = ACTIVE) and ($j$ = $(f, l_{up})$)) Then
04.          $\#_f(l_{up}, l_{dn}) = \#_f(l_{up}, l_{dn}) + 1$;
05.      End If
06.      If ((ScheduleArray [$i$][$j$] = ACTIVE) and ($j$ = $(f, l_{dn})$)) Then
07.          If ($\#_f(l_{up}, l_{dn}) > max_f(l_{up}, l_{dn})$) Then
08.              $max_f(l_{up}, l_{dn}) = \#_f(l_{up}, l_{dn})$;
09.          End If
10.          $\#_f(l_{up}, l_{dn}) = \#_f(l_{up}, l_{dn}) - 1$;
11.      End If
12.  End For
13.  Return $max_f(l_{up}, l_{dn})$;
End
```

Figure 5.9. Pseudo-code of algorithm **MaxBufLength ()** *without* NC

The input to **MaxBufLength ()** in Figure 5.9, is a schedule template with length $T_{schedule}$ and M sub-flows is stored in array *ScheduleArray* [0..$T_{schedule}$ – 1][0..M – 1]. We also assume that upstream link l_{up} and downstream link l_{dn} of a given flow f at an intermediate node along the routing path are predetermined (as indeed they are). With the given assumptions, the algorithm returns the maximimum length of a buffer at the intermediate node for the flow at step 13.

To this end, we use variables $\#_f(l_{up}, l_{dn})$ and $max_f(l_{up}, l_{dn})$, which are initialized to zeroes at step 1. Variable $\#_f(l_{up}, l_{dn})$ is increased (or decreased) by one whenever upstream link l_{up} (or downstream link l_{dn}) carrying flow f is scheduled, i.e., ACTIVE in step 3 (or step 6), during traversing the schedule template (steps 2 to 12). Hence, variable $\#_f(l_{up}, l_{dn})$ indicates the number of packets currently in the intermediate node's buffer. Note that before the variable is decreased, its greatest value so far is stored into variable $max_f(l_{up}, l_{dn})$ in steps 7 to 9, which therefore indicates the maximum buffer length. Note that notation (f, l) is to indicate flow f goes over link l.

To extend the algorithm to allow multiple flows $f \in \mathcal{F}$ with multiple respective upstream and downstream links, l_{up} and l_{dn}, stored in the buffer, we still use the two variables, $\#_f(l_{up}, l_{dn})$ and $max_f(l_{up}, l_{dn})$, for each of the flows f. However, variable $\#_f(l_{up}, l_{dn})$ is increased (or decreased) by one whenever any upstream link l_{up} (or any downstream link l_{dn}) carrying flow f is scheduled, i.e., ACTIVE (step 3), during traversing the schedule template (steps 2 to 12).

To be able to work with NC, i.e., a single flow f on upstream link l_{up} and downstream link l_{dn} can be a non-coded or coded sub-flow, we extend the algorithm in Figure 5.9 into that in Figure 5.10. Note that array *ScheduleArray* [0..T$_{\text{schedule}}$ – 1][0..M – 1] now can contain four possible active values for a non-coded or coded sub-flow: ACTIVE, F1_ACT, F2_ACT and BT_ACT. Hence, variable $\#_f(l_{up}, l_{dn})$ is increased (or decreased) by one whenever upstream link l_{up} (or downstream link l_{dn}) carrying flow f is scheduled, i.e., ACTIVE or F1_ACT or F2_ACT or BT_ACT in step 3 (or step 6), during traversing the schedule template (steps 2 to 12).

Input: Schedule array *ScheduleArray* [0..T$_{\text{schedule}}$ – 1][0..M – 1];

Flow $f \in \mathcal{F}$; Upstream link $l_{up} \in \mathcal{L}$; Downstream link $l_{dn} \in \mathcal{L}$;

Output: Maximum buffer length $max_f(l_{up}, l_{dn})$ for flow f with links l_{up} and l_{dn};

```
Begin
01. Set $\#_f(l_{up}, l_{dn})$ and $max_f(l_{up}, l_{dn})$ to zeroes;
02. For ($i$ from 0 to $T_{schedule} - 1$, $j$ from 0 to M – 1) Do
03.     If ((($ScheduleArray$ [$i$][$j$] = ACTIVE) or
              ($ScheduleArray$ [$i$][$j$] = F1_ACT) or
              ($ScheduleArray$ [$i$][$j$] = F2_ACT) or
              ($ScheduleArray$ [$i$][$j$] = BT_ACT)) and ($j = (f, l_{up})$)) Then
04.         $\#_f(l_{up}, l_{dn}) = \#_f(l_{up}, l_{dn}) + 1;$
05.     End If
06.     If ((($ScheduleArray$ [$i$][$j$] = ACTIVE) or
              ($ScheduleArray$ [$i$][$j$] = F1_ACT) or
              ($ScheduleArray$ [$i$][$j$] = F2_ACT) or
              ($ScheduleArray$ [$i$][$j$] = BT_ACT)) and ($j = (f, l_{dn})$)) Then
07.         If ($\#_f(l_{up}, l_{dn}) > max_f(l_{up}, l_{dn})$) Then
08.             $max_f(l_{up}, l_{dn}) = \#_f(l_{up}, l_{dn});$
09.         End If
10.         $\#_f(l_{up}, l_{dn}) = \#_f(l_{up}, l_{dn}) - 1;$
11.     End If
12. End For
13. Return $max_f(l_{up}, l_{dn})$;
End
```

Figure 5.10. Pseudo-code of algorithm **MaxBufLength ()** *with* NC

To extend the algorithm to allow multiple flows $f \in \mathcal{F}$ with multiple respective upstream and downstream links, l_{up} and l_{dn}, stored in the buffer, we still use the two variables, $\#_f(l_{up}, l_{dn})$ and $max_f(l_{up}, l_{dn})$ for each flow f. However, variable $\#_f(l_{up}, l_{dn})$ is increased (or decreased) by one whenever any of the upstream links l_{up} (or any of the downstream links l_{dn}) carrying flow f is scheduled, i.e., ACTIVE or F1_ACT or F2_ACT or BT_ACT in step 3 (or step 6), during traversing the schedule template (steps 2 to 12).

5.4 EXECUTION OVERHEAD OF MWIS

In our work, we need to find a maximum weighted independent set (MWIS) in the contention graph of ON sub-flows as vertices at each timeslot. This is aimed to find as many ON sub-flows with higher priority as possible for simultaneous transmission at the current timeslot. Since the MWIS problem is NP-hard, we use the approximation greedy algorithm by Sakai et al. [21] which has a time complexity of $O(n^2)$, where n is the number of ON sub-flows. To find out the computational overhead of our scheduling algorithm including the state matching and excision process, we should compare the complexity of the approximation MWIS algorithm used in our work with exact MWIS algorithms because the MWIS algorithm is most frequently used in work, i.e., in each timeslot.

There have been exact algorithms with exponential time complexity to solve the MWIS problem in the literature. One of the most inefficient algorithms is a brute force algorithm which solves the problem in time $O(n^2 2^n)$ by checking every vertex subset of the graph whether it is an independent set. Much more efficient than this naïve algorithm is the work by Robson [54] which solves the problem in time (1.2108^n), the work by Bourgeois [55], and the work by Fomin [56] that solve MWIS in $O(1.2127^n)$ and $O(1.2209^n)$, respectively.

From the complexity of the above exact algorithms, we can compare the time efficiency between the approximation algorithm from [21] and those from [54, 55, 56], which depends on the number of ON sub-flows, i.e., n in the above-mentioned algorithms. For example, suppose n is 50 ON sub-flows and it takes around 15 minutes for us to run a simulation using the greedy approximation algorithm as the MWIS step. The exact brute force algorithm would take $2^{50} * 15$ minutes, which takes for ever to run the simulation. Compared this also to the work of [54, 55, 56] which would take around 2 hours. However, if n is 100 ON sub-flows and a simulation takes 15 minutes to run, then the work of [54, 55, 56] would take more than 2 years to run a single simulation.

5.5 CODING COMBINATIONS OF POSSIBLE FLOWS

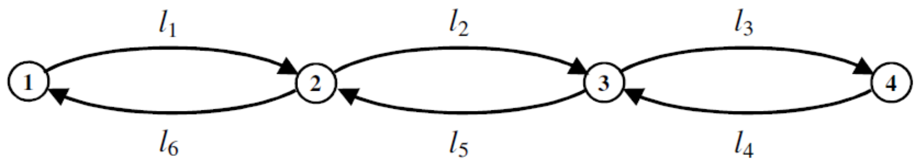
From earlier observations of Example 5.1., it is obviously seen that we can benefit from coding two flows, f_1 and f_2 with respect to throughput when the rates of the two flows are not equal, e.g., when a fraction of the larger (let's say) f_2 is coded with the entire flow f_1. Hence, another question is raised, when there is more than one possibility to pair flows for coding at an intermediate node, which pairing can give the best benefit in terms of throughput improvement?

Example 5.2. To answer the question, let us add another flow, i.e., f_3 shown in Figure 5.11.b, which can be coded with one of the two flows, i.e., f_1. Hence, there are two possible combinations for coding in the example, between flows f_2 and f_1 as shown in Figure 5.12.b.1 or between flows f_2 and f_1 as shown in Figure 5.12.b.2. To decide which combination can give better benefit of coding, we compute the rates of flows in cases without NC and with NC by each of the combinations.

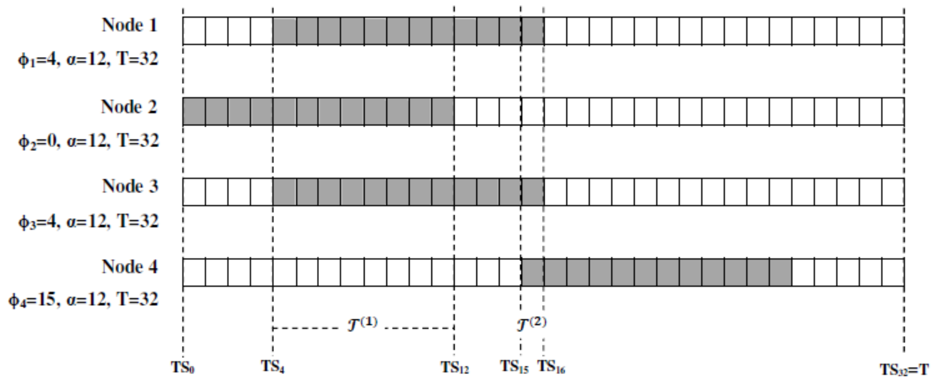
From Figure 5.11, i.e., without NC, we can calculate the following numerical flow rates in all stages.

$$r_1 = 0.03125,\ r_2 = 0.046875,\ r_3 = 0.046875;$$
$$r_{f_1}^{(1)}(l_1) = 0.03125,\ r_{f_1}^{(1)}(l_2) = 0.03125,\ r_{f_1}^{(1)}(l_3) = 0;$$
$$r_{f_2}^{(1)}(l_5) = 0.046875,\ r_{f_2}^{(1)}(l_6) = 0.046875;$$
$$r_{f_3}^{(1)}(l_1) = 0.046875,\ r_{f_3}^{(1)}(l_2) = 0.046875;$$
$$r_{f_1}^{(2)}(l_1) = 0,\ r_{f_1}^{(2)}(l_2) = 0,\ r_{f_1}^{(2)}(l_3) = 0.03125;$$
$$r_{f_2}^{(2)}(l_5) = 0,\ r_{f_2}^{(2)}(l_6) = 0;$$
$$r_{f_3}^{(2)}(l_1) = 0,\ r_{f_3}^{(2)}(l_2) = 0; \qquad (5.4)$$

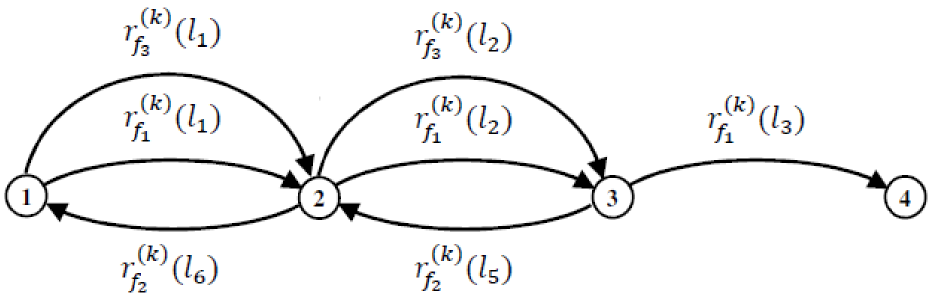
In Figure 5.12.b.1, i.e., with the coding combination of flows f_2 and f_1, we have the following flow rates.



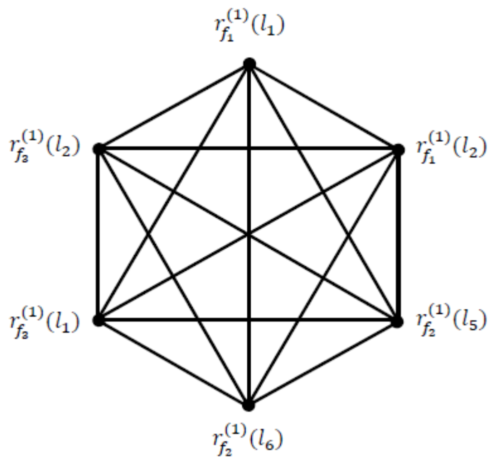
(a.1) Link topology

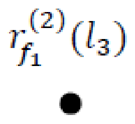


(a.2) States ON/OFF of nodes (ON slots are gray)



(b) Multi-hop flows

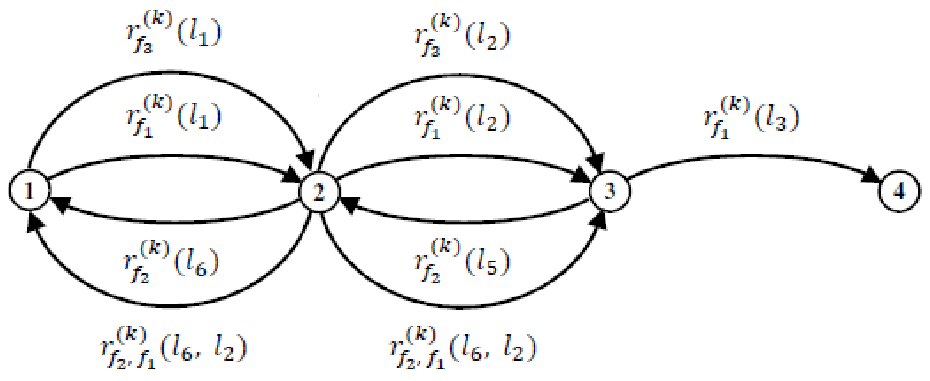




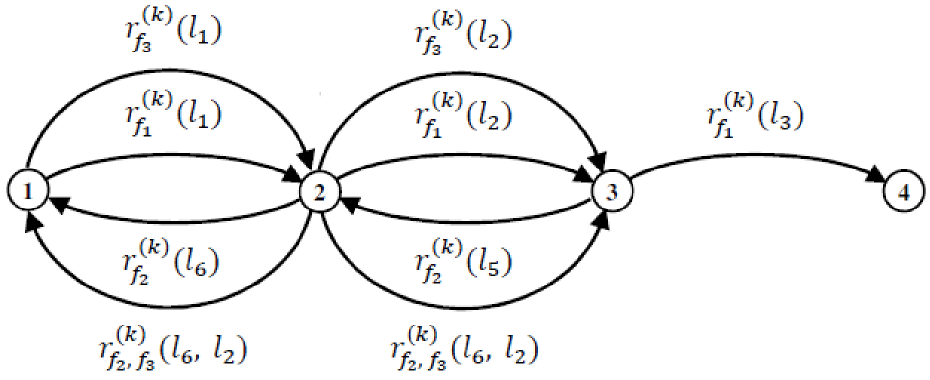
(c.1) CG in $\mathcal{T}^{(1)}$

(c.2) CG in $\mathcal{T}^{(2)}$

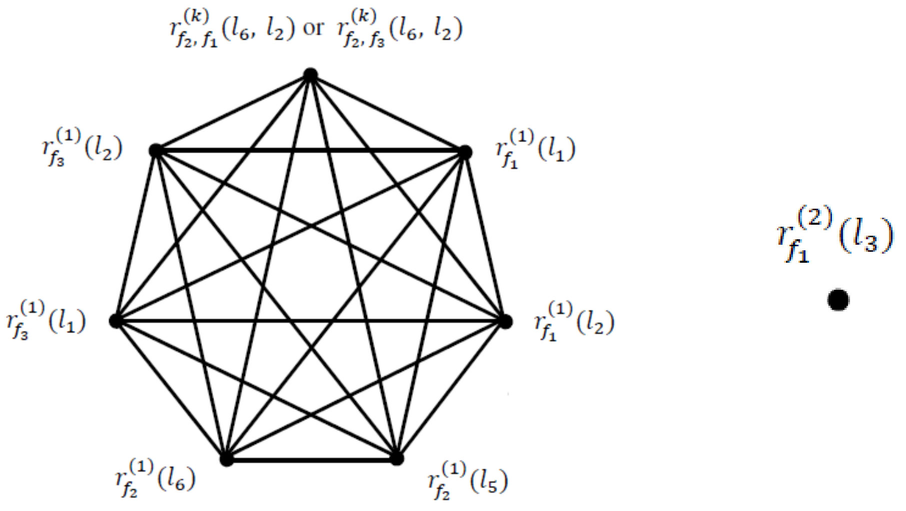
Figure 5.11. An illustrative example of the combination of possible flows for coding (*before* NC)



(b.1) Multi-hop flows with the coding of f_2 and f_1 (smaller rate)



(b.2) Multi-hop flows with the coding of f_2 and f_3 (greater rate)



(c.1) CG in $\mathcal{T}^{(1)}$ (c.2) CG in $\mathcal{T}^{(2)}$

Figure 5.12. An illustrative example of the combination of possible flows for coding (*after* NC)

$r_1 = 0.03125,\ r_2 = 0.0546875,\ r_3 = 0.0546875;$

$r_{f_1}^{(1)}(l_1) = 0.03125,\ r_{f_1}^{(1)}(l_2) = 0,\ r_{f_2,f_1}^{(1)}(l_6, l_2) = 0.03125,\ r_{f_1}^{(1)}(l_3) = 0;$

$r_{f_2}^{(1)}(l_5) = 0.0546875,\ r_{f_2}^{(1)}(l_6) = 0.0234375;$

$r_{f_3}^{(1)}(l_1) = 0.0546875,\ r_{f_3}^{(1)}(l_2) = 0.0546875;$

$r_{f_1}^{(2)}(l_1) = 0,\ r_{f_1}^{(2)}(l_2) = 0,\ r_{f_2,f_1}^{(2)}(l_6, l_2) = 0,\ r_{f_1}^{(2)}(l_3) = 0.03125;$

$r_{f_2}^{(2)}(l_5) = 0,\ r_{f_2}^{(2)}(l_6) = 0;\ r_{f_3}^{(2)}(l_1) = 0,\ r_{f_3}^{(2)}(l_2) = 0;$ (5.5)

In Figure 5.12.b.2, i.e., with the coding combination of flows f_2 and f_3, we have the following flow rates.

$r_1 = 0.03125,\ r_2 = 0.0625,\ r_3 = 0.0625;$

$r_{f_1}^{(1)}(l_1) = 0.03125,\ r_{f_1}^{(1)}(l_2) = 0.03125,\ r_{f_1}^{(1)}(l_3) = 0;$

$r_{f_2}^{(1)}(l_5) = 0.0625,\ r_{f_2}^{(1)}(l_6) = 0,\ r_{f_2,f_3}^{(1)}(l_6, l_2) = 0.0625;$

$r_{f_3}^{(1)}(l_1) = 0.0625,\ r_{f_3}^{(1)}(l_2) = 0;$

$r_{f_1}^{(2)}(l_1) = 0,\ r_{f_1}^{(2)}(l_2) = 0,\ r_{f_1}^{(2)}(l_3) = 0.03125;$

$r_{f_2}^{(2)}(l_5) = 0,\ r_{f_2}^{(2)}(l_6) = 0,\ r_{f_2,f_3}^{(2)}(l_6, l_2) = 0;\ r_{f_3}^{(2)}(l_1) = 0,\ r_{f_3}^{(2)}(l_2) = 0;$ (5.6)

From (5.4), (5.5) and (5.6), we have the following observations: (1) the greater coding rate, e.g., $r_{f_2,f_1}^{(1)}(l_6, l_2)$ or $r_{f_2,f_3}^{(1)}(l_6, l_2)$, the more the coding benefit; (2) the coding rate is constrained by the smaller of the two rates of the paired flows; (3) when the smaller rate cannot increase due to a flow constraint, e.g., the rate of f_1 from (5.5), the coding still helps the coding pair of flows to consume less bandwidth than that without NC; (4) when the smaller rate can increase further, the coding can also help to improve the rate, e.g., the rate of f_2 and f_3 from (5.6).

From the above observations, we propose the following heuristic approach when there is more than one possibility of coding two flows at an intermediate node we should select the pairs of flows, in which the smaller rate of the pair is greater than that of the other pairs, i.e., the other possibilities. This approach is essentially aimed to take full advantage of coding.

Input: Sub-flow array A_SF [0..M-1] of all the M sub-flows;

Nodes $n \in \mathcal{N}$; Flows $f \in \mathcal{F}$; Links $l \in \mathcal{L}$;

Output: Pairing matrix P_SF [0..M-1][0..M-1] of all the pairs of M sub-flows;

```
Begin
01.  For ($i$ from 0 to M – 1, $j$ from 0 to M – 1)  Do
02.      $C\_SF\ [i][j] = \text{NO}; P\_SF\ [i][j] = \text{NO}$;
03.  End For
04.  For Each node $n \in \mathcal{N}$  Do
05.      $maxRate = -1; max_i = -1; max_j = -1$;
06.      For ($i$ from 0 to M – 1, $j$ from 0 to M – 1)  Do
07.          If  (($C\_SF\ [i][j] = \text{NO}$) and
                 ($pair\left((f_1, l_1), (f_2, l_2)\right)$ = YES))  Then
08.              $LP\left((f_1, l_1), (f_2, l_2), r_{f_1}, r_{f_2}\right)$;
09.              $C\_SF\ [i][j]$ = YES;
10.              If  ($min\left(r_{f_1}, r_{f_2}\right) > maxRate$)  Then
11.                  $maxRate = min\left(r_{f_1}, r_{f_2}\right)$;
12.                  $max_i = i; max_j = j$;
13.              End If
14.          End If
15.      End For
16.      If  (($maxRate > 0$) and
             (($max_i \geq 0$) and ($max_i \leq$ M)) and
             (($max_j \geq 0$) and ($max_j \leq$ M)))  Then
17.          $P\_SF\ [max_i][max_j]$ = YES;
18.      End If
19.  End For
20.  Return $P\_SF$ [0..M-1][0..M-1];
End
```

Figure 5.13. Pseudo-code of algorithm **PairwiseCoding ()**

The pseudo-code of algorithm **PairwiseCoding ()** in Figure 5.13 shows more details of how to implement the proposed heuristic approach. We assume that the shortest-single-path routing for all the flows is predetermined, which means all the sub-flows are also determined. Hence, we suppose all the M sub-flows are stored in array A_SF [0..M-1]. With these assumptions, the algorithm returns pairing matrix P_SF [0..M-1][0..M-1] (step 20), whose entry P_SF [i][j] indicates YES when sub-flow i can be paired with sub-flow j for the best coding at the common sender of the two sub-flows.

To this end, we use coding matrix C_SF [0..M-1][0..M-1], whose entry C_SF [i][j] indicates YES (at step 9) when sub-flow i has been attemped to be paired with sub-flow j for coding at the common sender of the two sub-flows and hence the significant running time of steps 7 to 14 can be saved. Also, we use variables $maxRate$, max_i and max_j, which are initialized to negative values (at step 5) at the beginning of loop For (steps 4 to 19). Variable $maxRate$ is used to indicate which pair of sub-flows has the smaller rate greater than that of the other pairs at node n (steps 10 to 13). Meanwhile, variables max_i and max_j are two indexes of entry P_SF $[max_i][max_j]$, which stores the best selected pair (at step 17).

The algorithm works as follows. First, matrixes C_SF [0..M-1][0..M-1] and P_SF [0..M-1][0..M-1] are initialized to NO, i.e., no pairs of sub-flows have been selected for coding. Then, for each node n (steps 4 to 19), it checks if any pair of sub-flows, which has not been selected for coding and sastified with the coding condition, i.e., the two sub-flows have a common sender and the duty-cycle of the sender overlaps those of the two receivers of the two sub-flows, (f_1, l_1) and (f_2, l_2). This condition is checked by function $pair\left((f_1, l_1), (f_2, l_2)\right)$ at step 7, which returns YES if it is satisfied. Hence, two numerical rates r_{f_1} and r_{f_2} of two respective flows f_1 and f_2 are returned by calling function $LP\left((f_1, l_1), (f_2, l_2), r_{f_1}, r_{f_2}\right)$, which is actually done by lex-max formulations associated with the MP program from Chapter 3 and indicated by turning YES the entry $C\_SF\ [i][j]$. Finally, steps 10 to 13 are to check and store the best coding pair of sub-flows at node n, which are then updated to entry $P\_SF\ [max_i][max_j]$ by steps 16 to 18. Note that function $min\left(r_{f_1}, r_{f_2}\right)$ is to find the minimum value between two rates r_{f_1} and r_{f_2}.

The algorithm is a heuristic one because it does not consider all possibilities of pairing sub-flows for coding and hence the algorithm can save more time while being able to provide an acceptable solution. One of the most inefficient algorithms in time is a brute force algorithm that solves the problem in time $O(m^2 2^m LP(r,p))$ by checking every subset of m sub-flows whether it can result in the maximum total throughput, where $LP(r,p)$ is the complexity of linear programming determined by the number of flows r and p linear equalities in the LP formulations. Meanwhile, the heuristic algorihm in Figure 5.13 solves the problem in time $O(m^2 n LP(r,p))$, where n is the number of nodes in the network, which is much smaller than that of the brute force algorithm when the number of sub-flows becomes greater.

CHAPTER 6
CONCLUSIONS

Since transceiver duty-cycling (DC) is very popular technique to conserve energy in a wireless sensor network (WSN), we started our research with the intention of combining DC with Network Coding (NC), a technique that has also been seen as a means to achieve better throughput and energy efficiency. We have proposed and used a model for the DC characteristics of each node n, given in the form of a tuple $\langle \phi_n, \alpha_n, T_n \rangle$. This model originates from an abstract view of WSNs where the periodic DC is dictated by, either, application and hence traffic generation and coverage demands, or because of energy harvesting operation that varies from location to location.

An early attempt to combine DC and NC (Section 2.6) resulted in a MAC protocol and an elaborate NC mechanism, which demonstrated that the combination of DC and NC could indeed bear fruit. However, this early work was limited in applicability because of its assumptions pertaining to network topology and the particular DC timing. Reflecting on those early results, it was decided that developing schemes that would work in arbitrary topologies would be essential to broadening the applicability of NC+DC. However, with arbitrary topologies becoming our standard assumption for the subsequent parts of the thesis, the ability for spatial reuse of the medium would have to be incorporated. It was no longer sufficient to think in terms of extending the MAC protocol of the early work, but in terms of what would be a benchmark performance, by producing a, short, collision-free schedule. Hence, the quest for arbitrary topologies necessitated also the production of STDMA schedules.

The next step was to, first, express traffic flow rates as part of solving an optimization problem (in this case, with the objective to ensure fair rate allocation) and then perform the schedule construction that achieves those rates (Chapter 3). Two contributions helped in achieving the construction of the schedule. One was the characterization of the DC network as periodically repeating “stages”, and, second, the observation (and experimental confirmation) that the system as a whole would exhibit a periodic steady state. Essentially, in a simulation of the system, it becomes evident that the scheduling decisions repeat again and again in a periodic fashion (with some period) as they reflect deterministic actions on the periodically repeating state of the network. This periodic steady state was isolated, excised, and became a scheduling template, i.e., an STDMA schedule.

What Chapter 3 was lacking, i.e., the inclusion of NC, was addressed in Chapter 4. The techniques of Chapter 3 were extended to account for NC, and proved to be potent as they were without NC. However, dealing with NC required two additional techniques that to be developed. One was to create systematic NC opportunities such that they could be determined a-priori, i.e., during schedule construction. This was accomplished by combining (in the NC sense) pairs of flows. In other words, pairwise XOR NC was used. The scheme was applied at intermediate nodes, hence reducing the overall network traffic. However, a second technique had to be developed as a consequence of needing systematic opportunities for NC, a *delay coding* technique that could ensure the supply of enough packets to nodes such the NC combinations could take place. Essentially, by paying a bit of upfront delay cost, the flows are certain to have enough traffic to perform NC.

Our objective appears to have been achieved because with our proposed techniques we have provided simulation results in Chapter 4 that show, e.g., energy consumption savings of 23%, and throughput improvement of up to 50% in DC+NC configurations.

Finally, in Chapter 5, we demonstrated how the techniques of Chapter 3 & 4 can be applied to more general environments, e.g., with heterogeneous DC, or with less traffic than what the schedule was anticipating, or with different combinations of flows being paired for NC.

6.1 THESIS STATEMENT

To the extent that a thesis has been put forward, it is that in a network with periodic topology changes (as is the case with DC), fed by greedy traffic arrivals, and with deterministic behavior, the system's pattern of transmissions in the steady state ought to be a periodic one and that periodic behavior could be used to construct transmission schedules, i.e., a simulation is a valid way to produce the schedule.

6.2 OPEN ISSUES

There are many open issues towards making NC+DC schemes a viable set of techniques.

- In our future work, we will attempt to elaborate the inter-dependency between queues and transmissions and how this is manifested in the schedule template excision. As is now, it is based purely on conjecture that the periodic steady state will emerge and will morph the scheduling decisions into also being periodically repeating. A shortcoming of the excision process is that it ignores the "true" state of the system, i.e., the state of the queues whose influence is only accounted for indirectly by the impact they have on the scheduling decisions from slot to slot. The reader can question whether the state of queues ought to have been a component of the state comparison/matching. We have, so far, avoided matching the queue state in addition to the transmissions because it is unclear whether matching the exact number of packets in the queue is necessary (or just that there are some) and for the added complexity that this would bring about (the buffers can be as many as the flows times the number of links in the network).

- We have separated the routing from the scheduling problem, assuming that routing is given to us, or at least it is pre-calculated. The question of joint routing and scheduling for a DC network remains (and even more so for DC+NC).

- It remains an open issue of how to construct a MAC scheme that resembles the performance we can extract with the centralized STDMA schedule construction. In particular, a MAC protocol with randomized behavior would undermine seriously the determinism we assumed throughout the schedule construction. NC would then be also extended from pairwise-coding to opportunistic coding like COPE [22], becoming able to combine multiple transmissions.

- Similar to the concerns arising from the randomness of a MAC protocol would also arise if the active periods are also random. In particular, if the active periods are assumed to be a function of harvesting energy, their length could vary from one period to the next. Our framework currently has no provision for random active periods.

- A final note is that we have consistently opted for sacrificing delay in the interest of obtaining higher throughput (e.g., in the delay coding technique, in the slot deactivation extension, etc.). However, this approach is not acceptable to all applications. A drastically different approach would be needed if we wish to e.g., optimize the delay, as it would have to inescapably be mapped to a priority service scheme of (sub-)flows that honors delay requirements of the flows and not weight constructs like those we use for the sake of MWIS.

REFERENCES

[1] Somayeh Kianpisheh and Nasrolah Moghadam Charkari, "Dynamic Power Management for Sensor Node in WSN Using Average Reward MDP", *Wireless Algorithms, Systems, and Applications, 4th International Conference*, WASA 2009, Boston, MA, USA, August 16-18, 2009. Proceedings 2009

[2] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. "A Macroscope in the Redwoods". In *Proc. of ACM SENSYS*, 2005

[3] Y. Gu and T. He. "Data Forwarding in Extremely Low Duty-Cycle Sensor Networks with Unreliable Communication Links". In *Proc. Of ACM SENSYS*, 2007

[4] P. Ciciriello, L. Mottola, and G. P. Picco. "Efficient routing from multiple sources to multiple sinks in wireless sensor networks". In *Proc. of the 4th EWSN*, 2007

[5] Aman Kansal , Jason Hsu , Sadaf Zahedi , Mani B. Srivastava, "Power management in energy harvesting sensor networks," *ACM Transactions on Embedded Computing Systems (TECS)*, v.6 n.4, p.32-es, September 2007

[6] Wind Data 2001. Noaa recorded average wind speed data through 2001. http://www.berner.com/new/energy-windspeed.htm

[7] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro, "Time-Varying Graphs and Dynamic Networks," Arxiv preprint arXiv:1012.0009v2, 2011

[8] I. Chabini, “Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time,” *Transportation Research Records*, vol. 1645, pp. 170–175, 1998

[9] B. Ding, J. X. Yu, and L. Qin, “Finding time-dependent shortest paths over large graphs,” in *Proceedings of Extending database technology (EDBT’08)*, 2008, pp. 205–216

[10] H. Chon, D. Agrawa, and A. Abbadi., “Fates: Finding a time dependent shortest path,” *Mobile Data Management*, vol. 2574, pp. 165–180, 2003

[11] Shor, J.; Robertazzi, T.G., “Traffic Sensitive Algorithms and Performance Measures for the Generation of Self-Organizing Radio Network Schedules,” *Communications, IEEE Transactions on* , vol.41, no.1, pp.16,21, Jan 1993

[12] O. Somarriba, “Multihop Packet Radio Systems in Rough Terrain”, Tech.lic. thesis, *Radio Communication Systems*, Department of S3, Royal Institute of Technology, SE-100 44 Stockholm, Sweden, Oct.1995

[13] Gronkvist, J., “Traffic Controlled Spatial Reuse TDMA in Multi-hop Radio Networks,” *Personal, Indoor and Mobile Radio Communications, 1998. The Ninth IEEE International Symposium on* , vol.3, no., pp.1203,1207 vol.3, 8-11 Sep 1998

[14] Jimmi Gronkvist, “Novel Assignment Strategies for Spatial Reuse TDMA in Wireless Ad Hoc Networks”, *Wireless Networks*, 12:255–265, 2006

[15] Van Ho, Ioanis Nikolaidis, “A Schedule Template Construction Technique for Duty Cycled Sensor Networks”, ADHOC-NOW 2015: 48-61

[16] F. Zuyuan and B. Bensaou. "Fair bandwidth sharing algorithms based on game theory frameworks for wireless ad-hoc networks," In *INFOCOM 2004,* pp. 1284-1295

[17] A. Bar-Noy, V. Dreizin, and B. Patt-Shamir. "Efficient algorithms for periodic scheduling," *Computer Networks*, vol. 45, no. 2, pp. 155-173, 2004

[18] A. Casteigts *et al*. "Time-varying graphs and dynamic networks," In *ADHOC-NOW 2011*, pp. 346-359

[19] H. Willie, "Periodic steady state of loss systems with periodic inputs", *Advances in Applied Probability*, vol. 30, no. 1, pp. 152-166

[20] K. Jain, J. Padhye, V. N. Padmanabhan, and L. Qiu, "Impact of interference on multi-hop wireless network performance," in MobiCom 2003

[21] S. Sakai, M. Togasaki, and K. Yamazaki. "A note on greedy algorithm for the maximum weighted independent set problem," *Discrete Applied Mathematics*, vol. 126, no. 2-3, pp. 313-322, 2003

[22] Katti, S.; Rahul, H.; Wenjun Hu; Katabi, D.; Medard, M.; Crowcroft, J., "XORs in the Air: Practical Wireless Network Coding," *Networking, IEEE/ACM Transactions on* , vol.16, no.3, pp.497,510, June 2008

[23] S. Li, R. Yeung, and N. Cai, "Linear Network Coding", in IEEE Transactions on Information Theory, Vol 49, No. 2, pp. 371–381, 2003

[24] Carrano, R.C.; Passos, D.; Magalhaes, L.C.S.; Albuquerque, C.V.N., "Survey and Taxonomy of Duty Cycling Mechanisms in Wireless Sensor

Networks," *Communications Surveys & Tutorials, IEEE* , vol.16, no.1, pp.181,194, First Quarter 2014

[25] G. Lu, B. Krishnamachari, and C. S. Raghavendra, "An Adaptive Energy-Efficient and Low-Latency MAC for Data Gathering in Wireless Sensor Networks," *Proc. 18th Int'l. Parallel and Distrib. Processing Symp.*, Apr. 2004, p. 224

[26] Baochun Li, "End-to-End Fair Bandwidth Allocation in Multi-Hop Wireless Ad Hoc Networks," *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on* , vol., no., pp.471,480, 10-10 June 2005

[27] H. Luo, S. Lu, and V. Bharghavan, "A New Model for Packet Scheduling in Multihop Wireless Networks," in *Proceedings of ACM MobiCom*, 2000, pp. 76–86

[28] Aman Kansal , Jason Hsu , Sadaf Zahedi , Mani B. Srivastava, "Power Management in Energy Harvesting Sensor Networks," *ACM Transactions on Embedded Computing Systems (TECS)*, v.6 n.4, p.32-es, September 2007

[29] R. Nelson, L. Kleinrock, "Spatial-TDMA: A Collision-Free Multi-hop Channel Access Control", *IEEE Transactions On Communications* 33 (1985) 934-944

[30] L. Kleinrock and J. Silvester, "Spatial reuse in multihop packet radio networks," *Proceedings of the IEEE* , vol.75, no.1, pp.156,167, Jan. 1987

[31] Xiao Long Huang, Brahim Bensaou, "On max-min fairness and scheduling in wireless ad-hoc networks: analytical framework and implementation",

Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking & Computing, October 04-05, 2001

[32] Zuyuan Fang; Bensaou, B., "Fair bandwidth sharing algorithms based on game theory frameworks for wireless ad-hoc networks," *INFOCOM 2004. Twenty-third Annual-Joint Conference of the IEEE Computer and Communications Societies* , vol.2, no., pp.1284,1295 vol.2, 7-11 March 2004

[33] L. Tassiulas , S. Sarkar, "Maxmin fair scheduling in wireless ad hoc networks", *IEEE Journal on Selected Areas in Communications*, v.23 n.1, p.163-173, September 2006

[34] Heyman, D.P. and Whitt, W. (1984), "The asymptotic behavior of queues with time-varying arrival rates", *J. Appl. Prob. 21*, 143-156

[35] Bambos, N. and Walrand, J. (1989), "On queues with periodic inputs", *J. Appl. Prob. 26*, 381-389

[36] J.M. Harrison and A.J. Lemoine, "Limit theorems for periodic queues", J. Appl. Prob. 14, 566-576, 1977

[37] A.J. Lemoine, "Waiting time and workload in queues with periodic Poisson input", J. Appl. Prob. 26, 390-397, 1989

[38] T. Rolski, "Approximation of periodic queues", Adv. Appl. Prob. 19, 691-707, 1987

[39] C. H. Papadimitriou, K. Steiglitz (1998), "6.1 The Max-Flow, Min-Cut Theorem", *Combinatorial Optimization: Algorithms and Complexity*. Dover. pp. 120–128

[40] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung. "Network Information Flow", *IEEE Transactions on Information Theory*, 2000

[41] Firooz, M.H.; Zhiyong Chen; Roy, S.; Hui Liu, "Wireless Network Coding via Modified 802.11 MAC/PHY: Design and Implementation on SDR," *Selected Areas in Communications, IEEE Journal on* , vol.31, no.8, pp.1618,1628, August 2013

[42] Van Ho; Nikolaidis, I.; , "Trade-offs of Combining Network Coding and Duty Cycling in WSNs," *Communication Networks and Services Research Conference (CNSR), 2011 Ninth Annual* , vol., no., pp.231-238, 2-5 May 2011

[43] R. Chandanala, W. Zhang, R. Stoleru and M. Won, "On combining network coding with duty-cycling in flood-based wireless sensor networks," Ad Hoc Networks, vol. 11, no. 2, pp. 490-507, 2013

[44] Rout, R.R.; Ghosh, S.K., "Enhancement of Lifetime using Duty Cycle and Network Coding in Wireless Sensor Networks," *Wireless Communications, IEEE Transactions on* , vol.12, no.2, pp.656-667, February 2013

[45] Gburzynski, P., "Olsonet Communications: SIDE/SMURPH: a Modeling Environment for Reactive Telecommunication Systems," Version 3.1 manual, 2008

[46] Shabdanov, S.; Rosenberg, C.; Mitran, P., "Joint routing, scheduling, and network coding for wireless multihop networks," *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2011 International Symposium on* , vol., no., pp.33,40, 9-13 May 2011

[47] D. Bertsekas and R. Gallager. *Data Networks (2e)*, Prentice Hall, 1992

[48] Radunovic, B.; Le Boudec, J.-Y., "A Unified Framework for Max-Min and Min-Max Fairness With Applications," *Networking, IEEE/ACM Transactions on* , vol.15, no.5, pp.1073,1083, Oct. 2007

[49] D. Nace and M. Pioro. "Max-min fairness and its applications to routing and load-balancing in communication networks: A tutorial", *IEEE Comm. Surv. Tutorials*, vol. 10, no. 4, pp.5 -17, 2008

[50] M. Pioro and D. Medhi. "Routing, Flow and Capacity Design in Communication and Computer Networks", Morgan Kaufmann Publishers (2004)

[51] J. Kleinberg, Y. Rabani, and E. Tardos. "Fairness in routing and load balancing". In *Proc. 35th Annual Symposium on Foundations of Computer Science*, 1999

[52] W. Ogryczak, M. Pioro and A. Tomaszewski. "Telecommunications Network Design and Max-Min Optimization Problem". *J. Telecommunications and Information Technology* 3/2005, pp. 1-14

[53] M. Eickhoff, D. McNickle, and K. Pawlikowski. "Detecting the duration of initial transient in steady state simulation of arbitrary performance measures," In *ValueTools 2007,* article #42

[54] Robson, J. M. , "Algorithms for maximum independent sets", *Journal of Algorithms* 7 (3): 425–440, 1986

[55] Bourgeois, Nicolas; Escoffier, Bruno; Paschos, Vangelis Th.; van Rooij, Johan M. M., "A bottom-up method and fast algorithms for MAX INDEPENDENT SET", *Algorithm theory—SWAT 2010*, Lecture Notes in Computer Science 6139, Berlin: Springer, pp. 62–73, 2010

[56] Fomin, Fedor V.; Grandoni, Fabrizio; Kratsch, Dieter, “A measure & conquer approach for the analysis of exact algorithms”, *Journal of ACM* 56 (5): 1–32, article no. 25, 2009

[57] Akyildiz, I. F.; Kasimoglu, I. H. “Wireless sensor and actor networks: research challenges”, Ad Hoc Netw. 2004, 2 (4), 351-367

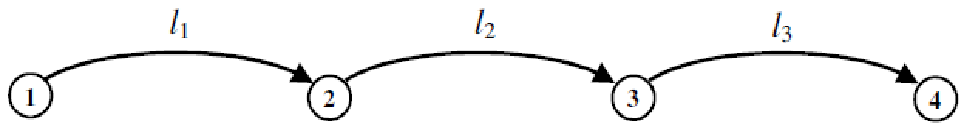
[58] Rezgui, A.; Eltoweissy, M. “Service-oriented sensor-actuator networks: Promises, challenges, and the road ahead”, Computer Communications 2007, 30, 2627-2648

[59] Melodia, T.; Pompili, D.; Gungor, V. C.; Akyildiz, I. F. “Communication and Coordination in Wireless Sensor and Actor Networks”. IEEE Transactions on Mobile Computing 2007, 6(10), 1116-1129

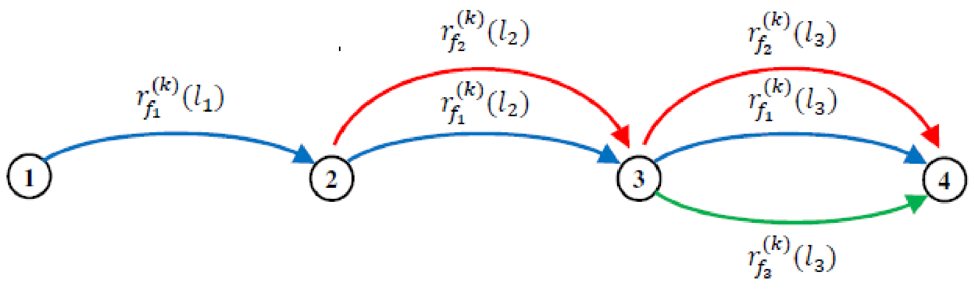
[60] Arnaud Casteigts, Paola Flocchini, Emmanuel Godard, Nicola Santoro, and Masafumi Yamashita. “On the expressivity of time-varying graphs”. In 19th International Symposium on Fundamentals of Computation Theory (FCT), Liverpool, United Kingdom, 2013

APPENDIX A
ANOTHER EXAMPLE OF COMPARISON WITH THE GRONKVIST'S ALGORITHM

For the sake of comparison of the Gronkvist's scheduling algorithm [13] against the one we present in Chapter 3, we present the simple topology of Example A.1 in Figure A.1 to clarify the steps of constructing a schedule.



(a) Link topology



(b) Multi-hop flows

Figure A.1. Topology and flows used in the comparison.

Example A.1. There are 3 directional links, from l_1 to l_3 in Figure A.1.a. Also, there are three flows with flow paths in Figure A.1.b, flow f_1 in blue from source 1 to sink 4, flow f_2 in red from source 2 to sink 4, and flow f_3 in green from source 3 to sink 4. Noting that the algorithm in [13] does not have sub-flows like in our work and only requires the average traffic loads on links to perform its schedule construction. Therefore, we run our scheduling algorithm on the network to get all the rates $r_f^{(k)}(l)$ of flow f over link l, from which we then calculate the average traffic load, λ_{ij} on link (i, j), or λ_l on link l and provide it as input to the algorithm in [13]. To model the interference among simultaneous transmissions, we assume both algorithms use the same model of link interference as in the work by Li [26].

By running our MMF scheduling algorithm on the network, we construct the schedule for all the sub-flows in each timeslot $\boldsymbol{t}$ shown in Figure A.2. Hence, we achieve the rates of all the flows over links as follows.

$$r_{f_1}^{(k)}(l_1) = r_{f_1}^{(k)}(l_2) = r_{f_3}^{(k)}(l_3)$$

$$r_{f_2}^{(k)}(l_2) = r_{f_2}^{(k)}(l_3) =$$

$$r_{f_3}^{(k)}(l_3) = \frac{1}{6} = 0.166666 \text{ (packets/slot)} \qquad \text{(A.1)}$$

$\boldsymbol{t}$	$\boldsymbol{r_{f_1}^{(k)}(l_1)}$	$\boldsymbol{r_{f_1}^{(k)}(l_2)}$	$\boldsymbol{r_{f_1}^{(k)}(l_3)}$	$\boldsymbol{r_{f_2}^{(k)}(l_2)}$	$\boldsymbol{r_{f_2}^{(k)}(l_3)}$	$\boldsymbol{r_{f_3}^{(k)}(l_3)}$
1	●	○	○	○	○	○
2	○	●	○	○	○	○
3	○	○	●	○	○	○
4	○	○	○	●	○	○
5	○	○	○	○	●	○
6	○	○	○	○	○	●

Figure A.2. Schedule is constructed by our MMF algorithm (black circles mean sub-flows are scheduled).

From (A.1) and flow paths in Figure A.1.b, we can calculate all the following traffic loads on links.

$$\lambda_{l_1} = r; \lambda_{l_2} = 2r; \lambda_{l_3} = 3r; \qquad \text{where } r = 0.166666 \text{ (packets/slot)} \qquad \text{(A.2)}$$

Now, to construct the schedule as per [13] we have.

N = 4 (nodes); M = 3 (links) (A.3)

Hence, we have the total traffic load of three flows in the network as follows.

$$\lambda = 3r = 0.5 \text{ (packets/slot)} \tag{A.4}$$

From [13], we have relative traffic Λ_l on link l defined as follows:

$$\Lambda_l = \lambda_l / (\lambda / \mathrm{N}\,(\mathrm{N}-1))$$

Hence, we can calculate the following.

$$\Lambda_{l_1} = 4; \Lambda_{l_2} = 8;\ \Lambda_{l_3} = 12; \tag{A.5}$$

From [13] and with (A.3), (A.4) and (A.5), we have the average relative traffic in the network, which is defined as follows.

$$\overline{\Lambda} = \frac{1}{M}\sum_{\forall l} \Lambda_l = (\Lambda_{l_1} + \Lambda_{l_2} + \Lambda_{l_3})/M = 8 \tag{A.6}$$

From [13], we have link l is guaranteed the following number of slots.

$$\left\lceil \frac{\Lambda_l}{\overline{\Lambda}} \right\rceil \tag{A.7}$$

From (A.6) and (A.7), we derive the following.

$$\left\lceil \frac{\Lambda_{l_1}}{\overline{\Lambda}} \right\rceil = 1; \left\lceil \frac{\Lambda_{l_2}}{\overline{\Lambda}} \right\rceil = 1; \left\lceil \frac{\Lambda_{l_3}}{\overline{\Lambda}} \right\rceil = 2; \tag{A.8}$$

t	$\Lambda_{l_1}\tau_{l_1}$	$\Lambda_{l_2}\tau_{l_2}$	$\Lambda_{l_3}\tau_{l_3}$	**List A**	**Schedule**
0	0	0	0	l_1, l_2, l_3	-
1	0	0	0	l_2, l_3	l_1
2	-	8	**12**	l_2, l_3	l_3
3	-	**16**	0	l_3	l_2
4	-	-	**12**	Φ	l_3

Figure A.3. Steps in Gronkvist's algorithm [13] to construct the schedule.

Figure A.3 shows the steps (in timeslots) run by the Gronkvist's algorithm on the network on Figure A.1 in order to construct a schedule, in which $\boldsymbol{\Lambda_l \tau_l}$ is the link priority of link l used in the algorithm, where $\boldsymbol{\tau_l}$ is the number of timeslots since the link was previously allocated a timeslot. Also, the **List A** is the set of links that still has not been given all their guaranteed timeslots. Note that the algorithm stops when the list is empty. Finally, the **Schedule** indicates which links are to be scheduled in each timeslot $\boldsymbol{t}$.

From (A.2), we rewrite in more details the traffic loads λ_l scheduled on link l by our scheduling algorithm.

$$\lambda_{l_1} = 0.166666; \lambda_{l_2} = 0.333333; \lambda_{l_3} = 0.5 \text{ (packets/slot)} \quad \text{(A.9)}$$

From (A.4), we also rewrite in more details the total throughput λ made by our algorithm including traffic loads of three flows f_1, f_2 and f_3.

$$\lambda = \lambda_{l_3} = 0.5 \text{ (packets/slot)} \quad \text{(A.10)}$$

From the results in Figure A.3, we have the traffic loads λ'_l scheduled on link l by the Gronkvist's algorithm.

$$\lambda'_{l_1} = 0.25; \lambda'_{l_2} = 0.25; \lambda'_{l_3} = 0.5 \text{ (packets/slot)} \quad \text{(A.11)}$$

From (A.11), we also have the total throughput λ' made by the Gronkvist's algorithm including traffic loads of three flows f_1, f_2 and f_3.

$$\lambda' = \lambda'_{l_3} = 0.5 \text{ (packets/slot)} \quad \text{(A.12)}$$

Clearly, [13] fails to capture the traffic demands accurately, even though it is indeed collision-free.