In-network Computation in Sensor Networks

by

Rajasekhar R. Sappidi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2012

Author's Declaration

Abstract

Sensor networks are an important emerging class of networks that have many applications. A sink in these networks acts as a bridge between the sensor nodes and the end-user (which may be automated and/or part of the sink). Typically, convergecast is performed in which all the data collected by the sensors is relayed to the sink, which in turn presents the relevant information to the end-user. Interestingly, some applications require the sink to relay just a function of the data collected by the sensors. For instance, in a fire alarm system, the sinks needs to monitor the maximum of the temperature readings of all the sensors. For these applications, instead of performing convergecast, we can let the intermediate nodes process the data they receive, to significantly reduce the volume of traffic transmitted and increase the rate at which the data is collected and processed at the sink: this is known as in-network computation.

Most of the current literature on this novel technique focuses on asymptotic results for large networks and for very elementary functions. In this dissertation, we study a new class of functions for which we want to compute explicit solutions for networks of practical size. We consider the applications where the sink is interested in the first M statistical moments of the data collected at a certain time. The k-th statistical moment is defined as the expectation of the k-th power of the data. The $M = 1$ case represents the elementary functions like MAX, MIN, MEAN, etc. that are commonly considered in the literature. For this class of functions, we are interested in explicitly computing the maximum achievable throughput including routing, scheduling and queue management for any given network when in-network computation is allowed.

Flow models have been routinely used to solve optimal joint routing and scheduling problems when there is no in-network computation and they are typically tractable for relatively large networks. However, deriving such models is not obvious when in-network computation is allowed. Considering a single rate wireless network and the physical model of interference, we develop a discrete-time model for the real-time network operation and perform two transformations to obtain a flow model that keeps the essence of in-network computation. This model gives an upper bound on the maximum achievable throughput. To show the tightness of that upper bound, we derive a numerical lower bound by computing a feasible solution to the discrete-time model. This lower bound turns out to be close to the upper bound proving that the flow model is an excellent approximation to the discrete-time model.

We then adapt the flow model to a wired multi-rate network with asynchronous transmissions on links with different capacities. To compute the lower bound for wired networks, we propose a heuristic strategy involving the generation of multiple trees and effective queue management that achieves a throughput close to the one computed by the flow model. This cross validates the tightness of the upper bound and the goodness of our heuristic strategy. Finally, we provide several engineering insights on what in-network computation can achieve in both types of networks.

Acknowledgements

I wish to express my deepest gratitude to Prof. Catherine Rosenberg, my supervisor for her invaluable guidance and support. Her positive encouragement and vision has been the key to the successful completion of this thesis. I am also very thankful to Prof. André Girard (INRS-EMT), who has been of immense help throughout this thesis preparation.

I would also like to thank all the members of my committe, Prof. Patrick Mitran, Prof. Mahesh Tripunitara, Prof. Samir Elhedi and Prof. D. Manjunath (Indian Institute of Technology, Bombay) for their time and invaluable comments. I wish to extend my warmest thanks to all the members of our research group, past and present, for their kind support and encouragement.

I would also like to thank all my friends in Waterloo for their support and companionship. Finally, and most importantly, I wish to thank my parents, to whom I dedicate this thesis.

Rajasekhar Sappidi

Waterloo, 2012

Table of Contents

List of Tables

List of Figures

List of Symbols

n	Number of nodes
$\mathcal{V}$	Set of nodes
m	Number of links
$\mathcal{L}$	Set of links
d_0	Near-field crossover distance
η	Path loss exponent
P	Transmission power
$G_{i,j}$	Normalized channel gain between nodes i and j
N_0	Received background noise power
β	Required SINR for successful packet reception

Chapter 1

Introduction

A spatially distributed group of autonomous sensor nodes that collectively serve an application or applications by monitoring physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants, is commonly known as a sensor network [2]. A sensor node (see Figure 1.1) also known as a mote or a pod, is a smart device with a limited energy supply (usually a battery or a solar cell), limited memory and limited computational power with a capability to communicate with other nodes and with a base-station/sink. The sensor nodes are inexpensive and are easily manufactured in bulk. They can monitor many ambient parameters like temperature, humidity, pressure, speed and direction of the object it is attached to, soil makeup, etc. The sink has a more powerful processor with a larger power supply and a larger memory and acts as a relay conveying the information gathered by the sensor network to the end-user (which may be automated and/or part of the base-station/sink). Communication among the nodes or with the base-station happens via radio frequency or optical means using lasers or infrared.

This emerging new class of networks has many interesting applications and configuring them to obtain the best performance is of prime importance. In several applications, the sink requires a function of the information gathered by all the sensors for further decision

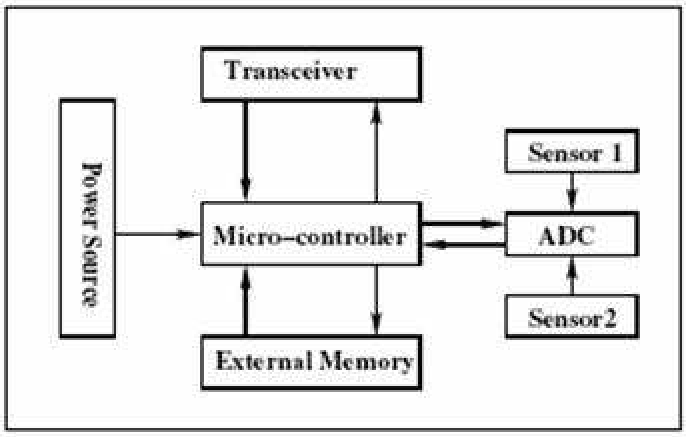


Figure 1.1: Block diagram of a typical sensor node

making, for instance, a fire-alarm system where the sink is monitoring the maximum temperature in a building. The standard approach known as *convergecast* or data collection in the literature, is to send all the individual measurements to the sink that then computes the required function. However, the volume of transmitted data can be significantly reduced by allowing the intermediate nodes to process the data they have received and send only an essential summary to the next node along the route to the sink. This is known as *in-network computation* in the literature [3]. This approach could potentially lead to significant improvements in the network performance in terms of lifetime, delay and throughput. Although, in-network computation is mainly advocated for the performance improvement of wireless sensor networks, in this dissertation, we also investigate its impact on wired networks.

The nature of the function that the sink is interested in computing plays an important role in determining the performance gains achievable utilizing in-network computation compared to convergecast. Computing the first M moments of the data is of interest in many sensor network applications in order to reconstitute the distribution of the measured phenomenon via its moment generating function. The accuracy of this process depends on

M with larger M resulting in better accuracy. We call these functions *statistical* functions because their main application is in the calculation of the statistical moments of the empirical distribution of the measured variables. An instance of an application is as follows. Consider a wireless sensor network which is deployed to measure the level of air pollution in a city at a given time. Factors such as wind, time of the day, cloud cover, etc., could effect the measurement at any given point. Hence, we measure the pollution at multiple points in the city at a given time. The average of the ensemble of data collected by all the sensors at a given time is a very accurate indication of the level of air pollution at that time in the city and the variance of this collected data shows how "non-uniform" the pollution is at that time. Note that in this example we are interested in measuring a single phenomenon (pollution) in a potentially noisy environment.

Formally, the k-th moment of a discrete random variable X, μ_k is defined as the expectation of X^k, i.e.,

$$\mu_k = E(X^k) \tag{1.1}$$

We can estimate μ_k as the average of the k-th powers of all the observed data, i.e.,

$$\mu_k \simeq \frac{\sum_{i=1}^{N} X_i^k}{N} \tag{1.2}$$

For any given network, we are interested in explicitly finding the optimal joint routing, scheduling and queue management that gives the maximum throughput (defined in the next section) when the sink is interested in computing the first M moments of the data collected by the sensor nodes and in-network computation is allowed. This leads to an optimal centralized solution. Although random access protocols have many desirable features such as robustness to node and link failures, low maintenance, etc., they are outperformed by centralized protocols, particularly in terms of throughput. Even if the optimal centralized protocol is not implementable in certain contexts, e.g., random deployment of sensor nodes in war zones, it can serve as a benchmark, i.e., provide an upper bound on the performance and guide the design of better distributed random access protocols.

In the next section, we explain the mechanism of in-network computation and how it leads to data aggregation.

1.1 In-Network Computation: Data Aggregation

Assume that there are n nodes in the network and that time is slotted. Also assume that every node i makes a measurement periodically after every δ time slots. Let the w-th measurement of node i be $x_i(w)$. Let $\mathbf{x}(w)$ denote the collection of the w-th measurements made by all the nodes. Assuming that all the nodes collect data at the same time, we call the collection $\mathbf{x}(w)$, the w-th *wave* of information. In applications like the fire-alarm system, the sink is interested only in some function $f(\mathbf{x}(w))$ of these measurements. We say that the sink has received the wave w if it is able to compute the function of the data in wave w. We define *throughput* as the rate at which the sink receives the waves. This throughput can be viewed as the "monitoring rate" of the phenomenon by the sensor network. However, we use throughput in this dissertation as it is the term traditionally used in convergecast.

Consider the instance when the sink is interested in $E[\mathbf{x}(w)]$, the average of the measurements in a wave over all nodes, i.e., $f(\mathbf{x}(w)) = \sum_i x_i(w)/n = S(w)/n$. In the convergecast mode, all the raw data $x_i(w)$ are individually sent to the sink where the average is computed. When in-network computation is allowed, each time a node i is allowed to transmit (as defined by the schedule), it will aggregate all the data of the wave, w, it has in its buffer (which may include its own data $x_i(w)$) into one (partial) sum $S_i(w)$ and send it as one packet to the next node along the path to the sink. Note that these partial sums must be computed only among the measurements made at the same time, i.e., each wave of information must be processed separately. Data from one wave cannot be aggregated with that of another wave, i.e., there must not be mixing among different waves. This is an important condition that we will have to take into account later on. Note also that care should be taken so that a raw measurement is not used more than once in the creation of partial sums. This requires that the routing has no loops and that a node empties its buffer of all the information that was used to create a partial sum after its transmission.

Computation of more complicated functions, such as the first two moments, i.e., variance and mean, can also be delegated to the nodes in the network. In this case, the sink needs two pieces of information: $S(w)$, the sum of the $x_i(w)'$s, and $R(w)$, the sum of the squares $x_i^2(w)$'s to compute them: $\text{var}(\mathbf{x}(w)) = \sqrt{\{R(w) - S(w)\}/n}$ and mean $= S(w)/n$. Thus, in-network computation creates two *types* of partial sums, i.e., a node i now computes not only the partial sum $S_i(w)$ but also the partial sum of squares $R_i(w)$ and transmits these two values and sends each value as one packet. Similarly, the sink can compute the first M moments of the data if it receives just the M sums of the first M powers of the data. The special case of $M = 1$ results in, what we call, perfect aggregation that covers functions like MAX and MIN, in addition to the first moment, e.g., MEAN.

Data aggregation is the main reason for allowing in-network computation since it can significantly reduce the total volume of the data transmitted compared to convergecast. This comes at the expense of an additional complexity since processing has to be performed at intermediate nodes. It is interesting to note that aggregation must not be performed automatically at all nodes when we want to compute more than one moment. To see this, consider a case where the sink is interested in the first two moments. Assume that some node i which is allowed to transmit has only its own data $x_i(w)$ in its buffer. If node i performs in-network computation blindly, it creates two packets corresponding to sum, $S_i(w) = x_i(w)$ and sum of squares, $R_i(w) = x_i^2$ and would require two time slots to send these packets. But, it is more efficient if i just sends its data as such and if in-network computation is delayed to further nodes along the path to the sink. This can be done easily by allowing the existence of three (instead of just two) different types of packets per wave, namely raw data, partial sum and partial sum of squares. More generally, when the sink is interested in computing the first M moments, there would be $M + 1$ different types of packets per wave.

In the next section, we summarize our contributions and give an overview of this dissertation.

1.2 Our Contributions

Instead of focusing on asymptotic results for large networks as is the current practice [3], we explicitly compute the maximum achievable throughput when the sink is interested in the first M moments of the collected data. We first focus on multi-hop single rate wireless networks. Flow models are routinely used in multi-hop wireless networks when there is no in-network computation and are typically tractable for relatively large networks. However, deriving such models is not obvious when in-network computation is allowed. We develop a discrete-time model for the real-time network operation and perform two transformations to obtain a flow model that keeps the essence of in-network computation. This gives an upper bound on the maximum achievable throughput. To show its tightness, we derive a numerical lower bound by computing a solution to the discrete-time model based on the solution to the flow model. This turns out to be close to the upper bound proving that the flow model is an excellent approximation to the discrete-time model.

We then adapt the flow model for a wired multi-rate network, whose underlying system is very different from that of the wireless network as the transmissions are not synchronized among different links that might have different capacities. Again, this gives an upper bound on the maximum achievable throughput and we propose methods which are different from those used for the wireless network, to obtain feasible solutions. We show via numerical results that these feasible solutions are near-optimal. We describe our methodology in more detail in Chapter 3.

In summary, our main contributions in this dissertation are:

1. A novel non-intuitive modeling of the in-network computation when the sink is interested in the first M moments.

2. A rigorous derivation of a flow model and a proof that its solution is an upper bound on the maximum achievable throughput using in-network computation.

3. Methods to compute a feasible solution and thus a lower bound on the maximum achievable throughput for both the wired and the wireless networks. These methods are also interesting in their own right as heuristics on how to operate the networks allowing in-network computation with near optimal performance.

4. Numerical evidence that the two bounds are close. This validates our flow model formulation and establishes the near-optimality of the feasible solutions.

5. Engineering insights and the quantification of the value of in-network computation.

Throughout the thesis, we use the physical interference model for the wireless network and we make no restricting assumptions on the channel gains beyond the fact that they should be quasi time-invariant.

1.3 Outline

The rest of the thesis is organized as follows. In Chapter 2, we review the related works. In Chapter 3, we rigorously derive the flow model for a wireless single rate network and then give an adaptation for a wired multi-rate network. In Chapters 4 and 5, we validate the flow models and provide several insights on the value and effects of in-network computation on the network performance for the two types of networks. We treat the wireless and the wired networks separately because the solution strategies for each of them are different. We conclude in Chapter 6 with several possible extensions of this work.

Chapter 2

Literature Review

In recent years, optimizing throughput or delay in *wireless sensor networks* for the two modes of data gathering, viz., convergecast [4–6] and in-network computation [3, 7–9] has received a lot of attention. On the other hand, for wired networks, although finding the maximum flow using convergecast has been well studied as the *evacuation problem* in the literature [10–12], in-network computation received little attention. The earliest study on in-network computation was made by Tiwari [13]. They studied the communication complexity, i.e., the number of bits needed to communicate the function to the sink in the worst case, in a two source network. Another early work is by Gallager [14], who proposed a distributed algorithm that computes the parity of the bits at the nodes in a broadcast network with binary symmetric channels with a given accuracy using only $O(\ln \ln N)$ bits per node.

In wired networks, to the best of our knowledge, [1] is the only other work (apart from ours) that considers finding the optimal routing explicitly when in-network computation is allowed. While we consider the case where the sink computes a statistical function, Shah et al [1] consider a different class of functions that could be represented by a computational schema. See Figure 2.1 which is taken from their work for an example of a schema. Their

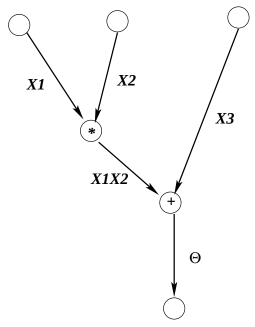


Figure 2.1: A schema to represent the function $\Theta = X_1X_2 + X_3$, *courtesy of* [1]

formulation is dependent on the schemas of the function and the computation time of their algorithm is linear on the number of schemas that can be used for computing the function. Thus, for a function with perfect aggregation property like AVERAGE or SUM which has an exponentially large number of computational schema representations, it is computationally difficult to find the optimal routings using their algorithm. In this thesis, we develop techniques that can address the problem of finding the optimal routing for the computation of functions with perfect aggregation. They are represented by $M = 1$ in this thesis and we find the maximum achievable data generation rate and the optimal routing not only for these functions but also for any M.

In-network computation has received much more interest in the wireless context and many different aspects of this problem have been studied in the literature. In these works, the two most commonly used models of wireless interference are the protocol model and the physical model [15]. In the protocol model, a receiver can successfully decode the message intended for it, if and only if it is within the transmission range of the sender and there are no other nodes transmitting within its interference range while in the physical model, a receiver can successfully decode if and only if its SINR exceeds a minimum threshold. In our work, we consider the physical model because it depicts the wireless interference more accurately [16, 17] than the protocol model. In Chapter 3, we describe this interference model in more detail.

Next, we discuss the work in wireless networks on throughput and delay with no in-network computation followed with the one that considers in-network computation.

(A) **Throughput with no in-network computation:** Gupta and Kumar [15] studied the capacity of a wireless network and gave asymptotic results for the achievable throughput under both the protocol and physical models of interference. Tassiulas and Ephremides [18] have proposed the *backpressure* algorithm for scheduling in wireless multi-hop networks and proved that it is throughput-optimal. Jain *et al* [5], Studi *et al* [6], Karnik *et al* [4] have all studied the problem of explicitly finding the maximum achievable throughput of a fixed wireless network. Lower and upper bounds on the maximum achievable throughput are obtained in [5]. A linear program formulation for explicitly computing the optimal throughput of a network under the physical interference model is given in [4] and Luo *et al* [19] used linear programming techniques like column generation to solve this problem efficiently for larger networks. In this thesis, we are interested in a similar formulation when the network allows in-network computation.

(B) **Delay with no in-network computation:** Another performance measure that is considered in the literature is delay. Florens *et al* [20] considered the minimization of delay for the convergecast problem under a protocol interference model. They assume that there is a single communication channel and that time is slotted such that every slot is long enough to send just one packet of data. They give closed form expressions for delay in terms of the number of time slots for some special topologies like line, multi-line, etc. However, they have considered only a restricted set of network topologies where the distance between any two neighboring nodes is the same. Gargano and Rescigno [21] also consider the problem of minimizing delay in the convergecast problem. They assume that the nodes have directional antennas so that any two links can be active at the same time if they do not have any nodes in common. In effect, the interference model they consider uses only transceiver constraints, i.e., a node can either receive or transmit but cannot do both at the same time and if it is receiving, it can receive from only one node. They present an algorithm using collision-free graph coloring that gives the optimal solution for the overall delay. Chen *et al* [22] also studied the problem of minimizing the delay under a variation of the protocol interference model. They showed that finding the optimal schedule for a general graph is NP-hard and proposed an heuristic that performs better than the RTS/CTS scheme. In our thesis, although the discrete-time model we formulate could be used to find the optimal schedule that minimizes delay, our focus is on computing the maximum achievable throughput when the network allows in-network computation.

(C) **Throughput with in-network computation:** Giridhar and Kumar [3] studied the problem of distributively computing the maximum rate at which symmetric functions of sensor measurements can be computed and communicated to the sink. They classified the symmetric functions into two subclasses, type-sensitive and type-threshold. They show that the type-sensitive functions like mean, mode, median can be computed at $O(1/n)$ rate in collocated networks[1] and at $O(1/\log n)$ rate in random planar multihop networks while the type-threshold functions like min, max, range can be computed at $O(1/\log n)$ rate in collocated networks and at $O(1/\log\log n)$ rate in random planar multihop networks. Note however that these sub-classes do not cover all the symmetric functions. These are asymptotic results and they do not compute the maximum

[1]In a collocated network, transmissions from any node are received at all the nodes, including the sink. If protocol model of interference is assumed, then only one node can successfully transmit at a time. Hence, no in-network computation is possible for a function like mean.

achievable throughput explicitly which is the problem we consider in this thesis. To the best of our knowledge, there is no other past literature on non-asymptotic results when in-network computation is allowed.

(D) **Delay with in-network computation:** Sudeep and Manjunath [23] studied the problem of computing MAX in an unstructured sensor network assuming the protocol model of interference. In an unstructured network, the nodes need not have unique identifiers. They show that in $O(\sqrt{n/\log n})$ slots, MAX can be made available at the sink with high probability. This is interesting because the best coordinated protocol given in [24] also takes the same $O(\sqrt{n/\log n})$ slots to compute MAX. Most-Aoyama and Shah [25] give an elegant distributed scheme based on exponential random variables to compute the class of *separable* functions such as SUM in a wireless network using gossip protocols. Ghosh *et al* [9] have considered in-network computation where the sink is interested in a function like mean, min or max, i.e., aggregation results in a single packet regardless of the number of input packets (perfect aggregation). They assume multiple channels and the protocol model of interference and give some tree-based heuristics to minimize delay and maximize throughput for the convergecast and the perfect aggregation problems. There is a lot of similar recent work on perfect aggregation [26–30] that consider the problem of minimizing delay. In this thesis, we consider aggregation when the sink is interested in the first M moments, of which the perfect aggregation is an example.

(E) **Others:** The modeling of compressed sensing in [31] is very similar to the way we model the computation of M moments. Using compressed sensing, the sink could retrieve all the raw data unlike in our current problem where it only retrieves the

M moments of the data. However, they consider minimizing energy consumption as their performance measure and do not consider finding the optimal joint routing and scheduling that maximizes throughput which we consider.

To the best of our knowledge, there are no previous results on explicitly computing the maximum achievable throughput of a given network that allows in-network computation in a wireless network.

Chapter 3

Problem Formulation

In this chapter, we define and formulate the problem of optimizing the throughput when the sink is interested in the first M moments of the data collected by the sensor nodes.

3.1 System Model

Recall from Section 1.1, that all the sensors collect a new raw data every δ units of time. In other words, new raw data is being generated by each of the sources at a rate of λ where $\lambda = 1/\delta$. This is the data generation rate of the sources. We assume that each of the nodes in the network can be uniquely identified. We also assume that there is a background mechanism that keeps the clocks of the sources synchronized so that they all collect new data at the same time.

Also recall that, we call the vector of data collected at the same time $\mathbf{x}(w)$, the w-th *wave* of information and consider applications where the sink is interested only in some function $f(\mathbf{x}(w))$ of these measurements. We say that the sink has received the wave w if it is able to compute the function of the data in wave w and define *throughput* as the rate at which the sink receives the waves. For the system to perform correctly, the sink needs to receive all the necessary data to compute the function, at the same rate at which the new data is being generated at the sources. Thus, the throughput of the system is precisely equal to the data generation rate of the sources. However, this does not imply that the sources have to wait until the sink receives the current wave before they collect a new wave of data. It is interesting to note that when the sink is receiving data of the wave w, it is possible that some nodes (particularly the ones farthest from the sink) may already be relaying data belonging to the later waves. This is known as the *pipelining* effect in which several waves of data are being transmitted at the same time in the network. Due to this effect, minimum delay (defined as the time between the collection of the data of a wave at the sources and its reception at the sink) is not equal to the inverse of the maximum throughput. In this work, we explicitly compute the maximum achievable throughput (or equivalently the maximum data generation rate) supported by a given network when the sink needs to compute a given statistical function, i.e., the first M moments. This computation implicitly assumes the pipelining effect.

As the sink is interested just in the first M moments and in-network computation is allowed, the intermediate nodes can perform partial computations on the data they receive. Suppose a node is on the path to receive $x_{i_1}(w), x_{i_2}(w) \dots x_{i_k}(w)$, then it could combine these data into M partial sums defined as $S_p(w) = \sum_{j=1}^{k} x_{i_j}^p, \quad \forall p = 1 \dots M$ and forward only these M packets instead of the k packets it received. Note that this kind of aggregation is beneficial only when $k > M$.

Consider the network in Figure 3.1. Let all the nodes other than the sink be sources in this network. Assume that the sink is interested in the first two moments of the data generated by these sources so that it can compute the mean and the variance. The sink can compute these quantities if it receives all the data generated by the sources for a given wave but this is a strain on the network resources. Instead, with in-network computation, node D can be an aggregator, i.e., it can perform partial computations and aggregate the

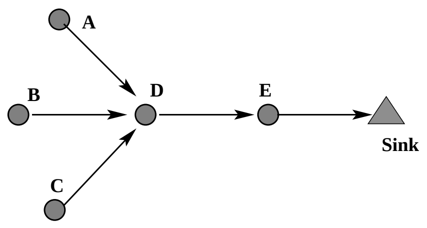


Figure 3.1: A network to illustrate in-network computation

packets it receives before forwarding. In order to minimize the total traffic carried in the network and to take full advantage of in-network computation, node D would have to wait until it receives the data of wave w from A, B and C before it sends any data for wave w to E. Similarly, node E would have to wait to receive the data of wave w from node D before forwarding any data from wave w to the sink. Note that if the nodes do not wait, they might not be able to aggregate and the purpose of in-network computation would be forfeited. The fact that waiting can be beneficial needs to be modelled carefully.

Under the assumption that it waits, node D, after it receives the raw packets $x_A(w), x_B(w)$ and $x_C(w)$ from A, B and C respectively, creates two partial sum packets, one for the sum of data, $S_1^D(w) = x_A(w) + x_B(w) + x_C(w) + x_D(w)$ and another for the sum of the squares, $S_2^D(w) = x_A^2(w) + x_B^2(w) + x_C^2(w) + x_D^2(w)$. It then sends only these two partial sum packets to node E. Node E now combines its own data, $x_E(w)$ with these two partial sum packets as $S_1^E(w) = S_1^D(w) + x_E(w)$ and $S_2^E(w) = S_2^D(W) + x_E^2(w)$ and sends only these two partial sum packets to the sink which now has all the information to compute the two moments. In this example, by waiting we have minimized the amount of traffic carried by each link to a maximum of two packets. If we assume that the links are wireless with a unit rate (in packets per second) and that no two links can be scheduled at the same time, we achieve a throughput of 1/7 using in-network computation as opposed to a throughput of 1/12 using convergecast. This example shows that in-network computation could indeed lead to significant improvements in network performance with appropriate scheduling, queue management (which involves waiting) and routing. For more complex networks, the real challenge is in finding the optimal joint routing and scheduling along with the queue management strategy that optimally utilizes this potential of in-network computation.

More generally, we see that in-network computation leads to the creation of new types of packets, i.e., the partial sums. If the sink is interested in the first M moments, the partial computations at the intermediate nodes (when performed) lead to M new types of packets in addition to the raw data. We call the packet containing the partial sum of the p-th powers of the data in the same wave as a packet of type p $(1 \leq p \leq M)$ and by convention type 0 represents the raw data.

For any given network with in-network computation modelled as above, we are not only interested in finding the maximum data generating rate that could be supported, i.e., the throughput using in-network computation but also in finding a solution detailing the operation of the network (routing, scheduling and queue management) that achieves this rate or close to it when implemented in a packet-based network.

In the next section, we present a systematic approach to solve this problem in a wireless single rate network. We then consider the multi-rate wired networks. We begin by introducing the network model and the physical model of wireless interference.

3.2 Single Rate Wireless Network Model

We assume that there is a static wireless sensor network with n nodes and that the channel gains are quasi time-invariant. In addition, there is a special node called the *sink* that collects and analyzes the data from all the sensor nodes. We also assume that the sink knows the unique identities of all the n nodes. Let them be labeled $0, 1, \ldots n$ with 0 representing the sink. We assume that there is a single communication channel and every node communicates with its neighbors through a transceiver which can either transmit or receive but not do both at the same time. We assume that the time is slotted and each slot is long enough to transmit one packet of data. We assume that the data collected by the sensor nodes is real numbers each of which fits in a single packet. We also assume that all the arithmetic operations that we do on the data result in a real number which remains within the allowed range when represented in the binary format and thus still fits in a single packet of data. For example, if $x_1(w), x_2(w)$ and $x_3(w)$ are three real numbers sensed by three different sensors for wave w, then, by our assumption $S_k(w) = x_1(w)^k + x_2(w)^k + x_3(w)^k$ can be transmitted using just one packet. We do not consider power control and assume that all the nodes transmit with the same power P. We model the wireless interference using the physical model which is based on Signal to Interference and Noise Ratio (SINR) [4, 15]. Experimental results [16] show that this models wireless interference more accurately than any other simpler model. Also, Iyer *et al.* [17] showed that the results obtained from simpler interference models may be qualitatively different from those obtained from the physical model. However, note that the techniques we present could be used to obtain the flow model for any given interference model.

A directed link from node i to node j is said to exist if $P_{i,j} \geq \beta N_0$ where $P_{i,j}$ is the total power received from node i at node j. Here N_0 is the thermal noise power and β is the minimum Signal to Noise Ratio (SNR) required for successful decoding of the message. Given a channel propagation model for $P_{i,j}$, we can compute the set of all feasible directed links $\mathcal{L}$ in the network using this SNR condition. We make no restricting assumptions on the channel propagation model.

In a wireless network, even though all the links use the same channel, we can typically schedule a group of links to transmit at the same time without causing excessive interference to any intended receivers. We call such a subset of links an *independent set* (Iset). The interference model dictates which subset of links can be successfully activated at the same time. In the physical interference model, when two or more links are active on the same channel, every receiver considers the power from the transmitters other than its own as interference.

Under the physical interference model, a subset of feasible links, I, is an Iset only if they form a matching i.e.,

$$i \neq i' \wedge i \neq j' \wedge j \neq i' \wedge j \neq j' \quad \forall (i,j),(i',j') \in I \tag{3.1}$$

and all the corresponding receivers have an SINR greater than or equal to β i.e., for all the links (i,j) in the Iset I,

$$P_{i,j} \geq \beta \left[N_0 + \sum_{(i',j') \in I \setminus \{(i,j)\}} P_{i',j} \right]. \tag{3.2}$$

The sum in (3.2) is the total interference received by the destination node j of link (i,j) due to the transmissions on all other links (i',j') in I.

In the next section, we discuss the methodology we adopt and the reasons behind it to solve the problem of computing the maximum achievable throughput using in-network computation in a given wireless sensor network.

3.3 Methodology

The network model described above naturally results in a discrete-time model formulation for computing the maximum achievable throughput for convergecast. However, the discrete-time model is typically an integer program that is not tractable and is solvable only for very small networks. For this reason, the standard approach taken in the literature is to ignore the discrete nature of the packets and assume that packets flow on links like fluids flow in pipes. This is known as the *flow model* and it does not require the notion of time slots and waves. In the following, we say that a problem formulation is a flow model of the network operation (e.g., convergecast or in-network computation) if it can be used to compute the maximum achievable throughput (exactly or approximately) and it does not include the notion of time slots or of waves. Flow models have been typically derived from intuition without any reference to the underlying discrete-time model. The nonlinear multi-commodity flow model used to compute the minimum average packet delay in wired networks [32] is an example of such a model. Deriving a flow model for convergecast in wireless networks is also quite intuitive and they have worked well to explicitly compute the maximum achievable throughput [4, 5, 19]. In all these cases, solving the flow model has brought insights on the optimal network structure and operation that would have been impossible to obtain from the discrete-time model as we cannot solve it within a reasonable time.

Trying to derive a flow model straight away for in-network computation problem does not result in a tractable accurate formulation. The reasons are

1. There is no conservation of flows at the nodes since some packets effectively disappear when they are aggregated.

2. Aggregation can happen only between data belonging to the same wave. And this is not easy to ensure using flows.

Hence, we have to start by formulating the problem using the discrete-time model. There is more than one way to formulate a discrete-time model. The straightforward and intuitive way to formulate a discrete model for the case of $M = 1$ is based on keeping track of the number of packets on a per-wave basis at each node. This led to constraints with product forms, i.e., non-linear constraints. We found that this problem is not tractable even for small networks. In one of our attempts to simplify this formulation, we tried to linearize these constraints but this led to a very poor approximation of the original problem. Also, as the number of required moments M increased, the complexity of these constraints increased as well.

We have been able to overcome these difficulties by taking an implicit modelling approach which is a completely different view regarding the modeling of in-network computation in a sensor network. It is based on *tracking* each measurement, also called a piece of information until it reaches the sink as opposed to the conventional method that focuses on tracking packets at every node. In this problem, although packets are not conserved, information is. To understand how we model the in-network computation, consider the example of computing the *mean*, i.e., $f(\mathbf{x}(w)) = \sum_i x_i(w)/n$. Whenever a node has more than one packet of data from the same wave, it computes their sum and transmits only the sum when one of its link is activated. In our approach, although only the sum is being transmitted, we track all the information units contained in the sum, i.e., we model the process *as if* each information unit contained in the sum is transmitted in the same slot. We capture this in our model by allowing the node to create a *virtual information* unit per wave for every source that is involved in the sum and to transmit all the *virtual information* units in the same time slot. The model thus keeps track of all the information units until they reach the sink. Note that if the node is transmitting raw information, it can only send raw data from one source in one time slot. This novel modeling technique results in a discrete-time model which is a linear integer program, where the conservation property holds (nothing is lost, everything is tracked and is applied to information as opposed to packets).

However, this integer program is still hard to solve. Thus, we transform it into a flow model which could be solved optimally for much larger networks than that is possible for the discrete-time model. We show that this flow model provides an upper bound on the maximum achievable throughput. To validate the tightness of this upper bound, in Chapter 4, we derive a feasible solution to the discrete-time model based on the solution to the flow model and show that it yields a throughput that is close to the optimal throughput computed with the flow model. In summary,

1. We formulate a novel information-based discrete time model and rigorously derive a flow model formulation from it.

2. We show that this flow model provides an upper bound on the maximum achievable throughput.

3. We show that this upper bound is tight by computing a near-optimal feasible solution to the system operation based on the solution to the flow model. This validates our flow model and enables us to use it to study the effects of in-network computation on network performance.

4. When we apply our methodology (i.e., formulating the discrete model, followed by the derivation of the flow model) to convergecast, it resulted in the same flow model as the one given in the literature [4]. So, our method is consistent with what has been done intuitively in the past.

5. We also show that when $M = n$, where n is the number of nodes in the network, the flow model formulation for in-network computation is the same as the convergecast formulation. This further strengthens the confidence in our model as the $M = n$ case is intuitively convergecast.

Next, we present the discrete-time model formulation.

3.4 The Discrete Model

Assume that we want to compute the number of time slots T_m, taken to receive m waves at the sink. Recall that we say that the sink has received the wave w if it is able to compute the function of interest of the data in wave w. Also, recall that for every wave, there are $M + 1$ different types of data in the network, viz., the raw data (represented with $p = 0$), the partial sums (represented with $p = 1$), the partial sum of squares (represented with

$p = 2$) and so on. Define the state variable $q_i^{s,w,p}(k)$ as a binary variable that is 1 if the information[1] from source s, wave w and type p is stored at node i at the end of time slot k. Hence, $q_i^{s,w,p}(k)$ tracks the information (raw or virtual) on a per wave, per source and per type basis. By convention, we assume that it is 0 when $k = 0$.

Assume that the sensors are collecting data every δ slots, i.e., $\sigma = 1/\delta$ is the input rate of the information. Let $v^w(k)$ be a binary value that is 1 if the sources collect the data of wave w at the end of time slot k. Thus, we have

$$v^w(k) \triangleq \begin{cases} 1 & \text{if } k = \delta w \\ 0 & \text{otherwise} \end{cases}$$

The decision variables in the model are all binary variables with the following definitions:

1. $x_{i,j}(k)$: If the link (i, j) is active in slot k, it is 1, otherwise it is 0. This is the scheduling variable.

2. $y_{i,j}^{s,w,p}(k)$: If the information from source s, wave w and type p is sent on link (i, j) in slot k, it is 1, else 0.

3. $u_i^{s,w}(k)$: This variable tracks if node i aggregates raw data from source s and wave w at the end of time-slot k, i.e., in the model if it is 1, M virtual information units are created for this source out of the raw data.

4. $z_{i,j}^{w,p}(k)$: It is 1 if the information of type p in wave w is selected for transmission on link (i, j) in slot k, else 0.

By convention, the x, y and z variables are defined only when the link (i, j) exists.

With these definitions, the discrete-time model formulation $\mathcal{P}_d(m)$ is as follows. All constraints are over $w = 1 \ldots m$, $s = 1 \ldots n$ and $k = 1 \ldots T_m$ wherever w, k or s appear,

[1]The information is raw data if $p = 0$ and virtual information if $p > 0$

unless there is a summation over them. The constraints are also either over all nodes $i = 1 \ldots n$ or over all links $(i, j) \in \mathcal{L}$ depending upon the constraint. Also, all the variables are non-negative. Recall that all the nodes except the sink collect data.

$$\mathcal{P}_d(m): \qquad \underset{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{u}}{\textbf{Minimize}}\ T_m \tag{3.3}$$

subject to

$$T_m \geq \delta m \tag{3.4}$$

$$q_i^{s,w,p}(T_m) = 0 \tag{3.5}$$

$$\begin{aligned} q_i^{i,w,0}(k) &= v^w(k-1) + q_i^{i,w,0}(k-1) - u_i^{i,w}(k) \\ &\quad - \sum_j y_{i,j}^{i,w,0}(k) + \sum_j y_{j,i}^{i,w,0}(k) \end{aligned} \tag{3.6}$$

$$\begin{aligned} q_i^{s,w,0}(k) &= q_i^{s,w,0}(k-1) - u_i^{s,w}(k) \\ &\quad - \sum_j y_{i,j}^{s,w,0}(k) + \sum_j y_{j,i}^{s,w,0}(k) \quad \forall i \neq s \end{aligned} \tag{3.7}$$

$$\begin{aligned} q_i^{s,w,p}(k) &= q_i^{s,w,p}(k-1) + u_i^{s,w}(k) \\ &\quad - \sum_j y_{i,j}^{s,w,p}(k) + \sum_j y_{j,i}^{s,w,p}(k) \\ &\quad \forall p = 1 \ldots M \end{aligned} \tag{3.8}$$

$$y_{i,j}^{s,w,p}(k) \leq q_i^{s,w,p}(k-1) \tag{3.9}$$

$$\sum_{w=1}^{m} \sum_{p=0}^{M} z_{i,j}^{w,p}(k) \leq x_{i,j}(k) \tag{3.10}$$

$$\sum_{s=1}^{n} y_{i,j}^{s,w,0}(k) \leq z_{i,j}^{w,0}(k) \tag{3.11}$$

$$y_{i,j}^{s,w,p}(k) \leq z_{i,j}^{w,p}(k) \quad \forall p = 1 \ldots M \tag{3.12}$$

$$\sum_{(i,j)} x_{i,j}(k) + \sum_{(j,i)} x_{j,i}(k) \leq 1 \tag{3.13}$$

$$\begin{aligned} Q\left(1 - x_{i,j}(k)\right) + P_{i,j} &\geq \beta \times \\ &\left[N_0 + \sum_{(i',j') \in I \neq (i,j)} P_{i',j} x_{i',j'}(k) \right] \end{aligned} \tag{3.14}$$

Note that in this model, we do not explicitly force the in-network computation to happen at any node. The solution of $\mathcal{P}_d(m)$ with the optimal throughput decides the nodes at which it is most conducive for the aggregation of data to happen. The constraints (3.5–3.8) are the information conservation equations which guarantee that all the information reaches the sink. Constraint (3.4) ensures that the network operates at least as long as is necessary to generate all the waves.

Constraint (3.5) is the final target state where all the information units have reached the sink. Constraints (3.6–3.8) keep track of the flows of information. Whenever $u_i^{s,w}(k) = 1$, the raw information from source s and wave w disappears at node i which behaves as an aggregator for this information, and appears at node i as M higher types of data for that source and wave. This is how the information units, for the aggregated data are created.

The actual constraints on the decisions to transmit are modelled by (3.9–3.12). Constraint (3.9) states the fact that we cannot transmit a unit of information on a link if this information is not in the buffer at the end of the previous time slot. Constraint (3.10) states that information of at most one type of one wave can be transmitted in one time slot by a node.

The actual effect of aggregation is represented by (3.11–3.12). Recall that some data might be transmitted as raw data without aggregation. If there is no aggregation, a packet carries the information for a single source. In other words, if we decide to transmit a commodity $(s, w, 0)$, we must choose a single value for s. This is modelled by constraint (3.11) which restricts the flow of raw data on every link to utmost one packet in each time slot, so that raw data is transmitted like in convergecast. When we use aggregation, on the other hand, a packet can carry more than one virtual information unit for a given commodity (s, w, p), for instance the aggregated values of the measurements of some subset of sources. This is modelled by constraint (3.12), which allows multiple virtual information units from different sources to be sent on the same packet if they belong to the same aggregated type and wave (for $p \geq 1$). This constraint essentially captures the effect of in-network computation. Comparing constraints (3.11) and (3.12), we can see the advantage of aggregation which permits more than one unit of information to be carried on a given packet.

Finally, constraints (3.13) and (3.14) represent the physical model of wireless interference. In (3.14), Q is a very large positive constant [33]. The magnitude of Q is chosen so that constraint (3.14) is trivially true whenever any link (i, j) is inactive, i.e., $x_{i,j}(k) = 0$. On the other hand, if the link is active, constraint (3.14) reduces to (3.2) which is the minimum SINR requirement constraint of the physical model.

Let T_m^d be the optimal solution to problem $\mathcal{P}_d(m)$. This problem is a very large integer linear program that cannot be solved optimally except for very small networks and even then, with large computation times. Thus, we are interested in obtaining a problem formulation that is based on a flow model and hopefully is more tractable. The two dimensions, time and waves are the primary sources of complexity in $\mathcal{P}_d(m)$. We transform $\mathcal{P}_d(m)$ such that both time and waves disappear resulting in a flow model formulation.

We use two types of transformations on $\mathcal{P}_d(m)$ to derive the flow model formulation. They are:

1. Averaging over time.

2. Bundling of waves.

We explain in detail how they are performed in the next two sections.

3.5 First Transformation: Averaging Over Time

The idea is to sum the constraints over time and replace the variables with averages. Let

$$x_{i,j} \triangleq \frac{1}{T_m} \sum_{k=1}^{T_m} x_{i,j}(k) \tag{3.15}$$

$$y_{i,j}^{s,w,p} \triangleq \frac{1}{T_m} \sum_{k=1}^{T_m} y_{i,j}^{s,w,p}(k) \tag{3.16}$$

$x_{i,j}$ represents the fraction of time link (i, j) is active while $y_{i,j}^{s,w,p}$ represents the fraction of time information from source s of wave w and type p is flowing on the link (i, j). We define similar averages for the variables u and z as well. Note that as the input is just one packet for every wave at every source, we have $\sum_{k=1}^{T_m} v^w(k) = 1$.

Averaging out the constraints (3.6–3.12) using these definitions is straightforward. However, time also appears in the interference constraints (3.13–3.14). It is possible in principle to average them too but the resulting flow model is very inaccurate since the interference model is averaged out. Instead, we reformulate these constraints using the extensive formulation of the interference constraints. First, we generate all the ISets compatible with (3.13–3.14). This is equivalent to computing the link-Iset incidence matrix

$$A_{(i,j),I} \triangleq \begin{cases} 1 & \text{if link } (i,j) \in I \\ 0 & \text{otherwise} \end{cases}$$

We then replace constraints (3.13–3.14) by introducing another set of variables $\alpha_I(k)$, a binary variable that is 1 if we use Iset I in slot k and 0 otherwise. The modified interference constraints are

$$\sum_I \alpha_I(k) \leq 1 \tag{3.17}$$

$$x_{i,j}(k) \leq \sum_{I} \alpha_I(k) A_{(i,j),I} \tag{3.18}$$

Constraint (3.17) ensures that only one Iset is chosen in slot k and constraint (3.18) allows a link to be active in slot k if and only if an Iset containing it is active in that slot. Note that all the links in any given Iset satisfy the interference constraints by construction and hence the original interference constraints (3.13–3.14) are redundant in the problem formulation and hence can be eliminated.

For the purpose of averaging, we then define

$$\alpha_I \triangleq \frac{1}{T_m} \sum_{k=1}^{T_m} \alpha_I(k)$$

α_I represents the fraction of time Iset I would be active in the schedule. The discrete time dimension disappears from the constraints (3.5–3.12) and (3.17–3.18) when we sum each of them over k and divide with T_m, leading to problem $\mathcal{P}_2(m)$. Note that after this summation, all $q_i^{s,w,p}(k)$ variables cancel from the equations (3.6), (3.7) and (3.8), except when $k = 0$ and $k = T_m$. When $k = T_m$, the q's are 0 due to the final condition and when $k = 0$, the q's are 0 by convention. The only non-zero entity in the R.H.S of constraints (3.21) and (3.22) in $\mathcal{P}_2(m)$ is due to the summation $\sum_{k=1}^{T_m} v^w(k)$ which is equal to 1. All constraints are over all $w = 1 \ldots m$ and all $s = 1 \ldots n$ wherever w, or s appear, unless there is a summation over them. The constraints are also either over all nodes $i = 1 \ldots n$ or over all links $(i,j) \in \mathcal{L}$ depending upon the constraint. Also, all the variables are non-negative.

$$\mathcal{P}_2(m): \qquad \underset{\boldsymbol{\alpha},\mathbf{y},\mathbf{z},\mathbf{u}}{\textbf{Minimize}}\ T_m \tag{3.19}$$

subject to

$$T_m \geq \delta m \tag{3.20}$$

$$\sum_j y_{i,j}^{s,w,0} - \sum_j y_{j,i}^{s,w,0} + u_i^{s,w} = \begin{cases} \dfrac{1}{T_m} & \text{if } i = s \\ 0 & \text{otherwise} \end{cases} \tag{3.21}$$

$$\sum_j y_{i,j}^{s,w,p} - \sum_j y_{j,i}^{s,w,p} - u_i^{s,w} = 0 \quad \forall p = 1 \dots M \tag{3.22}$$

$$\sum_{w=1}^{m} \sum_{p=0}^{M} z_{i,j}^{w,p} \leq x_{i,j} \tag{3.23}$$

$$\sum_{s=1}^{n} y_{i,j}^{s,w,0} \leq z_{i,j}^{w,0} \tag{3.24}$$

$$y_{i,j}^{s,w,p} \leq z_{i,j}^{w,p} \quad \forall p = 1 \dots M \tag{3.25}$$

$$x_{i,j} \leq \sum_I A_{(i,j),I} \alpha_I \tag{3.26}$$

$$\sum_I \alpha_I \leq 1 \tag{3.27}$$

We can interpret the left-hand side of equation (3.21) as the average information rate out of node i of wave w. It shows that in the time-averaged model, this rate is the same for all sources s and all waves w since the right-hand side does not depend on either of these indices. It is equal to the inverse of the total time T_m if i is the source of information s and 0 otherwise. Replacing $1/T_m$ with λ_m and changing the objective to maximize λ_m, we have a linear program. Note however that because of constraint (3.20) and since the number of constraints depends on m, this problem is not independent of m, the number of waves.

Since a feasible solution to $\mathcal{P}_2(m)$ (3.19–3.27) can be constructed from any feasible solution to $\mathcal{P}_d(m)$ (3.3–3.14), we have

$$T_m^2 \leq T_m^d$$

where T_m^d is the optimal solution to $\mathcal{P}_d(m)$ and T_m^2 is the optimal solution to $\mathcal{P}_2(m)$.

Next, we transform $\mathcal{P}_2(m)$ into $\mathcal{P}_f$ by bundling the waves. This gives us the flow model for computing the maximum achievable throughput using in-network computation.

3.6 Second Transformation: Bundling of Waves

If T_m^d is the optimal solution to $\mathcal{P}_d(m)$ then m/T_m^d is the optimal rate at which the statistical function of each of the m waves is made available at the sink. Bundling the waves of $\mathcal{P}_2(m)$ removes the dependence on m and formulates the problem as maximizing this rate. For this, we define new variables that bundle the waves together, i.e.,

$$\sum_{w=1}^{m} y_{i,j}^{s,w,p} \triangleq y_{i,j}^{s,p}$$

where $y_{i,j}^{s,p}$ represents the total amount of information flow from source s, type p and all waves on link (i,j). We define similar variables for z and u as well. The waves lose their identity with the introduction of these variables. We sum constraints (3.21–3.25) over w and obtain problem $\mathcal{P}_3$. Again, all constraints are over all $s = 1 \ldots n$ and all $k = 1 \ldots T_m$ wherever k or s appear. The constraints are also either over all nodes $i = 1 \ldots n$ or over all links $(i,j) \in \mathcal{L}$ depending upon the constraint. Also, all the variables are non-negative.

$$\mathcal{P}_3 : \qquad \underset{\boldsymbol{\alpha},\mathbf{y},\mathbf{z},\mathbf{u}}{\textbf{Minimize}}\; T_m \tag{3.28}$$

subject to

$$T_m \geq \delta m \tag{3.29}$$

$$\sum_j y_{i,j}^{s,0} - \sum_j y_{j,i}^{s,0} + u_i^s = \begin{cases} \dfrac{m}{T_m} & \text{if } i = s \\ 0 & \text{otherwise} \end{cases} \tag{3.30}$$

$$\sum_j y_{i,j}^{s,p} - \sum_j y_{j,i}^{s,p} - u_i^s = 0 \quad \forall p = 1 \ldots M \tag{3.31}$$

$$\sum_{p=0}^{M} z_{i,j}^p \leq x_{i,j} \tag{3.32}$$

$$\sum_{s=1}^{n} y_{i,j}^{s,0} \leq z_{i,j}^0 \tag{3.33}$$

$$y_{i,j}^{s,p} \leq z_{i,j}^p \quad \forall p = 1 \ldots M \tag{3.34}$$

$$x_{i,j} \leq \sum_I A_{(i,j),I}\alpha_I \tag{3.35}$$

$$\sum_I \alpha_I \leq 1 \tag{3.36}$$

The right-hand side of Eq. (3.30) can be interpreted as the number of waves flowing out of node i per unit time when the node i is the source of information s. It is the average rate of information flow out of it. We replace it as $m/T_m \triangleq \lambda_m$ and define the objective as the maximization of this rate. With this replacement, the constraint (3.29) changes to

$$\lambda_m \leq \frac{1}{\delta} \tag{3.37}$$

This implies that the output rate cannot exceed the input rate which is as expected. The constraints and the variables are no longer dependent on m and we can replace λ_m

with just λ. As we are interested in computing the maximum achievable throughput, it is customary to remove constraint (3.29) (which corresponds to constraint (3.37)).

Thus, the final flow model formulation $\mathcal{P}_f$ is as follows.

$$\mathcal{P}_f : \qquad \underset{\boldsymbol{\alpha},\mathbf{y},\mathbf{z},\mathbf{u}}{\textbf{Maximize}}\ \lambda \tag{3.38}$$

subject to

$$\sum_j y_{i,j}^{s,0} - \sum_j y_{j,i}^{s,0} + u_i^s = \begin{cases} \lambda & \text{if } i = s \\ 0 & \text{otherwise} \end{cases} \tag{3.39}$$

$$\sum_j y_{i,j}^{s,p} - \sum_j y_{j,i}^{s,p} - u_i^s = 0 \quad \forall p = 1 \dots M \tag{3.40}$$

$$\sum_{p=0}^{M} z_{i,j}^p \leq x_{i,j} \tag{3.41}$$

$$\sum_{s=1}^{n} y_{i,j}^{s,0} \leq z_{i,j}^0 \tag{3.42}$$

$$y_{i,j}^{s,p} \leq z_{i,j}^p \quad \forall p = 1 \dots M \tag{3.43}$$

$$x_{i,j} \leq \sum_I A_{(i,j),I}\alpha_I \tag{3.44}$$

$$\sum_I \alpha_I \leq 1 \tag{3.45}$$

Let λ^f be the optimal solution to problem $\mathcal{P}_f$. Given a feasible solution to $\mathcal{P}_2(m)$ (3.19–3.27), we can construct a feasible solution to the flow problem $\mathcal{P}_f$ (3.38–3.45) so we have

$$\frac{m}{\lambda^f} \leq T_m^2 \leq T_m^d \tag{3.46}$$

This inequality (3.46) proves that the throughput computed by the flow model $\mathcal{P}_f$ is an upper bound on the maximum achievable throughput using in-network computation when the sink is interested in the first M moments. Note that we would get the same problem formulation $\mathcal{P}_f$ even if we have performed bundling of waves before averaging over time. Formulation $\mathcal{P}_f$ (3.38–3.45) has all the desirable features. It is independent of m, has continuous and fewer variables and constraints than in $\mathcal{P}_d(m)$ and it is also a linear program. It has $O(|I|)$ variables and $O(Mn^2+Ml)$ constraints, where $|I|$ is the total number of feasible Isets, n is the total number of nodes and l is the total number of feasible links in the network.

Recall from Chapter 1 that the partial sums must be computed only among the measurements made at the same time, i.e., each wave of information must be processed separately. Data from one wave cannot be aggregated with that of another wave, i.e., there must not be mixing among different waves. Formulation $\mathcal{P}_f$ is independent of the identities of waves and thus, there is no constraint that explicitly prevent mixing from occuring. This indicates that there is a possibility that the upper bound on throughput computed by it may not be close to the optimal throughput (obtained with the discrete-time model) as it may implicitly violate the constraint of not mixing among the waves.

The intermediate problem formulation $\mathcal{P}_2(m)$ maintains the distinction between different waves. In this model, Constraint (3.23) shares each link among different waves and thus explicitly prevents mixing from occuring. Next, we prove that the throughput computed by $\mathcal{P}_f$ is exactly equal to the one computed by $\mathcal{P}_2(m)$, independent of m. This can be interpreted as either mixing does not occur in $\mathcal{P}_f$ or if it does occur, it does not improve the throughput computed by $\mathcal{P}_f$.

3.6.1 Proof that $\mathcal{P}_f$ and $\mathcal{P}_2(m)$ yield the same throughput

We prove that either no mixing occurs in $\mathcal{P}_f$ or if it does occur, it has no impact on the throughput computed by it, by showing that the throughput computed by $\mathcal{P}_2(m)$ (which explicitly prohibits mixing) is exactly equal to the throughput computed by $\mathcal{P}_f$. The main idea in this proof is to split $\mathcal{P}_2(m)$ into m identical sub-problems, so that each of them is equivalent to $\mathcal{P}_f$. This is accomplished by adding the following constraint to $\mathcal{P}_2(m)$ which aims at decoupling the waves in constraint (3.23).

$$\sum_{p=0}^{M} z_{i,j}^{w,p} \leq \frac{x_{i,j}}{m} \quad \forall s \tag{3.47}$$

Let this new problem be $\mathcal{P}_4(m)$ and since we add a potentially restrictive constraint, we have

$$T_m^2 \leq T_m^4 \tag{3.48}$$

where T_m^4 is the optimal solution of $\mathcal{P}_4(m)$. The constraint (3.23) of $\mathcal{P}_4(m)$ is redundant so that it separates into m identical sub-problems for each w, each with the same optimal solution T_m^4.

The sub-problem is as follows.

$$\mathcal{P}_4^{sub}(m): \qquad \underset{\boldsymbol{\alpha},\mathbf{y},\mathbf{z},\mathbf{u}}{\textbf{Minimize}}\ T_m \tag{3.49}$$

subject to

$$T_m \geq \delta m \tag{3.50}$$

$$\sum_j y_{i,j}^{s,w,0} - \sum_j y_{j,i}^{s,w,0} + u_i^{s,w} = \begin{cases} \dfrac{1}{T_m} & \text{if } i = s \\ 0 & \text{otherwise} \end{cases} \tag{3.51}$$

$$\sum_j y_{i,j}^{s,w,p} - \sum_j y_{j,i}^{s,w,p} - u_i^{s,w} = 0 \quad \forall p = 1 \dots M \tag{3.52}$$

$$\sum_{p=0}^{M} z_{i,j}^{w,p} \leq \frac{x_{i,j}}{m} \tag{3.53}$$

$$\sum_{s=1}^{n} y_{i,j}^{s,w,0} \leq z_{i,j}^{w,0} \tag{3.54}$$

$$y_{i,j}^{s,w,p} \leq z_{i,j}^{w,p} \quad \forall p = 1 \dots M \tag{3.55}$$

$$x_{i,j} \leq \sum_I A_{(i,j),I}\alpha_I \tag{3.56}$$

$$\sum_I \alpha_I \leq 1 \tag{3.57}$$

But, this is precisely the flow model $\mathcal{P}_f$ (3.38–3.45) with the change of variables $y_{i,j}^{s,w,p} \to y_{i,j}^{s,p}/m$ and $T_m \to m/\lambda$. Hence,

$$\frac{m}{\lambda^f} = T_m^4 \tag{3.58}$$

From inequalities (3.48) and (3.58), we conclude

$$T_m^2 \leq \frac{m}{\lambda^f} \tag{3.59}$$

But from inequality (3.46), we know

$$T_m^2 \geq \frac{m}{\lambda^f} \tag{3.60}$$

so that $T_m^2 = \frac{m}{\lambda^f} = T_m^4$. This equivalence implies that the potential implicit mixing of waves in $\mathcal{P}_f$ has no impact on the λ^f computed by it.

Thus, we have rigorously derived the problem formulation $\mathcal{P}_f$ (3.38–3.45) which is the desired flow model for a wireless single rate network. In the next section, we adapt this flow model $\mathcal{P}_f$ to a wired multi-rate network.

3.7 Wired Multi-rate Network

The underlying system of a wired multi-rate network is very different from that of a wireless network as we no longer have time-slots and the transmission of packets is not synchronized among different links which may have different capacities. Thus, a feasible solution for a wired network is fundamentally different from that of a wireless network. Interestingly, regardless of these differences, a simple adaptation of the flow model constructed for the wireless network computes a very tight upper bound on the maximum achievable throughput for wired networks (see Chapter 5). In the following, we present this adapted flow model for wired multi-rate networks.

Consider a directed wired network with n nodes and l links with one of the nodes designated as the *sink*. Let the set of nodes and the set of links be denoted by $\mathcal{N}$ and $\mathcal{L}$ respectively. Assume that the maximum transmission rates supported by each of the links, i.e., their capacities, are also given. The transmission of packets on different links is asynchronous and is uncoordinated. In this network, a subset of nodes are sources, denoted as $\mathcal{S} \subseteq \mathcal{N}$, that are periodically collecting new data and the sink is interested in the first M moments of this data.

Let $c_{i,j}$ be the capacity of the link (i,j). Let $y_{i,j}^{s,0}$ be the amount of information corresponding to the raw data from source s carried on link (i,j) and let $y_{i,j}^{s,p}$ be the amount of information of type p from source s carried on link (i,j). The following is the flow model that we propose to compute the maximum achievable data generation rate in a given wired multi-rate network when the sink is interested in the first M moments.

$$\mathcal{P}_f^{\text{wired}} : \qquad \underset{\mathbf{u},\mathbf{y},\mathbf{z}}{\textbf{Maximize}}\ \lambda \tag{3.61}$$

$$\sum_j y_{i,j}^{s,0} - \sum_j y_{j,i}^{s,0} + u_i^s = \begin{cases} \lambda & \text{if } i = s \in \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{3.62}$$

$$\sum_j y_{i,j}^{s,p} - \sum_j y_{j,i}^{s,p} - u_i^s = 0 \quad \forall p = 1 \dots M \tag{3.63}$$

$$\sum_{p=0}^{M} z_{i,j}^{p} \leq c_{i,j} \tag{3.64}$$

$$\sum_s y_{i,j}^{s,0} \leq z_{i,j}^{0} \tag{3.65}$$

$$y_{i,j}^{s,p} \leq z_{i,j}^{p} \quad \forall p = 1 \dots M \tag{3.66}$$

Constraints (3.62) and (3.63) conserve the *information*. The variable u_i^s models aggregation, i.e., it is the amount of information from the raw data flow from source s that disappears due to aggregation at node i and appears as information in the M higher flows. The variable $z_{i,j}^{p}$ is the fraction of the capacity of link (i,j) that is allocated to the flow of type p. Using z's, constraint (3.64) restricts the total transmission rate supported by the link (i,j) to its capacity $c_{i,j}$ which is shared among the $M+1$ different types of flows. The difference between a raw data flow and a higher type flow is modelled in constraints (3.65) and (3.66) which captures the essential nature of in-network computation and its advantage. A higher type of flow, say $p > 0$ can utilize the entire allocated link capacity for type p ($z^t_{i,j}$) simultaneously with the same type of flows from all the other sources available at node i. But, as the raw data packets from different sources cannot be transmitted at the same time, the raw data flows crossing node i have to share the allocated portion of the link's capacity for raw data ($z^0_{i,j}$) and hence there is a summation over all the sources in (3.65).

The flow model (3.61–3.66) for a wired network is a linear program that is not very different from the flow model for convergecast [10]. It has $O(Mnl + n^2)$ variables and $O(Mn^2 + Ml)$ constraints where l is the total number of links and n is the total number of nodes in the network (in the worst case of every node being a source). Thus, it can be solved in polynomial time [34]. Let λ^*_M be the solution to (3.61–3.66) for a given M. The solution λ^*_M computed using this flow model is an upper bound on the maximum achievable data generation rate in a real network due to two reasons. The first is because we have ignored the packet nature of the flows in this formulation. And the second reason is that there is no constraint ensuring that the aggregation does not happen between different waves. This might lead to higher rates than that is permissible by the network operation which restricts aggregation to happen only between data of the same wave. We cannot introduce a constraint to ensure this without introducing integer variables which would increase the computational complexity of the problem. Thus, there is a need to validate this model, i.e., we have to show that the λ^*_M computed by this flow model is close to the achievable data generation rate in a network that is operated with in-network computation restricting the aggregation between packets of the same wave.

In the next two chapters, we validate the flow models and provide some insights on the value and effects of in-network computation on the network performance for the two types of networks. We treat the wireless and the wired networks separately because the solution strategies for each of them are different.

Chapter 4

Wireless Networks: Validation and Insights

In this chapter, we first validate the flow model for the single rate wireless networks and then provide several engineering insights on the value of in-network computation for these networks.

4.1 Validation

In this section, we validate our flow model formulation for wireless networks, $\mathcal{P}_f$, i.e., we attempt to show that it computes an excellent upper bound to the solution of the discrete-time model $\mathcal{P}_d(m)$. We first prove several propositions related to the maximum achievable throughput of the system. Next, we propose a heuristic that computes a feasible solution to the discrete-time model (and thus a method of operation of the network that utilizes in-network computation) based on the solution to the flow model. As this solution gives a throughput that is very close to the one computed by the flow model for all values of M, we have validated our flow model formulation.

In the following, λ_M represents the maximum achievable throughput with in-network computation when the sink is interested in the first M moments. This relates to the optimal solution of the discrete-time model $\mathcal{P}_d(m)$ as follows.

$$\lambda_M = \lim_{m \to \infty} \frac{m}{T_m} \tag{4.1}$$

As explained in the introduction (Section 1.1), in-network computation leads to data aggregation. This reduces the total volume of data transmitted in the network and potentially increases the achievable throughput. This indicates that using in-network computation could never perform worse than convergecast in terms of the maximum achievable throughput. We prove this in the following proposition.

Proposition 4.1. *If λ_M is the maximum achievable throughput with in-network computation when the sink is interested in the first M moments of the data collected by the sensor nodes and λ_C is the maximum achievable throughput using convergecast, then*

$$\lambda_M \geq \lambda_C \quad \forall M \geq 1 \tag{4.2}$$

Proof: In $\mathcal{P}_d(m)$, if we set $u_i^{s,w} = 0$ (no aggregation) and $q_i^{s,w,p}(k) = y_{i,j}^{s,w,p} = z_{i,j}^{w,p} = 0 \ \forall p = 1 \ldots M$ (only raw data exists), then the problem reduces to the discrete-time model for convergecast. Thus, a solution to convergecast is always a feasible solution to $\mathcal{P}_d(m)$, irrespective of M. Thus, $\lambda_M \geq \lambda_C$. □

Next, we prove that λ_M is a non-decreasing function of M.

Proposition 4.2. *If λ_M is the maximum achievable throughput with in-network computation when the sink is interested in the first M moments of the data collected by the sensor nodes then*

$$\lambda_M \geq \lambda_{M+1} \quad \forall M \geq 1 \tag{4.3}$$

Proof: By ignoring the variables corresponding to the $(M+1)^{th}$ type of data in the optimal solution to $\mathcal{P}_d(m)$ when the sink is interested in $(M+1)$ moments, we get a feasible solution to $\mathcal{P}_d(m)$ when the sink is interested in M moments. Thus, $\lambda_M \geq \lambda_{M+1}$. $\square$

In the next sub-section, we prove that when $M = n$, the throughput computed by $\mathcal{P}_f$, i.e., λ_n^f is equal to the throughput computed by the flow model for convergecast (λ_C) given in [4]. This is interesting because intuitively we do not expect any benefit from in-network computation when $M \geq n$. Thus, our flow model is consistent with our intuition.

4.1.1 The Special Case of $M = n$

Let λ_n^f be the optimal throughput of $\mathcal{P}_f$ when $M = n$ and let λ_C be the optimal convergecast throughput obtained by solving the following convergecast problem $\mathcal{P}_C$ [4]. Recall that there is a flow from every source s to the sink in convergecast.

$$\mathcal{P}_C : \qquad \underset{\boldsymbol{\alpha},\mathbf{y}}{\textbf{Maximize}}\ \lambda \tag{4.4}$$

$$\sum_j y_{i,j}^s - \sum_j y_{j,i}^s = \begin{cases} \lambda & \text{if } i = s \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

$$\sum_{s=1}^{n} y_{i,j}^s \leq \sum_I A_{(i,j),I}\alpha_I \tag{4.6}$$

$$\sum_I \alpha_I \leq 1 \tag{4.7}$$

Since λ_n^f is an upper bound on λ_n and using Proposition 4.1, we have

$$\lambda_n^f \geq \lambda_n \geq \lambda_C \tag{4.8}$$

Next, we perform a series of transformations on $\mathcal{P}_f$ when $M = n$. The idea is to get rid of the dependence on the flow type p and show that when $M = n$, $\mathcal{P}_f$ is equivalent to $\mathcal{P}_C$. From constraint (3.40), we get the value of u_i^s

$$u_i^s = \sum_j y_{i,j}^{s,p} - \sum_j y_{j,i}^{s,p}$$

Replacing u_i^s in constraint (3.39) and rearranging the terms gives

$$\sum_j (y_{i,j}^{s,0} + y_{i,j}^{s,p}) - \sum_j (y_{j,i}^{s,0} + y_{j,i}^{s,p}) = \begin{cases} \lambda & \text{if } i = s \\ 0 & \text{otherwise} \end{cases}$$

We sum on both sides over all p to get

$$\sum_{p=1}^{n} \sum_j (y_{i,j}^{s,0} + y_{i,j}^{s,p}) - \sum_{p=1}^{n} \sum_j (y_{j,i}^{s,0} + y_{j,i}^{s,p}) = \begin{cases} n\lambda & \text{if } i = s \\ 0 & \text{otherwise} \end{cases}$$

Next, we add constraint (3.42) to constraint (3.43) and get

$$\sum_{s=1}^{n} (y_{i,j}^{s,0} + y_{i,j}^{s,p}) \leq z_{i,j}^{0} + n z_{i,j}^{p} \quad \forall p \quad \forall (i,j) \in \mathcal{L}$$

We sum the above inequality over all p to get

$$\sum_{p=1}^{n} \sum_{s=1}^{n} (y_{i,j}^{s,0} + y_{i,j}^{s,p}) \leq n \sum_{p=0}^{n} z_{i,j}^{p}$$

Now, we replace

$$\sum_{p=1}^{n} (y_{i,j}^{s,0} + y_{i,j}^{s,p}) \to n y_{i,j}^{s} \text{ and } \sum_{p=0}^{n} z_{i,j}^{p} \to z_{i,j}$$

With these changes, problem $\mathcal{P}_f$ becomes problem $\mathcal{P}_5$

$$\mathcal{P}_5 : \qquad \underset{\boldsymbol{\alpha},\mathbf{y},\mathbf{z}}{\textbf{Maximize}}\ \lambda \tag{4.9}$$

$$\sum_j y^s_{i,j} - \sum_j y^s_{j,i} = \begin{cases} \lambda & \text{if } i = s \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

$$z_{i,j} \leq x_{i,j} \tag{4.11}$$

$$\sum_{s=1}^{n} y^s_{i,j} \leq z_{i,j} \tag{4.12}$$

$$x_{i,j} \leq \sum_I A_{(i,j),I}\alpha_I \tag{4.13}$$

$$\sum_I \alpha_I \leq 1 \tag{4.14}$$

It is not hard to see the equivalence of this new problem with the flow model for convergecast (4.4–4.7). Thus, if λ_5 is the optimal throughput of this new problem, then we have

$$\lambda_C = \lambda_5 \tag{4.15}$$

Since, a feasible solution to $\mathcal{P}_5$ can be constructed from the optimal solution to $\mathcal{P}_f$ with $M = n$, we have

$$\lambda_5 \geq \lambda^f_n \tag{4.16}$$

From (4.8), (4.15) and (4.16), we conclude that $\lambda^f_n = \lambda_C$, which means that when $M = n$, the throughput computed by $\mathcal{P}_f$ (3.38–3.45) is equal to the throughput computed by the convergecast formulation given in the literature [4].

In the next subsection, we discuss the difficulties in obtaining numerical solutions to $\mathcal{P}_f$ and how we address them.

4.1.2 Numerical Solution to $\mathcal{P}_f$

By design, the flow model $\mathcal{P}_f$ (3.38–3.45) gives an upper bound on the maximum achievable throughput. Recall that it has $O(|I|)$ variables and $O(Mn^2 + Ml)$ constraints, where $|I|$ is the total number of feasible Isets, n is the total number of nodes and l is the total number of feasible links in the network. As the number of possible Isets is exponentially large in the number of links, $\mathcal{P}_f$ is a very large linear program and thus, solving $\mathcal{P}_f$ is not straightforward. The most common technique used in the literature to solve such large programs is *column generation*. We use this technique to solve $\mathcal{P}_f$ to optimality, which involves solving a series of small linear programs [19]. Unlike the column generation technique in [19], we a priori enumerate all the possible Isets (using the efficient algorithm for enumeration proposed in [19]). In the first iteration, we solve $\mathcal{P}_f$ with only the single link Isets as the input. For every iteration after this, the most useful Isets to be added are determined using the dual values of the constraints of the current iteration. We iterate until all the constraints have non-positive dual values which ensures optimality. We have successfully employed this technique for networks with up to 30 nodes. For larger networks, a priori enumeration of Isets is not scalable and we need more sophisticated column generation techniques that generates new Isets as required after every iteration. However, past work on wireless sensor networks suggests that a clustering approach which divides a large network into many small manageable clusters is the most efficient approach for many applications [35], [36]. Thus, the approach we took in this work is useful in most practical scenarios.

In the next subsection, we give an heuristic to construct a feasible solution to the discrete-time model formulation, whose achievable throughput is close to the optimal.

4.1.3 Feasible Solution: Lower Bound

An obvious question now is how close the upper bound computed by $\mathcal{P}_f$ is to the optimal solution of the discrete-time model given in Section 3.4. We answer this question by numerically computing the optimality gap between the upper bound and some lower bound. If the gap is small, we know that the upper bound is good. In this subsection, we show that this is the case by obtaining a feasible solution (based on the solution to the flow model) to the discrete-time model. The achievable throughput of this feasible solution is naturally a lower bound on the maximum achievable throughput that we would compute for $\mathcal{P}_d(m)$.

Typically, a feasible solution to the discrete-time model $\mathcal{P}_d(m)$ is a sequence of Isets (one Iset per time slot generally with some periodic pattern). This solution is also a way to operate the network with in-network computation. In order to compute the throughput given by a feasible solution, it is much simpler to simulate the network operation than to solve $\mathcal{P}_d(m)$. So, we simulate the actual network operation according to the rules of in-network computation given below and the given feasible solution.

1. If the node receives a raw data packet and it already has M packets of the same wave (irrespective of their type) in its buffer, then all the raw data packets get aggregated (effectively disappearing) and M different types of packets (corresponding to the M partial sums described above) are created (some of them may already exist and are replaced by the updated version). Else, no aggregation is performed.

2. If the node receives a packet of type $p > 0$, three cases arise

    (a) The node already has a packet of the same type for the same wave, then the new packet is just added to this packet.

    (b) The node does not have the same type of packet for the same wave but it already has M packets of the same wave, then all raw data packets of this wave disappear and M different types of partial sum packets are created.

    (c) Else no aggregation is performed.

In addition, we also assume that in the simulator, whenever a node transmits, it chooses the packet with the lowest type from the oldest wave in its queue.

In the simulation, we activate the Isets in the sequence given in the feasible solution and update the buffers at all nodes after every time slot according to the rules of in-network computation. From this, we get the number of time slots (Δ_m) required to empty the network of the first m waves (we use $m = 10,000$). We can now compute the throughput of the given feasible solution as m/Δ_m and compare it with the solution λ of $\mathcal{P}_f$.

The question now is how to obtain a good feasible solution. One would naturally guess that we should derive it from the optimal solution to the flow model. However, much information is lost in the transformation from $\mathcal{P}_d(m)$ to $\mathcal{P}_f$ so that converting back the optimal solution of $\mathcal{P}_f$ to a feasible solution of $\mathcal{P}_d(m)$ is difficult. Especially, the routing produced by the flow model is generally too complex to be of any practical use. We observed that if the routing was simple with no loops, the solution to $\mathcal{P}_f$ would readily convert to a feasible solution to $\mathcal{P}_d(m)$ and thus gives a routing and scheduling to operate a network with in-network computation.

The idea thus, is to find a simple routing and solve $\mathcal{P}_f$ assuming this routing. This new solution is also a set of Isets with the fractions of time each of them should be activated for. It is straight-forward to convert this solution to a feasible solution of $\mathcal{P}_d(m)$. We just normalize the fractions of time each Iset is active for and find the number of times each Iset needs to be active. We assume a random order of activation of these Isets and obtain a sequence of Isets that is a feasible solution to the discrete model. In the simulator, we assume that this Iset pattern is repeated until all the waves reach the sink and compute the throughput of this feasible solution.

For validation purposes, we use three different techniques to fix the routing. They are as follows.

1. Impose new constraints on $\mathcal{P}_f$ to force a single path routing or

2. Fix the routing to a min-hop tree or

3. Restrict the routing to a tree on the most active links[1] in the optimal solution to $\mathcal{P}_f$.

Clearly, the throughput achieved with these three routings is lower than the optimal. Luo *et al* [19] show that for convergecast, imposing a single path routing (our first option) achieves close to the optimal throughput [19]. So, we expect the same even when in-network computation is allowed.

For the first method, in order to add constraints that impose a single path routing, we define a new set of binary variables $w_{i,j}$ that indicate whether the link (i,j) is used or not. The new constraints are

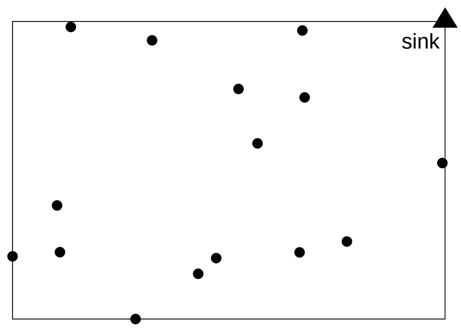
$$y_{i,j}^{s,p} \leq w_{i,j} \quad \forall (i,j) \quad \forall s \quad \forall p$$
$$\sum_j w_{i,j} \leq 1 \quad \forall i, \quad w_{i,j} \in \{0,1\}$$

With these constraints, $\mathcal{P}_f$ has been transformed from a standard LP into a mixed integer program (IP). There are l binary variables and $Mln + n$ constraints involving those binary variables. So, computing an optimal solution becomes extremely time consuming as M or the size of the network increases. In the cases where M becomes too large to solve this problem within a reasonable time, we fix the routing using the other two techniques and obtain a solution to $\mathcal{P}_f$ which we can then convert to a feasible solution of $\mathcal{P}_d(m)$ and compute the throughput using the simulator. For some instances, the min-hop tree gives better throughput than the other tree and for some other instances, it is vice versa.

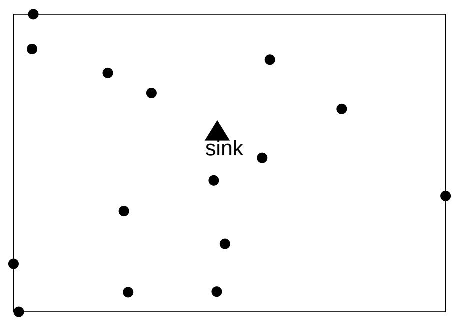
Next, we present numerical results for three randomly generated networks[2], viz., two 16-nodes networks (netA and netB) and a 30-nodes network (netC), given in Figure (4.1). We used the network parameters given in Table 4.1 with the following channel propagation

[1]Activity of a link is defined as the total fraction of time it is active for in the optimal solution to the flow model.

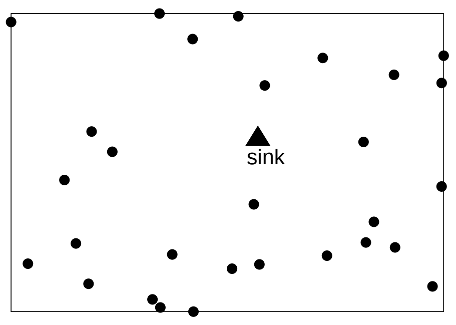
[2]We have calculated these results for 10 random networks. All these results are similar to the ones presented in this document.



(a) First 16 nodes network, netA



(b) Second 16 nodes network, netB



(c) A 30 nodes network, netC

Figure 4.1: Random Networks

model.

$$P_{i,j} = \frac{G_{i,j} P}{(\frac{d_{i,j}}{d_0})^{\eta}}$$

$$SNR = \frac{P_{i,j}}{N_0}$$

where $d_{i,j}$ is the physical distance between nodes i and j, $G_{i,j}$ is the channel gain on link (i, j) that accounts for channel fading and shadowing, d_0 is the near-field cross over distance, η is the path-loss exponent and N_0 is the thermal noise power in the frequency band of operation. Recall that we assume the channel gains to be quasi time-invariant. For simplicity in numerical calculations, we have assumed the same constant G on all the links. However, this should not be construed as a limitation as computations with different G on different links are not more complex.

β	6.4 dB
N_0	-100 dBm
η	3
d_0	0.1 m
G_l	1

Table 4.1: Network parameters used for obtaining numerical results

Network	At the lowest power	At the highest power
netA	2	15
netB	3	15
netC	6	29

Table 4.2: Average node degree in netA, netB and netC

The average node degrees of the three networks at the lowest and the highest powers are given in Table 4.2. For the two 16-nodes networks, we were able to optimally solve the single-path problem for $M < 8$ but for higher M, we have fixed the routing using the two methods described before. For the 30-nodes network, CPLEX was unable to solve the

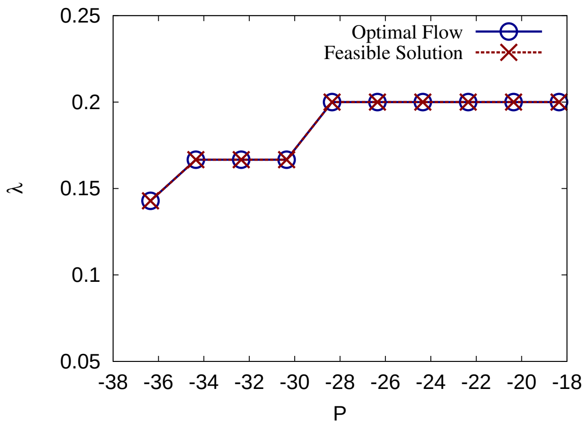


Figure 4.2: λ vs P for the 16-nodes network, netA when $M = 1$

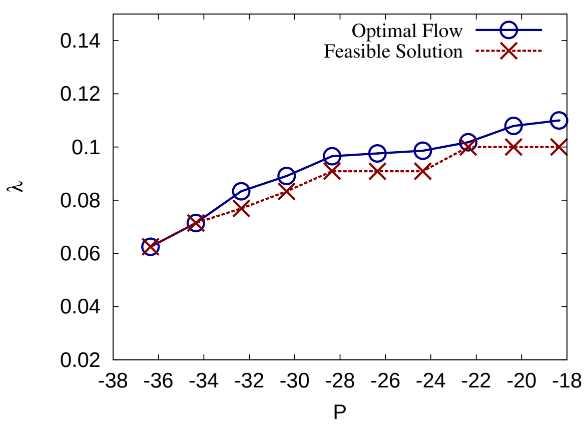


Figure 4.3: λ vs P for the 16-nodes network, netA when $M = 3$

optimal single-path problem for any M as it is a very large binary program. So, we use the other two methods to fix the routing and obtain the feasible solutions.

Plots of the maximum achievable throughput λ versus the transmit power P (used by all the nodes) for some selected M for the three networks (netA, netB and netC) are given in Figures (4.2 – 4.9). All these plots except Figure (4.8) show that the upper bound (labelled optimal flow) is close to the lower bound (labelled feasible solution which is the best of the three heuristics). This validates the flow model $\mathcal{P}_f$.

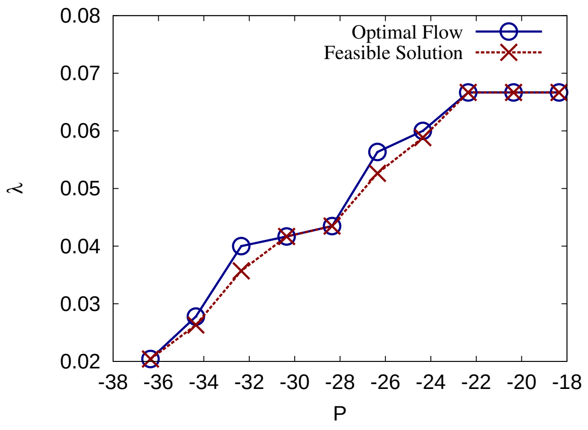


Figure 4.4: λ vs P for the 16-nodes network, netA when $M = 15$

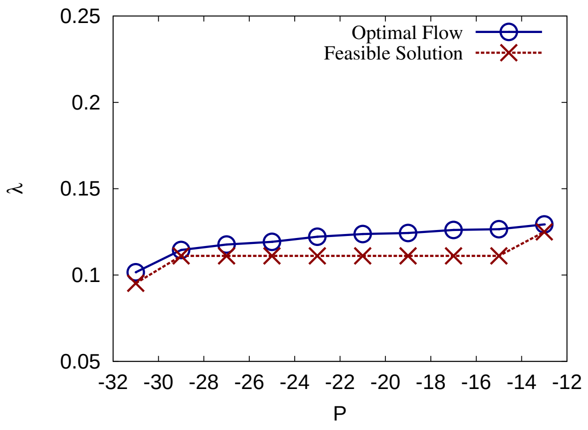


Figure 4.5: λ vs P for the 16-nodes network, netB when $M = 2$

However, note that the feasible solutions in Figure (4.8) for the 30-nodes network when $M = 1$ are not as close to the upper bound as those for the 16-nodes network in Figures (4.2). The reason for this is that we were unable to optimally solve the large binary program (that imposes single path routing) for the 30-nodes network that gives the best feasible solutions for the 16-nodes network for small M.

In the next section, we give several engineering insights on the value and effects of in-network computation in single-rate wireless networks.

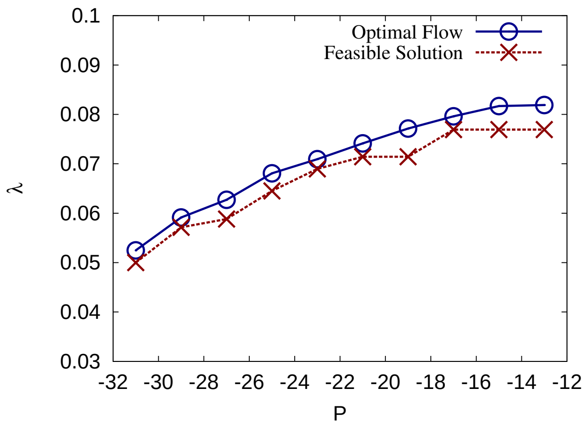


Figure 4.6: λ vs P for the 16-nodes network, netB when $M = 5$

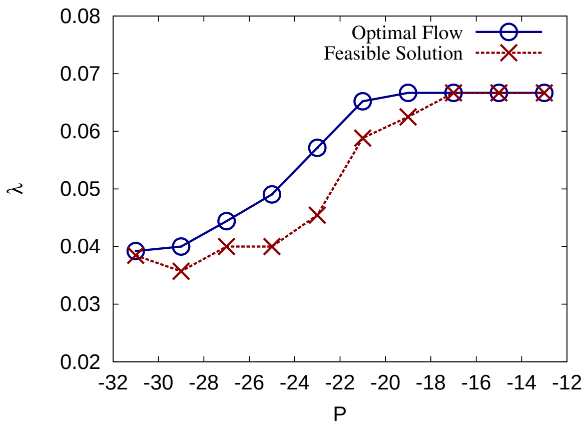


Figure 4.7: λ vs P for the 16-nodes network, netB when $M = 10$

4.2 Insights

As mentioned in the introduction, in-network computation is expected to potentially improve the throughput significantly compared to convergecast. In this work, we have attempted to quantify this improvement and to the best of our knowledge, this is the first such attempt in the literature. For a given M, we define γ_M as the relative increase in throughput due to in-network computation, with respect to that of convergecast. Mathematically, we have

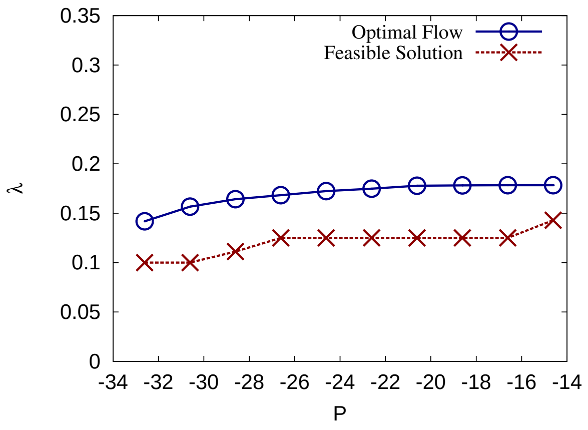


Figure 4.8: λ vs P for the 30-nodes network, netC when $M = 1$

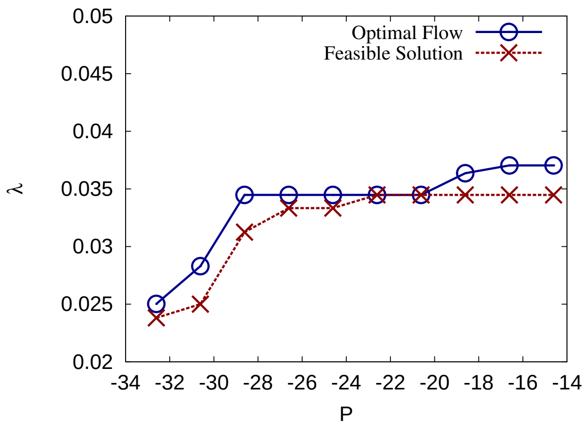


Figure 4.9: λ vs P for the 30-nodes network, netC when $M = 12$

$$\gamma_M = \frac{\lambda_M - \lambda_C}{\lambda_C} \tag{4.17}$$

In Figures (4.10–4.11), we have plotted γ_M with respect to M for different transmit powers. We observe from these plots that for low M, in-network computation does lead to significant gains in throughput compared to convergecast at all powers. For $M = 1$, the throughput using in-network computation is several times larger than that is possible using convergecast and even for relatively large M like $M = 5$, the gains in throughput

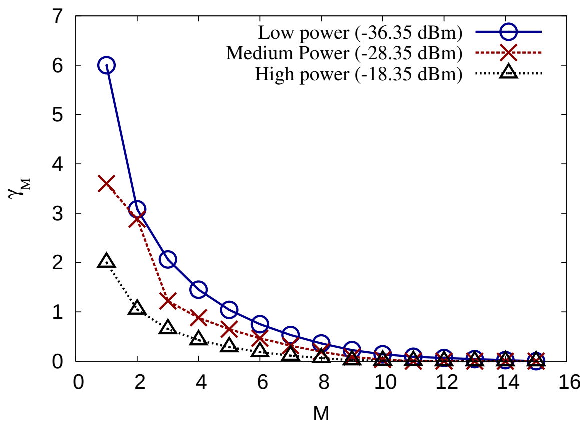


Figure 4.10: γ (gain) vs M for netA at three different powers

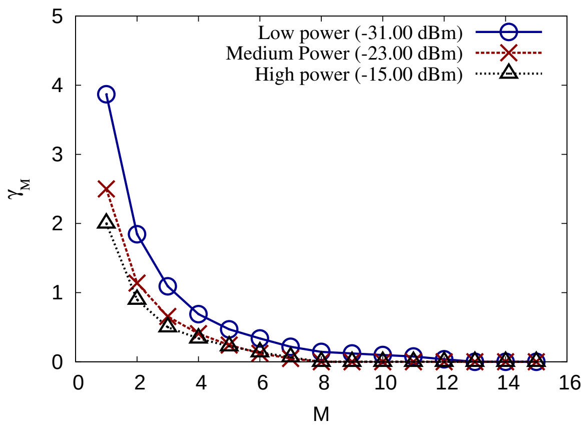


Figure 4.11: γ (gain) vs M for netB at three different powers

are surprisingly still significant. We also note that these gains are much larger at lower powers than at higher powers which implies that λ_M increases slower than λ_C when the transmission power increases. This is because the increased spatial reuse resulting from higher transmission power is more useful for convergecast that transports larger volume of data compared to when in-network computation is allowed.

The high gains in throughput at low M can be utilized in two different ways. Using in-network computation, we could either use the sensor network to monitor at a very high

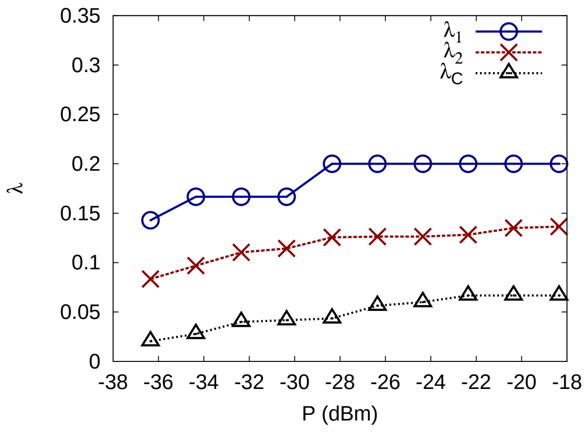


Figure 4.12: Comparison of throughputs when $M = 1$, $M = 2$ and convergecast for netA

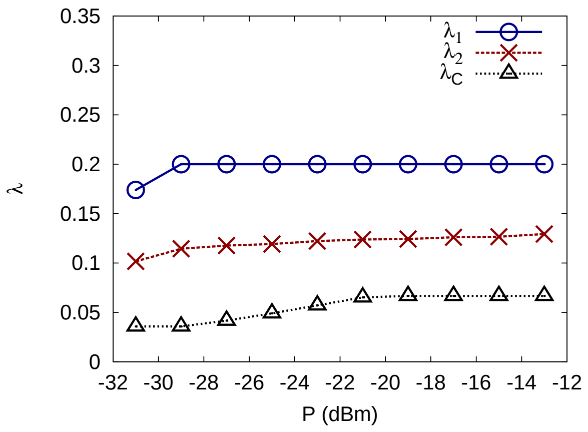


Figure 4.13: Comparison of throughputs when $M = 1$, $M = 2$ and convergecast for netB

frequency or we could put the nodes into a sleeping mode more often saving energy and increasing the lifetime of the network. In Figures (4.12–4.13), we compare the throughputs when $M = 1$, $M = 2$ and convergecast with respect to power. We observe that irrespective of the power, there is an almost constant gain in throughput due to in-network computation when $M = 1$ and when $M = 2$.

Another insight for wireless networks comes from Figures (4.2–4.7). Because of the way we have constructed the feasible solution, these results show that we can get near-optimal throughput using a single-path routing. This is important since single path routing is easy to implement and our results confirm that this is not a bad idea. These results also show that at higher M, a simple routing based on min-hop tree is sufficient to achieve a near optimal throughput.

In the next chapter, we consider the validation of the flow model adapted for the multi-rate wired networks.

Chapter 5

Wired Networks: Validation and Insights

In this chapter, we validate the flow model for the multi-rate wired networks and provide several engineering insights on the value of in-network computation for these networks.

5.1 Validation

In this section, we consider the validation of the flow model (3.61–3.66) for the wired multi-rate networks (given in Section 3.7). Recall that the underlying system of a wired multi-rate network is very different from that of a wireless single rate network considered in the last chapter. In the wired network, there are no time-slots and the transmission of packets is not synchronized among different links which may have different capacities. Also, unlike in a wireless network, all the links can be active at the same time in a wired network. Owing to these fundamental differences, the validation technique used in the last chapter does not work unaltered for the wired networks. In the following, by *heuristic strategy* we mean the routing, scheduling and queue management that is required to operate a network with in-network computation.

The basic methodology of validation is still the same in essence as in the last chapter. The key difference is that we did not define a discrete-time model for the wired networks. However, it has no impact on the validation method as we simulate the network to compute the throughput of a given heuristic strategy, i.e., a feasible solution of the operation of the network. In the next subsection, we present a heuristic strategy that could be used to operate the wired network that allows in-network computation of the first M moments. We compare the throughput of this heuristic strategy with the upper bound computed by the flow model and if they are close, we have validated the flow model. We show that both throughputs are indeed close, validating the flow model.

5.1.1 Heuristic Strategy

Recall that λ_M^* is the upper bound computed by the flow model on the maximum achievable throughput for a wired multi-rate network when the sink is interested in the first M moments. In this subsection, we propose a heuristic strategy that could be implemented in a wired network and use the rules of in-network computation given in Section 4.1.3. It is based on the idea that to enable the largest number of opportunities for aggregation, some delay needs to be imposed at intermediate nodes. However, note that this does not impact the end-to-end delay to compute the function for a given wave since the sink cannot compute this function before it has received all the data for that wave. We show that this heuristic strategy yields a data generation rate close to λ_M^* for two sets of 200 random wired networks (one set with links of unit capacity and the other with links of random capacity). This validates our flow model. This strategy is also useful in itself as a heuristic to operate a network and achieve a near optimal data generation rate for a given M.

Any strategy for operating a network has three components, viz., scheduling, routing and queue management. In a wired network, scheduling seems meaningless as any link can be actively transmitting all the time as long as there are packets to be sent on that link. However, when in-network computation is allowed, it could be beneficial to wait sometimes to take advantage of aggregation. Thus, scheduling could be understood as when to stop the transmissions and wait to receive data from other nodes so as to aggregate.

Now, the next question is how to arrive at such a strategy. One attractive way would be to derive some information about that strategy from the solution to the flow model. Unfortunately, the numerical solution does not give much useful information beyond the value of λ_M^* which we know is an upper bound on the maximum achievable data generation rate for the given M. There are typically multiple solutions that result in the same optimal λ_M^* and in general, the solution we get for the flow model by using a commercial solver like CPLEX is not amenable for deriving an implementable strategy from it.

We illustrate this statement in the following example. Consider solving the flow model with $M = 1$ on the 4×4 (directed) grid network, given in Figure 5.1. Every link in this network represents two directed links of unit capacity in either direction. This is a very simple case and it can easily be seen that the optimal throughput for this network is 2 which can be achieved by operating two independent trees (i.e., the two trees do not share any links) such that each tree connects all the nodes and uses one of the links $(1, sink)$ or $(2, sink)$ as its final link to the sink (see Figure 5.2 for the trees and we explain later in this section in more detail how to operate a tree and multiple trees). However, the only useful information from the solution obtained using CPLEX to this flow model is that the value of λ_1^* is 2 and the numerical solution, i.e., the optimal values of the variables $\mathbf{y}, \mathbf{u}$ and $\mathbf{z}$ for which this λ_1^* is computed, does not reflect these trees and is quite complicated. We have tried to add constraints to the flow model to gear its solution towards something simpler that can be interpreted but we were unsuccessful even for this simple network. Thus, there is a need to develop a heuristic strategy that is both implementable in a network and yields a good throughput. We believe that strategies such as these which are implementable in

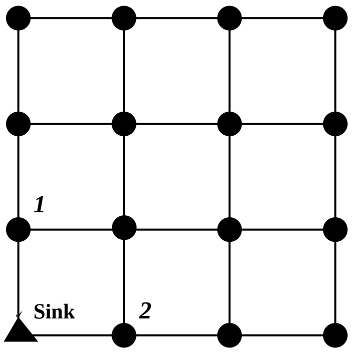


Figure 5.1: A grid network: Every link represents two directed links, one in either direction

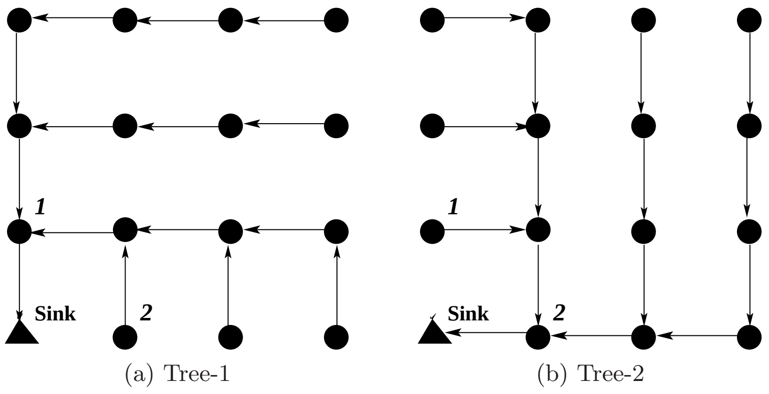


Figure 5.2: Two independent trees of the grid network in Figure 5.1

a packet-based network have not received much attention in the literature. Our proposed heuristic strategy is the first of its kind that has all the components necessary to implement in-network computation in a wired network.

The heuristic strategy presented in this chapter is based on a set of observations and a set of assumptions. The first fact it depends upon is that we know how to operate a network with in-network computation and achieve the maximum possible throughput if the network is a single path tree rooted towards the sink (see Section 5.1.2 for details). Another observation is that in a wired network, multi-path routing is necessary to provide high throughput. For example, any single path routing is sub-optimal for the wired network in Figure 5.3 for both convergecast and in-network computation and it is sub-optimal for

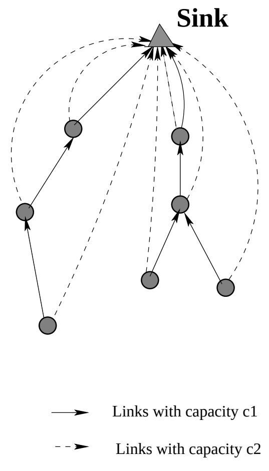


Figure 5.3: An example network

the wired grid network in Figure 5.1 when in-network computation is allowed. A routing with two trees achieves significantly higher data generation rates than the one with just one of the trees. Thus, for a general wired network, we have to consider multi-path routing for higher performance.

A difficulty when multi-path routing is used is that it might result in cycles like in the network in Figure 5.4 which has two trees, viz., one with solid links and the other with dotted links. The combination of these two trees results in a cycle ABCA and any packet being stuck in this cycle is undesirable. It is known that to avoid loops in convergecast, a source-based path routing (as opposed to a hop by hop routing) needs to be used in which the sources arbitrarily map the packets they generate to one of the paths originating from them towards the sink. The proportion of packets mapped to a path depends on the fraction of the total data rate the path is expected to carry.

Thus, the main idea of our heuristic strategy comes from the following facts.

1. There is a need for multi-path routing.

2. We know how to develop an optimal strategy to achieve the maximum achievable data generation rate in a tree network. This is an important building block towards

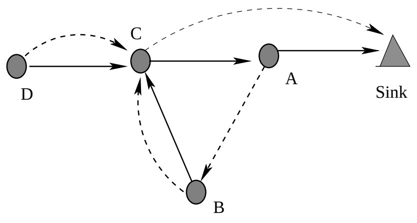


Figure 5.4: Multi-path routing

developing a strategy for a general network. This *single tree strategy* is presented in sub-section 5.1.2.

Hence, our heuristic strategy generates multiple trees, computes the maximum data generation rate supported by each of the trees and operates them simultaneously so that the network supports a data generation rate which is the sum of the data generation rates of the individual trees. In sub-section 5.1.3, we propose a *multiple tree generating algorithm* based on depth-first search algorithm. In this algorithm, we do not impose any condition to generate only independent trees, i.e., we allow the trees to share links if the capacities of the links allow it.

When in-network computation is allowed and multiple trees are used, we note that to enable aggregation, we have to facilitate the "meetings" in time and space of the packets of the same wave. We try to enforce this by assigning all packets (irrespective of their type) from a given wave to the same tree. This assignment has to be done locally at each source node but in a way that ensures that all nodes map the same wave to the same tree. This also prevents the packets from entering a cycle that could result from operating multiple trees together. We propose a distributed *"wave to tree assignment" algorithm* to perform this in sub-section 5.1.4.

Finally, we need to extend the queue management and the aggregation strategy developed for a single tree to the case of multiple trees that might not be independent.

In summary, our strategy is based on the following three components.

1. An optimal strategy for a given tree which is the main building block for our heuristic strategy for a general network;
2. An algorithm to generate multiple trees;
3. A queue management scheme that is an extension of the optimal queue management scheme for a single tree. This extension also ensures that packets of the same wave are assigned to the same tree using a distributed wave to tree assignment algorithm.

We address each of these components in the next three sub-sections. We begin with presenting the optimal strategy for a tree network.

5.1.2 Strategy for a Single Tree

Consider a network with a tree topology directed towards the sink, i.e., there is exactly one path from every node to the sink (e.g., the network with just solid links in Figure 5.3). Without any loss of generality, assume that some of the nodes (except the sink) are sources and let the data generation rate of each source be $\lambda(M)$, assuming that the sink is interested in the first M moments. For this network, there is only one choice of routing. Let the capacity of a link (i, j) in this network be $c_{i,j}$. As the link (i, j) is the only outgoing link from node i, if G_i is the number of sources in the children of node i (including i), then the link (i, j) carries the data from all these G_i sources. Since in-network computation is possible, the expected traffic on link (i, j) is between $(M\lambda(M), G_i\lambda(M))$ if $G_i \geq M$, else it is between $(G_i\lambda(M), M\lambda(M))$. So, we have either $c_{i,j} \geq G_i\lambda(M)$ or $c_{i,j} \geq M\lambda(M)$ which implies that

$$\lambda(M) \leq \max\{\frac{c_{i,j}}{M}, \frac{c_{i,j}}{G_i}\} \tag{5.1}$$

The bottleneck link is the one that has the least $\lambda(M)$ computed using equation (5.1). Thus, we have

$$\lambda(M) = \min_{(i,j)}\{\max\{\frac{c_{i,j}}{M}, \frac{c_{i,j}}{G_i}\}\} \tag{5.2}$$

If a strategy on the tree network supports the $\lambda(M)$ computed by (5.2), then it is optimal as it an upper bound because of the bottleneck. Hence, the expected amount of traffic to be carried on a link (i, j) is $\min\{M\lambda(M), G_i\lambda(M)\}$. In order to ensure that the traffic carried is indeed the expected amount, we propose the following queue management strategy. First, the nodes need to determine if the minimum traffic on their outgoing link is $M\lambda(M)$ or $G_i\lambda(M)$. A simple centralized or a distributed algorithm can be used to accomplish this task.

If the minimum expected traffic is $M\lambda(M)$ on the outgoing link of a node, then it waits for the data from all the sources in its children. In particular, the node maintains two buffers, one for in-network computation which we call the *nodal buffer* and the other for the outgoing link (in the case of a tree, each node has exactly one outgoing link), which we call the *output queue*. The packets move from the nodal buffer to the output queue when they are ready to be transmitted. All the incoming and the generated packets first enter the nodal buffer. The node performs aggregation on the packets in this buffer using the rules given in Section 4.1.3. After the node receives all the expected packets from its immediate children for a given wave, it sends the aggregated packets of that wave to the output queue. In the output queue, the packets are ordered from oldest to the newest wave and packets of the oldest wave are transmitted first.

On the other hand, if the minimum expected traffic is $G_i\lambda(M)$, then the node maintains just the output queue and transmits the packets in the order of the wave number without the need to wait. This queue management strategy ensures that the least possible amount of traffic is carried on every link. Although, it seems that we are delaying some of the data from reaching the sink quicker, it should be noted that the sink can compute the function of the data in a wave only after it receives all the data of the wave. Thus, from that point of view, there is no additional delay. However, it should be noted that this is just one of the many possible heuristics that achieve the optimal throughput for a tree network.

In the next sub-section, we present our algorithm for generating multiple trees from a given network.

5.1.3 Generation of Multiple Trees

We address the task of generating multiple trees by proposing Algorithm 1. It generates trees by performing the following three steps in a loop until the network is disconnected, i.e., until there is at least one source for which there is no path to the sink.

1. Extract a tree directed towards the sink from the network.
2. Compute the $\lambda(M)$ supported by it using equation (5.2).
3. Update the capacities of the links in the network by subtracting the capacity needed by the links in the tree to support $\lambda(M)$. We remove the links whose capacity after update becomes 0.

Since, we are reducing the capacity of the links of the network in every cycle of the while loop, the network would eventually become disconnected terminating the algorithm.

Algorithm 1 Heuristic to generate multiple routings

```
Input: A connected graph, $\mathcal{G}$ with link capacities
Output: A set of routings, $\mathcal{R}$
1: $\mathcal{G}' \leftarrow \mathcal{G}$
2: while $\mathcal{G}'$ is connected do
3:     $T \leftarrow$ DFS ($\mathcal{G}'$) // Extract a tree $T$ from $\mathcal{G}'$
4:     $\mathcal{R} \leftarrow \mathcal{R} \cup T$
5:     Compute $\lambda$ for tree $T$ using equation 5.2
6:     $\forall (i,j) \in T, c_{i,j}(\mathcal{G}') = c_{i,j}(\mathcal{G}') - \min\{M\lambda, (G_i)\lambda\}$ // We perform $\mathcal{G}' \leftarrow \mathcal{G}' - T$ in this
       step
7: end while
```

For the task of generating a tree (the step (3) in Algorithm 1), we use an adaptation of the well-known depth-first search DFS($\mathcal{G}'$) algorithm [37] to extract a tree T from the network $\mathcal{G}'$. DFS($\mathcal{G}'$) [1] is given in Algorithm 2. It uses Algorithm 3. The key difference between our depth-first search and the classic depth-first search (given in [37]) is that our algorithm tries to find up to M nodes at the same depth before it increases the depth of exploration. This adaptation was done so as to increase the opportunities for aggregation as much as possible in the generated trees.

Algorithm 2 Extract a tree using depth-first search: DFS($\mathcal{G}'$)

```
Input: A connected graph, $\mathcal{G}'$
Output: A tree $T$ // i.e., parent[$i$]   $\forall i \in$ Nodes($\mathcal{G}$)$'$
1: $\forall i \in \text{Nodes}(\mathcal{G}'), \text{color}[i] \leftarrow$ WHITE
2: $\forall i \in \text{Nodes}(\mathcal{G}'), \text{parent}[i] \leftarrow$ NIL
3: DFS_VISIT($sink$)
```

Let $\lambda^h_{_M}$ be the sum of the data generation rates supported by all the trees generated by Algorithm 1. This value is a measure of the performance of the algorithm. The higher it is, the better is the performance of the task of generation of multiple trees. If $\lambda^*_{_M}$ is the upper bound computed by the flow model on the maximum achievable data generation rate, c_{min} is the capacity of the link with minimum capacity and C is the minimum of the min-cuts as defined in Chapter 3, then for any algorithm that generates trees (in any order), we have the following bound on its performance because the algorithm would generate at least one tree which in the worst case has the link with c_{min} capacity as the bottleneck.

$$\frac{\lambda^h_{_M}}{\lambda^*_{_M}} \geq \frac{c_{min}}{C} \qquad (5.3)$$

Later, in Section 5.2 we show that our algorithm performs very well on many random networks.

[1]A bread-first search (BFS) approach was also tested but its performance was inferior in comparison to DFS

Algorithm 3 Recursive subroutine for DFS: DFS_VISIT(i)

Input: A connected graph, $\mathcal{G}'$, node i and M
Output: Updates parent vector

```
1: color[$i$] ← BLACK // Node $i$ is explored
2: $m \leftarrow 0$
3: if $i$ is not *sink* then
4:    for each vertex $j$ such that $i \in \text{Adj}(j)$ do
5:       if color[$j$] = WHITE then
6:          parent[$j$] ← $i$
7:          color[$j$] ← GRAY
8:          $m \leftarrow m + 1$
9:          if $m = M$ then
10:            DFS_VISIT($j$)
11:         end if
12:      end if
13:   end for
14:   if $m \neq 0$ then
15:      for each vertex $j$ such that $i \in \text{Adj}(j)$ do
16:         if parent[$j$] = $i$ and color[$j$] = GRAY then
17:            DFS_VISIT($j$)
18:         end if
19:      end for
20:   end if
21: else
22:   for each vertex $j$ such that $i \in \text{Adj}(j)$ do
23:      if color[$j$] = WHITE then
24:         parent[$j$] ← $i$
25:         color[$j$] ← GRAY
26:         DFS_VISIT($j$)
27:      end if
28:   end for
29: end if
```

In the next sub-section, we present a queue management strategy for the operation of multiple trees. This is based on the optimal strategy we have presented for a single tree network.

5.1.4 Queue Management Strategy for Multiple Trees

Assume that the heuristic in Algorithm 1 has generated k trees with $\lambda_M^1, \lambda_M^2 \dots \lambda_M^k$ as the respective data generation rates supported by each of them individually. We assume that each node uses the same tree numbering and knows the corresponding λ_M^i. Then, the aim of our queue management strategy is to ensure that the network supports a total rate of λ_M^h where

$$\lambda_M^h = \sum_{i=1}^{k} \lambda_M^i. \tag{5.4}$$

Every source is generating new packets at rate λ_M^h. Every newly generated raw data packet is assigned a new wave number and is saved in the nodal buffer of the source. All the raw data packets carry the wave number in their header. All packets of type $p > 0$ also carry the same wave number as the raw data they are created from. For enabling as many opportunities for aggregation as possible, we have to ensure that all the nodes map packets of the same wave to the same tree. For this task, we propose Algorithm 4 that can be run by any node to determine the assignment of the waves to the trees. It is simply a weighted round robin that allocates a wave to a tree based on the weights $\frac{\lambda_M^i}{\lambda_M^h}$. All nodes use the same `MAX_WAVE` $= \lceil \frac{1}{\lambda_{min}} \rceil$ in Algorithm 4, where λ_{min} is the rate of the tree with the minimum rate. The assignment generated by Algorithm 4 would be repeated cyclically every `MAX_WAVE` number of waves. Since all the nodes run the algorithm with the same inputs (λ_M^i's), a given wave is mapped to the same tree at all the nodes. The mapping for a wave w to a tree i is saved at a node until all the packets of wave w that are supposed to be forwarded by the node have been transmitted.

Algorithm 4 Assignment Algorithm: waves to trees

```
Input: $\lambda_M^1, \lambda_M^2, \ldots, \lambda_M^k$
1: $w \leftarrow 0$
2: for $i = 1$ to $k$ do
3:     $t_i \leftarrow 0$
4:     $t_i^p \leftarrow 0$
5: end for
6: while w $\leq$ MAX_WAVE do
7:     for $i = 1$ to $k$ do
8:         $t_i \leftarrow t_i^p + \frac{1}{\lambda_M^i}$
9:     end for
10:    $t_{min} \leftarrow \min_i t_i$ and $j \leftarrow \{i : t_i = t_{min}\}$
11:    $w \leftarrow w + 1$
12:    Map wave $w$ to Tree $j$
13:    $t_j^p \leftarrow t_{min}$
14: end while
```

Next, we propose a queue management strategy when the k generated trees are operated simultaneously. We assume that every node knows its outgoing link for every tree. As was the case for the single tree, every node first determines for every tree if it is an aggregator or just a forwarder using either a centralized or a simple distributed algorithm. If a node is an aggregator in a tree, it also determines the total number of packets it expects to receive from all of its immediate children in that tree. A node could be an aggregator in some trees and just a forwarder in others. As was the case in the single tree strategy, the aggregator of any tree waits for its immediate children in that tree to send all the data of the wave before it performs in-network computation and forwards the aggregated data to the output buffer of the outgoing link in that tree.

If an outgoing link is shared among multiple trees then its output queue receives packets corresponding to all those trees from the nodal buffer and the packets of the oldest wave (irrespective of the tree number) gets priority in transmission. This strategy ensures that a link is never used beyond its capacity, even when it is shared among many trees.

We have implemented a discrete event simulator in C++ that imitates the network operation to check that the λ_M^h, our heuristic strategy claims to support in a network is indeed feasible. This simulator was used not only as an additional check that our heuristic strategy works but also to check if any other simpler queue management strategy (one that does not force the aggregator nodes to wait before transmitting data for a given wave) could have worked. We have seen that there are networks for which a queue management without the waiting in the nodal buffer did not work.

In the next section, we compare the throughput achieved by this heuristic strategy against the upper bound computed by the flow model.

5.2 Numerical Results

In this subsection, we present numerical results on two sets of 200 instances of 30-node random networks (one set with links of unit capacity and the other with links of random capacity) that show that the upper bound computed by the flow model is close to the practical achievable data generation rate using the heuristic strategy for different values of M. Thus, cross-validating the flow model and the proposed heuristic strategy. In all the networks considered, we assume that all the nodes other than the sink are sources.

We generate a random network by letting a link (i, j) exist with probability p and we have used values of p from 0.04 to 0.5. We have ordered the network ids in terms of the number of links in the network (the lower the network id, the lower the number of links). The first set of 200 random networks have unit capacity on all their links while the second set have links with random capacity between 1 and 2. Note that for the 200 networks in a set, multiple networks might have the same number of links. Table 5.1 gives a rough mapping of the number of links to the network ids. Note that the connectivity increases when the network id increases.

We can observe that when $M = 1$, the data generation rate supported by the heuristic strategy (labeled λ_1^h) is almost always the same as the upper bound (labeled λ_1^*) computed by the flow model (see Figure 5.5). When $M = 2$, the comparison is given in Figure 5.6. The average difference between the two throughputs is 15.58% for the set of networks with unit capacity links and it is 14.76% for the set of networks with random capacity. As it takes longer to compute the upper bound using the flow model for $M = 3$, we have computed λ_3^* for every fourth network in each set. The comparison for these 50 networks in each set when $M = 3$ is shown in Figure 5.7. The average difference between the two throughputs is 9.13% for the networks with unit capacity links and it is 11.6% for the set of networks with random capacity links. Thus, the heuristic strategy achieves data generation rates close to the upper bound. The implications of this is two-fold. First, it validates the flow model, i.e., it shows that even though the flow model does not take waves into account and ignores the discrete nature of the packets, it gives a tight upper bound on the maximum achievable data generation rate. Second, it shows that the heuristic strategy is close to the optimal.

Network ID	Number of links
1 to 50	30 to 125
51 to 100	126 to 230
101 to 150	231 to 350
151 to 200	351 to 450

Table 5.1: The range of the number of links in the networks

In the next section, we provide engineering insights for wired networks when in-network computation is enabled.

5.3 Insights

In Figure 5.8, we have plotted the throughputs computed using the flow model when $M = 1$, $M = 2$ and for convergecast on the two sets of 200 instances of 30-node random wired networks used for validation. The throughput of convergecast is very low when compared to the throughput when in network computation is enabled and hence for ease of illustration, we have magnified its throughput by a factor of ten in the plots in Figure 5.8. We see from these plots that in-network computation results in significant improvements in network performance when compared to convergecast. The throughput possible when $M = 1$ or $M = 2$ is almost always at least 10 times higher than that possible with convergecast, especially at higher connectivity. Recall that the number of links and thus the connectivity of the networks increases with ID.

In Figure 5.8, we also observe that the increase in throughput when connectivity is very high, can be very significant when in-network computation is allowed compared to when convergecast is used. Although, connectivity helps in increasing the throughput of convergecast, the increase is not as significant as it is when in-network computation is allowed.

In the same plot, we note that $\lambda_1^* \geq \lambda_2^*$ and observe that at low connectivity (for network ID < 50), the throughput possible when $M = 1$ is twice that possible when $M = 2$. But as the connectivity increases, the throughputs of these are not very different and the average difference between them is only 19.30% for networks with unit capacity links and 19.126% for networks with random capacity links.

From the operation point of view, we have noted that queue management is important if the routing is based on multiple trees. If we do not force the aggregation to happen at the nodes where it is expected by making them delay their transmissions, it would result in unstable queues. This was not an issue in wireless networks as the routing comprised of just a single tree.

We conclude this chapter with bounds on λ_M^* using the max-flow min-cut theorem.

5.3.1 Bounds on λ_M^*

Convergecast in wired networks is an instance of a multi-commodity maximum flow problem with an additional constraint that the output flow from all the sources is equal. The multi-commodity maximum flow problem is well studied in the literature [38]. In this subsection, we use the solution to a single commodity maximum flow problem and prove tight bounds on λ_M^*.

Let C_s be the solution to the single commodity maximum flow problem for the source s and the sink in the given network. From the max-flow min-cut theorem, C_s is also the value of the minimum cut between the source s and the sink. Let $C \triangleq \textbf{Minimium}_s\ C_s$. Then, we have

$$\lambda_1^* \leq C \tag{5.5}$$

$$\frac{\lambda_1^*}{M} \leq \lambda_M^* \leq C \tag{5.6}$$

Inequality (5.5) is true because if all sources have to maintain equal data generation rates and the links have to operate within their capacity then no source can send more than the rate allowed by the minimum of all the min-cuts. The upper-bound in the inequality (5.6) also follows from this argument. The lower bound in (5.6) is true because given a solution achieving λ_1^*, we can achieve λ_1^*/M when the sink is interested in the first M moments if we simply replace every source with M sources (at the price of over aggregation) and scale down the $c_{i,j}$'s of the links by M. With this replacement, we have M identical problems with a scaled down network capacity and $M = 1$ for the data aggregation.

We can further show that the upper bound in inequality (5.5) is tight by constructing a feasible solution to (3.61–3.66) when $M = 1$. The feasible solution is as follows. Let $u_s^s = \lambda$, $y_{i,j}^{s,0} = 0$ and $z_{i,j}^0 = 0$. This means that the data is converted to type 1 at the sources itself. Since $M = 1$, it simply replaces the type 0 data with type 1 and does not create any redundant additional data to transmit. With this, the problem separates into identical sub-problems, one for every source s. Each of these sub-problems is a single commodity maximum flow problem, each with a solution of C_s for a given source, s. Thus, we have a feasible solution with a rate of C. Thus, we have

$$\lambda_1^* = C \tag{5.7}$$

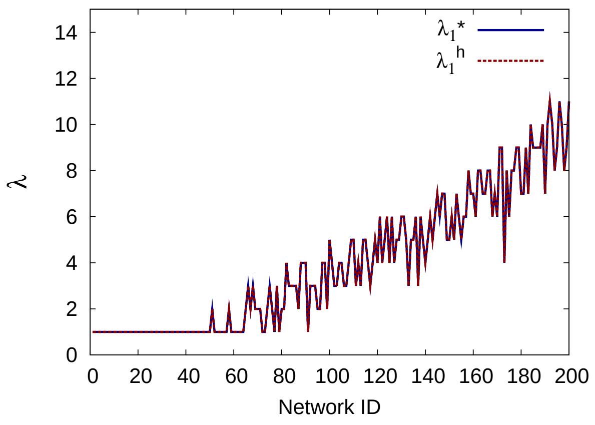
$$\frac{C}{M} \leq \lambda_M^* \leq C \tag{5.8}$$

An example that shows that λ_M^* could be strictly greater than $\frac{C}{M}$ is as follows. Consider the network given in Figure (5.3). It can be decomposed into a tree rooted at the sink with every link having a capacity c_1 and a star network with each link of capacity c_2. For this network, irrespective of M, the star with dotted links always achieves a data generation rate of c_2 (no need for aggregation) while the tree achieves a minimum data generation rate of $\frac{c_1}{M}$. Thus, for this network, we have $C = c_1 + c_2$ but the lower bound is

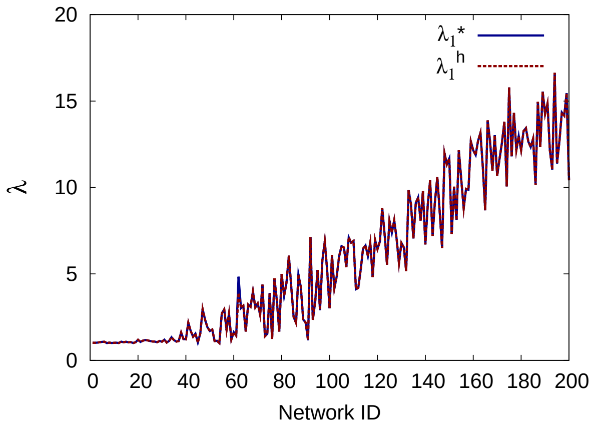
$$\lambda_M^* \geq c_2 + \frac{c_1}{M} > \frac{c_1 + c_2}{M} = \frac{C}{M} \tag{5.9}$$

We conclude this chapter with two interesting facts. The first one is that there is one network, the star network (all nodes are directly connected to the sink), for which in-network computation does not help. The throughput is equal to the capacity of the link with minimum capacity and it is the same as that of convergecast for all M. The second fact is on the networks with unit capacity links. For these networks when $M = 1$, any tree achieves a data generation rate of exactly 1. And thus, λ_1^* is also equal to the number of trees needed to achieve it. This shows that a single path routing could be λ_1^* times worse than a multiple path routing. Thus, for wired networks, whenever the sink is interested in a statistical function of the data being collected, in-network computation should be used with multiple tree routing.

We conclude this thesis in the next chapter with several directions of possible future work.

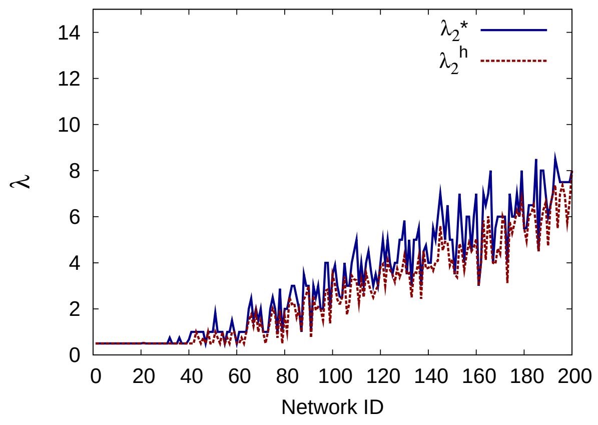


(a) With unit capacity links

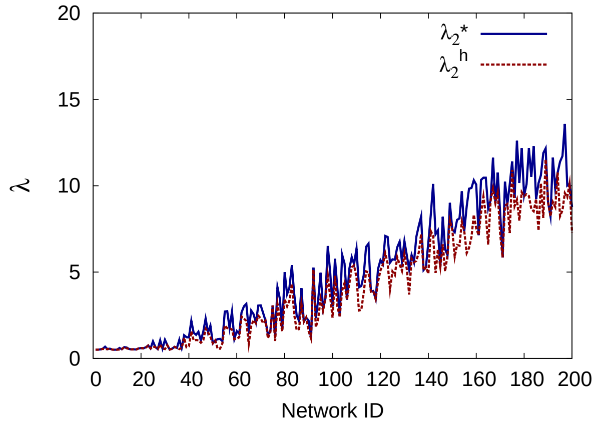


(b) With random capacity links

Figure 5.5: For 200 instances of 30-node random networks for each set, data generation rate obtained by the heuristic strategy vs the upper bound computed by flow model when $M = 1$

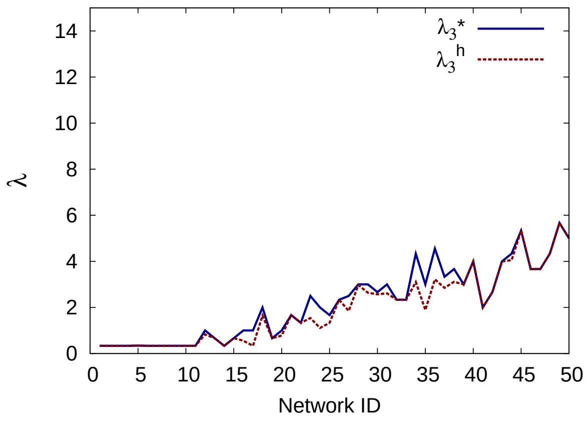


(a) With unit capacity links

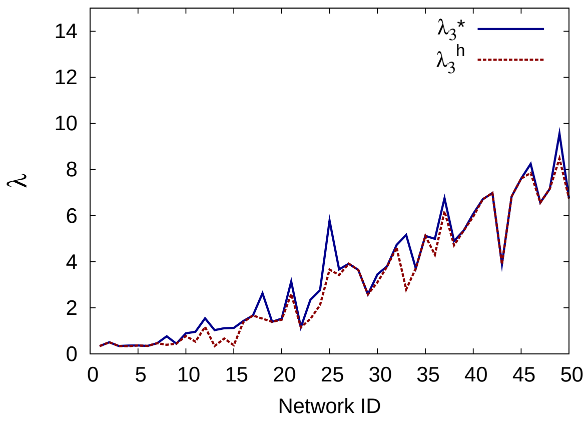


(b) With random capacity links

Figure 5.6: For 200 instances of 30-node random networks for each set, data generation rate obtained by the heuristic strategy vs the upper bound computed by flow model when $M = 2$

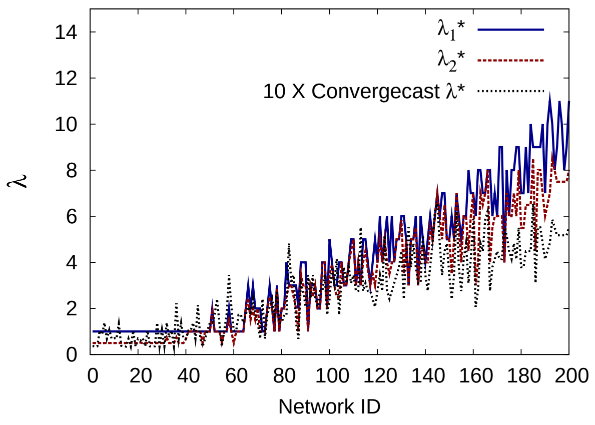


(a) With unit capacity links

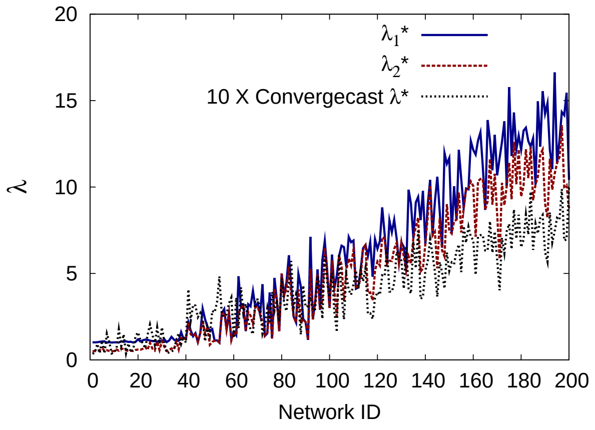


(b) With random capacity links

Figure 5.7: For 50 30-node random networks for each set, data generation rate obtained by the heuristic strategy vs the upper bound computed by flow model when $M = 3$



(a) With unit capacity links



(b) With random capacity links

Figure 5.8: For 200 instances of 30-node random networks for each set, the comparison of throughputs when $M = 1$, $M = 2$ and convergecast

Chapter 6

Conclusion

6.1 Summary

In this dissertation, we have identified and addressed some of the key challenges posed by allowing in-network computation. We have rigorously developed a framework to study the introduction of in-network computation in sensor networks for the class of statistical functions. In particular, we have formulated a discrete-time model that accurately models a single rate wireless network with in-network computation when the sink is interested in the first M moments of the collected data. This formulation is based on a novel and non-intuitive modelling of in-network computation based on conservation of information. From this discrete-time model, we have derived a flow model that computes a tight bound on the maximum achievable throughput for these networks. We have then adapted this flow model to a multi-rate wired network. In addition, we have given methods to obtain near-optimal feasible solutions that can be implemented in a real network for both the wireless and the wired networks. Using these methods, we have also presented numerical evidence that the flow model computes a tight upper bound on the maximum achievable throughput.

We observed that for single rate wireless networks, there are significant gains in the throughput even for relatively large M. These gains are also much larger at lower powers than at higher powers. For these networks, we have also seen that it is possible to achieve close to the optimal throughput using a single path routing.

For wired networks, we have developed a heuristic strategy (involving generation of multiple trees) that can be implemented in practice in a network. This strategy achieves a data generation rate close to the upper bound computed by the flow model. It is based on the idea that to enable the largest number of opportunities for aggregation, some delay needs to be imposed at intermediate nodes. However, note that this does not impact the end to end delay to compute the function for a given wave since the sink cannot compute this function before it has received all the data for that wave. Even for wired networks, throughputs are observed to be substantially higher when in-network computation is allowed compared to the performance of convergecast.

6.2 Extensions

Several extensions of the work done in this thesis are possible. They are as follows.

1. **Energy and lifetime:** Maximizing the lifetime of a wireless sensor network is an important objective due to the scarcity of the power at the nodes. The lifetime of a sensor network can be extended by efficiently using the limited energy resource. In-network computation has the potential to reduce the energy consumption per node and thus increase the overall lifetime. Given a model for power consumption, modifying our flow model to study this problem is straightforward.

2. **Delay:** Several sensor network applications are delay sensitive. Maximizing the throughput which we have studied in this thesis does not ensure the minimization of the delay. For the wireless networks, minimization of delay involves solving the discrete-time model $\mathcal{P}_d(m)$ for a single wave of data, i.e., $m = 1$. The challenge is

that it is a large integer program which is hard to solve for non-trivial networks.

3. **Advanced physical layer techniques:** Studying the impact of advanced physical layer techniques like SIC, superposition coding, etc. with in-network computation on network performance is a promising direction of further research on this topic.

4. **More functions:** We have considered only the statistical functions in this thesis. Extension of the models developed in this thesis to more functions needs to be explored.

5. **Multiple rates:** Wireless networks with multiple rates have not be considered in this thesis. However, extension of the flow model to these type of networks is straightforward.

6. **Power control:** We have considered that all the nodes transmit with the same equal power in the models presented in this thesis. Power control might result in more efficient use of the network resources and thus, is an interesting direction of exploration.

Thus, there are many interesting future directions of work to explore in this field and we believe that our work is a stepping stone for a more comprehensive understanding of the fascinating field of in-network computation.

Bibliography

[1] V. Shah, B.K. Dey, and D. Manjunath. Network flows for functions. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pages 234 –238, aug. 2011.

[2] K. Römer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, December 2004.

[3] A. Giridhar and P.R. Kumar. Computing and communicating functions over sensor networks. *IEEE J. Sel. Areas Commun.*, 23(4):755–764, April 2005.

[4] Aditya Karnik, Aravind Iyer, and Catherine Rosenberg. Throughput-optimal configuration of fixed wireless networks. *IEEE/ACM Trans. Netw.*, 16(5):1161–1174, 2008.

[5] Kamal Jain, Jitendra Padhye, Venkata N. Padmanabhan, and Lili Qiu. Impact of interference on multi-hop wireless network performance. In *the 9th annual international conference on Mobile computing and networking*, pages 66–80, New York, NY, USA, 2003. ACM.

[6] Patrick Stuedi and Gustavo Alonso. Computing throughput capacity for realistic wireless multihop networks. In *MSWiM*, pages 191–198, 2006.

[7] O. Durmaz Incel and B. Krishnamachari. Enhancing the data collection rate of tree-based aggregation in wireless sensor networks. In *5th IEEE ComSoc Conference on*

Sensor, Mesh and Ad Hoc Communications and Networks. San Francisco, USA, pages 569–577, July 2008.

[8] A. Ghosh, O.D. Incel, V.S.A. Kumar, and B. Krishnamachari. Multi-channel scheduling algorithms for fast aggregated convergecast in sensor networks. In *Mobile Adhoc and Sensor Systems, MASS, IEEE 6th International Conference on*, pages 363–372, Oct. 2009.

[9] A. Ghosh, O. D. Incel, V. S. A. Kumar, and B. Krishnamachari. Multichannel scheduling and spanning trees: Throughput; delay tradeoff for fast data collection in sensor networks. *Networking, IEEE/ACM Transactions on*, (99), 2011.

[10] L.R.Ford and D.R.Fulkerson. Constructing maximal dynamic flows from static flows. *Operation Research*, 6:419–433, 1958.

[11] Rainer Burkard, Karin Dlaska, and Bettina Klinz. The quickest flow problem. *Mathematical Methods of Operations Research*, 37:31–58, 1993.

[12] Naoyuki Kamiyama, Naoki Katoh, and Atsushi Takizawa. An efficient algorithm for evacuation problem in dynamic network flows with uniform arc capacity. *IEICE - Trans. Inf. Syst.*, E89-D(8):2372–2379, 2006.

[13] Prasoon Tiwari. Lower bounds on communication complexity in distributed computer networks. *J. ACM*, 34(4):921–938, 1987.

[14] R. Gallager. Finding parity in a simple broadcast network. *IEEE Trans. Info. Theory*, 34(2):176–180, March 1988.

[15] P. Gupta and P.R. Kumar. The capacity of wireless networks. *IEEE Trans. Inf. Theory*, 46(2):388–404, March 2000.

[16] Ritesh Maheshwari, Jing Cao, and Samir R. Das. Physical interference modeling for transmission scheduling on commodity wifi hardware. *IEEE INFOCOM*, April 2009.

[17] Aravind Iyer, Catherine Rosenberg, and Aditya Karnik. What is the right model for wireless channel interference? *IEEE Trans. Wireless. Commun.*, 8(5):2662–2671, 2009.

[18] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *Automatic Control, IEEE Transactions on*, 37(12):1936 –1948, Dec 1992.

[19] J. Luo, C. Rosenberg, and A. Girard. Engineering wireless mesh networks: Joint scheduling, routing, power control and rate adaptation. *IEEE/ACM Trans. Netw.*, 18(3):1387–1400, October 2010.

[20] C. Florens, M. Franceschetti, and R.J. McEliece. Lower bounds on data collection time in sensory networks. *IEEE J. Sel. Areas Commun.*, 22(6):1110–1120, August 2004.

[21] Luisa Gargano and Adele A. Rescigno. Optimally fast data gathering in sensor networks. *Mathematical Foundations of Computer Science*, 4162/2006:399–411, 2006.

[22] Hongsik Choi, Ju Wang, and Esther Hughes. Scheduling for information gathering on sensor network. *Wireless Networks*, 15:127–140, 2009.

[23] S. Kamath and D. Manjunath. On distributed function computation in structure-free random sensor networks. In *IEEE ISIT*, July 2008.

[24] Nilesh Khude, Anurag Kumar, and Aditya Karnik. Time and energy complexity of distributed computation of a class of functions in wireless sensor networks. *IEEE Trans. Mobile Comput.*, 7(5), 2008.

[25] D. Mosk-Aoyama and D. Shah. Fast distributed algorithms for computing separable functions. *IEEE Trans. Inf. Theory*, 54(7):2997–3007, July 2008.

[26] Xujin Chen, Xiaodong Hu, and Jianming Zhu. Minimum data aggregation time problem in wireless sensor networks. In Xiaohua Jia, Jie Wu, and Yanxiang He, editors, *Mobile Ad-hoc and Sensor Networks*, volume 3794 of *Lecture Notes in Computer Science*, pages 133–142. Springer Berlin / Heidelberg, 2005.

[27] Hongxing Li, Qiang Sheng Hua, Chuan Wu, and Francis Chi Moon Lau. Minimum-latency aggregation scheduling in wireless sensor networks under physical interference model. In *the 13th ACM international conference on Modeling, analysis, and simulation of wireless and mobile systems*, pages 360–367, New York, NY, USA, 2010. ACM.

[28] Xiang-Yang Li, XiaoHua Xu, ShiGuang Wang, ShaoJie Tang, GuoJun Dai, JiZhong Zhao, and Yong Qi. Efficient data aggregation in multi-hop wireless sensor networks under physical interference model. In *Mobile Adhoc and Sensor Systems, 2009. MASS '09. IEEE 6th International Conference on*, pages 353 –362, oct. 2009.

[29] Peng-Jun Wan, Scott C.-H. Huang, Lixin Wang, Zhiyuan Wan, and Xiaohua Jia. Minimum-latency aggregation scheduling in multihop wireless networks. In *the tenth ACM international symposium on Mobile ad hoc networking and computing*, pages 185–194, New York, NY, USA, 2009. ACM.

[30] XiaoHua Xu, ShiGuang Wang, XuFei Mao, ShaoJie Tang, and Xiang Yang Li. An improved approximation algorithm for data aggregation in multi-hop wireless sensor networks. In *the 2nd ACM international workshop on Foundations of wireless ad hoc and sensor networking and computing*, pages 47–56, New York, NY, USA, 2009. ACM.

[31] L. Xiang, J. Luo, and A.V. Vasilakos. Compressed data aggregation for energy efficient wireless sensor networks. In *the 8th IEEE SECON*, pages 46–54, 2011.

[32] L. Fratta, M. Gerla, and L. Kleinrock. The flow deviation method: An approach to store-and-forward communication network design. *Networks*, 3(2):97–133, 1973.

[33] Jun Luo, André Girard, and Catherine Rosenberg. Efficient algorithms to solve a class of resource allocation problems in large wireless networks. In *the 7th international conference on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, WiOPT*, pages 313–321, 2009.

[34] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, STOC '84, pages 302–311, New York, NY, USA, 1984. ACM.

[35] Ameer Ahmed Abbasi and Mohamed Younis. A survey on clustering algorithms for wireless sensor networks. *Computer Communications*, 30(1415):2826 – 2841, 2007.

[36] O. Boyinbode, H. Le, A. Mbogho, M. Takizawa, and R. Poliah. A survey on clustering algorithms for wireless sensor networks. In *Network-Based Information Systems (NBiS), 2010 13th International Conference on*, pages 358 –364, sept. 2010.

[37] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms.* McGraw-Hill Higher Education, 2nd edition, 2001.

[38] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, November 1999.