Computing Statistical Functions in Wired Networks

Rajasekhar Sappidi, Catherine Rosenberg and André Girard

***Abstract*—For applications in which a node is interested in a function of the data generated at different sources, in-network computation is a promising approach to improve the network performance. In this paper, we study the problem of computing the first M moments of the data using in-network computation in an arbitrary wired communication network. We are interested in finding a routing and queue management strategy that maximizes the data rate at which the sources could generate new data.**

We first propose a very simple tractable flow model that computes an upper bound on the maximum data generation rate that could be supported in a given network for a given M. To validate the tightness of this upper bound and to provide a practical feasible solution, we then propose a heuristic strategy involving the generation of multiple trees and effective queue management that achieves data generation rates close to this upper bound. This cross-validates the tightness of the upper bound and the goodness of our heuristic strategy. Finally, using the flow model, we provide engineering insights on what in-network computation can achieve.

***Index Terms*—Wired networks, in-network computation, aggregation, statistical functions, flow model, heuristics.**

I. Introduction

CONSIDER an application in which there are several sources that are periodically generating data destined to a single node (called sink in the following) that is interested only in a function of this data. A fire alarm system in which the sink is monitoring the maximum temperature is one such application. Traditionally, the intermediate nodes simply store and forward the packets towards the sink without altering the data in any way. After receiving all the packets, the sink computes the function. This data collecting approach is known as *convergecast* in the literature. As the sink is interested only in the value of the function and not all the data, sending all of it to the sink is a strain on the network resources. If we allow the intermediate nodes to perform partial computations (depending on the function) on the packets they receive, it reduces the volume of the data transferred in the network and could significantly improve the maximum achievable data generation rate. We call this approach, *in-network computation*. In this paper, we identify the challenges when operating a network allowing in-network computation and quantify the gains of this approach in terms of network performance.

In some applications, the sink is interested only in the first M moments of the data (what we call a *statistical function*). The k-th moment of a discrete random variable X, μ_k is defined as the expectation of X^k, i.e.,

$$\mu_k = E(X^k) \tag{1}$$

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

$$\mu_k \simeq \frac{\sum_{i=1}^{N} X_i^k}{N} \tag{2}$$

For instance, in an application maintaining uniform temperature in a building, the sink might be monitoring both the average and the variance, that can be computed with the first two moments. These functions have a property that the sink could compute the first M moments of the data if it knows just the sums of the first M powers of the data. For example, the first two moments can be computed by knowing the sum of the data and the sum of the squares of the data (provided the sink knows the number of sources). So, the intermediate nodes could transmit just the sum and sum of squares of the data they receive, instead of all the raw data. Thus, the data is aggregated resulting in a reduction in the volume of data transferred in the network, improving the network performance. 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.

When in-network computation is allowed, the conservation of packets no longer holds at the intermediate nodes. In our previous work [1], we have proposed a formulation based on conservation of information that models in-network computation in a wireless network. The system considered in that work is a wireless sensor network with transmissions synchronized to time-slots and a single modulation and coding scheme yielding a unit rate on all the links. For this system, we have derived a flow model that computes the maximum achievable data generation rate from a discrete model based on time-slots. We then showed that a simple single path routing achieves a throughput that is very close to the value computed by the flow model.

In this paper, we consider a similar problem on a wired network. The underlying system is very different as we no longer have time-slots and the transmission of packets is not synchronized among different links. We also consider that the links might have different capacities. For this system, we are interested in finding the optimal routing and queue management strategy that would support the maximum data generation rate at the sources. Borrowing the idea of *conservation of information* from our previous study [1], we formulate a flow model that computes the maximum achievable data generation rate for a given wired network utilizing in-network computation. However, unlike wireless networks, for wired networks, the throughput achieved by any single path routing is quite

Manuscript received ; revised . This work was supported in part by Natural Sciences and Engineering Research Council of Canada (NSERC) and by General Motors (GM).

R. Sappidi and C. Rosenberg are with Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada (e-mail: rrsappid@uwaterloo.ca, cath@ece.uwaterloo.ca.

A. Girard is with INRS-EMT, Montréal, Québec, Canada.

Digital Object Identifier 10.1109/JSAC.2013.1304xx.

far from the optimal in general. Multi-path routing requires effective queue management to allow maximum aggregation to occur and to avoid loops. We elaborate more on these issues in Section IV.

The following are our contributions.

1) A very simple tractable flow model that computes a tight upper bound λ_M^* on λ_M, the maximum data generation rate [1] using in-network computation in a wired network when the sink is interested in the first M moments of the data collected by the sources. We also give simple bounds on λ_M^*.
2) A heuristic strategy to operate a network under multiple trees based routing using in-network computation which supports data generation rates close to the upper bound. This cross validates the tightness of the upper bound obtained by our flow model and the efficiency of our heuristic strategy.
3) Engineering insights and the quantification of the possible performance gains using in-network computation.

The rest of the paper is organized as follows. In Section II, we present the related literature. The network model and the problem formulation are given in Section III. In Section IV, we present a heuristic strategy (multiple trees based routing and queue management) to operate the network that achieves throughput close to the upper bound computed by the flow model. We give numerical results and engineering insights in Section V and conclude in Section VI.

II. Related Work

Finding the maximum flow in a wired network using convergecast has been well studied in the literature [2]–[4]. In-network computation adds an interesting dimension to the problem of maximizing the flows in a network. The earliest studies on this problem were made by Tiwari [5]. 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. This problem has received much more interest in the wireless context and many different aspects of this problem have been studied in the literature. Giridhar and Kumar [6] gave a good survey of the existing literature on this topic. Gallager [7] proposed a distributed algorithm that could compute 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. Giridhar and Kumar [8] studied the computation of symmetric functions in a multihop wireless networks under a protocol model of interference and gave asymptotic results on the achievable throughput. Kamath and Manjunath [9] compute MAX efficiently in a structure-free network while Damon and Shah [10] give an elegant scheme based on exponential random numbers to compute separable functions (e.g., SUM) in a structure free network. Information theoretic approach has been taken to tackle the problem of in-network computation in some recent works [11], [12].

To the best of our knowledge, [1] and [13] are the only two works that consider finding optimal routing explicitly when in-network computation is allowed. While in [1] we consider computing statistical functions in a wireless time-slot based network, Shah et al [13] consider a similar problem in wired networks. They address the problem of finding the optimal routing that maximizes the throughput for a given function using in-network computation in a wired network. The class of functions they consider are the ones that could be represented by a computational schema. See Figure 1 which is taken from their work for an example of a schema. Their formulation is heavily dependent on the schemas of the function. The computation time of their algorithm is linear in 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. With the techniques developed in this paper, we address the problem of finding a near-optimal routing for the computation of functions with perfect aggregation. They are represented by $M = 1$ in this paper and we find the maximum achievable data generation rate and near-optimal routings not only for these functions but also for any M.

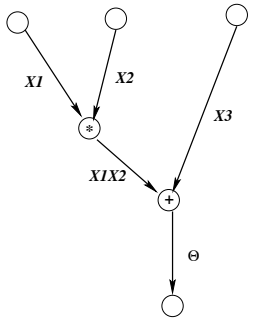


Fig. 1. A schema to represent the function $\Theta = X_1X_2 + X_3$, *courtesy of* [13]

III. Problem Formulation

In this section, we describe our network and formulate the problem to compute the maximum achievable data generation rate using in-network computation in a given network.

A. The Network and its Operation

A directed wired network with n nodes and l links with one of the nodes designated as the *sink* is given. 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. We assume that all the sources are collecting a new raw data packet 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 there is a background mechanism

[1]This rate will be defined precisely in Section III

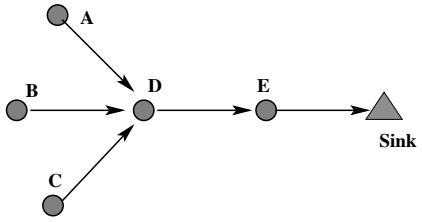


Fig. 2. A network to illustrate in-network computation

that keeps the clocks of the sources synchronized so that they collect new data at the same time.

Let the w-th measurement of source i be $x_i(w)$. Let $\mathbf{x}(w)$ denote the collection of the w-th measurements made by all the sources. Assuming that all the sources 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. 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. We are interested in finding the maximum data generation rate (equivalently throughput) supported by a given network when the sink is interested in computing a given statistical function.

If the sink is interested in the first M moments and in-network computation is allowed, then the intermediate nodes can perform partial computations on the data they receive. Suppose a node is on the path to receive $x_{i_1}(w), x_{i_2}(w) \ldots x_{i_k}(w)$, then it could combine these data into M partial sums defined as $S_p(w) = \sum_{j=1}^{k} x_{i_j}^p, \quad \forall p = 1 \ldots M$ 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 2. 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 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. This will be discussed at length in Section IV.

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, can create 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 can then send only these two partial sum packets to node E. Node E now can combine 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 2 moments.

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 created by 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. The following are the conditions under which nodes perform in-network computation.

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). 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.

For any given network 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 and queue management) that achieves this rate or close to it when implemented in a packet-based network.

In the next sub-section, we propose a flow model that can be used to compute an upper bound on the maximum achievable data generation rate in a given network using in-network computation when the sink is interested in the first M moments of the data.

B. The Flow Model

We define a flow as the stream of packets from a source to the sink. In the following formulation, we ignore the discrete nature of the packets leading to the assumption that the flows are continuous. We also make the a-priori strong assumption to ignore the notion of waves. The impact of this assumption will be discussed later. The idea in a conventional flow model is to balance the incoming and outgoing flows at every node. As

the conservation of flows no longer holds in the conventional sense when in-network computation is allowed, we balance the flow of *information* at every node.

This is better explained on an example first. Consider again the network in Figure 2 where the sink is interested in the first two moments. Every node is a source and produces a raw data flow f_0^s of rate λ. If no in-network computation is allowed, these data flows would have to share the links, e.g., link (D,E) would have to carry 4λ corresponding to the raw data flows f_0^A, f_0^B, f_0^C and f_0^D. If in-network computation is possible, we could try to model the aggregation by saying that D transforms these 4 flows into 2 flows corresponding to the 2 partial sums S_1 and S_2. However, if we modelled the problem this way, we cannot easily write the conservation rules and hence we cannot be sure that information from all the sources has reached the sink. To ensure this, i.e., to track the information from each source, we do not create 2 flows in D out of the 4 raw data flows but 8 (since M=2) virtual flows, 4 of type 1 and 4 of type 2 (one for each source involved in the partial sums at D). To model in-network computation, we allow flows of the same type $p > 0$ to be transmitted *together* on a link and we account for that in the constraints of our flow model. Note that the flows of type $p = 0$ cannot be transmitted together on a link.

For the purpose of tracking in our flow-based model, we use the variable, $y_{i,j}^{s,p}$, which represents the amount of information from source s of type p carried on link (i, j). Whenever aggregation happens at a node, the raw information is replaced with M higher types of information, one for each of the M partial sums. The advantage these higher types have over raw data is that all information of the same type from any number of sources is transmitted at the same time on a link while raw data from only one source could be transmitted at a time. This tracking is for modelling purposes only to make sure that we account for all the information correctly. So, in the flow model, there can be up to $M + 1$ different types of flows for every source in the network, viz., raw data and one for each of the M partial sums. Of course these flows can be split over multiple paths.

In the actual network operation, aggregation does not happen between data belonging to different waves. Our flow model does not keep track of the waves and hence it cannot enforce this condition. However, as was the case in the wireless networks [1], we hope that despite this apparent relaxation, the flow model computes a tight upper bound on the maximum data generation rate in a network when using in-network computation. We show in Section V that it indeed gives a tight upper bound.

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 network when the sink is interested in the first M moments.

$$\mathcal{P}_f : \qquad \underset{\mathbf{u},\mathbf{y},\mathbf{z}}{\text{Max}}\ \lambda \tag{3}$$

$$\sum_j y_{i,j}^{s,0} - \sum_j y_{j,i}^{s,0} + u_i^s = \begin{cases} \lambda & \text{if } i = s \in \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$\sum_j y_{i,j}^{s,p} - \sum_j y_{j,i}^{s,p} - u_i^s = 0 \quad \forall p = 1 \ldots M \tag{5}$$

$$\sum_{p=0}^{M} z_{i,j}^p \le c_{i,j} \quad \forall (i,j) \in \mathcal{L} \tag{6}$$

$$\sum_s y_{i,j}^{s,0} \le z_{i,j}^0 \quad \forall (i,j) \in \mathcal{L} \tag{7}$$

$$y_{i,j}^{s,p} \le z_{i,j}^p \quad \forall s \quad \forall p = 1 \ldots M \quad \forall (i,j) \in \mathcal{L} \tag{8}$$

Constraints (4) and (5) 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 (6) 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 (7) and (8) 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 ($z_{i,j}^t$) 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 ($z_{i,j}^0$) and hence there is a summation over all the sources in (7).

The flow model (3–8) is a linear program that is not very different from the flow model for convergecast [2]. It has $(Mnl+n^2+Ml)$ variables and $((n^2+l)(M+1)+l)$ 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 [14]. Let λ_M^* be the solution to (3–8) 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.

C. Bounds on λ_M^*

In this subsection, we present some bounds on the optimal value computed by the flow model, λ_M^* using the max-flow min-cut theorem. Let C_s be the solution to the max-flow min-cut problem for the source s and the sink in the given network. Define $C \triangleq \text{Min}_s\, C_s$. Then, we have

$$\lambda_1^* \leq C \tag{9}$$

$$\frac{\lambda_1^*}{M} \leq \lambda_M^* \leq C \tag{10}$$

Inequality (9) is true because no node can send more than the rate allowed by the min-cut if the links have to operate within their capacity. The upper-bound in the inequality (10) also follows from this argument. The lower bound in (10) 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 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 (9) is tight by constructing a feasible solution to (3–8) when $M = 1$. The feasible solution is as follows. Let $u_s^s = \lambda$, $y_{i,j}^{s,0} = 0$ and $z_{i,j}^0 = 0$. With this, the problem separates into identical sub-problems, one for every source s. The solution to each of these sub-problems is the max-flow min-cut problem for that source and sink and thus its solution is C_s for a given source, s. Thus, we have a feasible solution whose rate is $C \triangleq \text{Min}_f\, C_f$. Thus, we have

$$\lambda_1^* = C \tag{11}$$

$$\frac{C}{M} \leq \lambda_M^* \leq C \tag{12}$$

To see why λ_M^* could be strictly greater than $\frac{C}{M}$, consider a tree network rooted at the sink with every link having a capacity c_1. Now add directed links of capacity c_2 from every node to the sink resulting in a network like the one in Figure (3). 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

$$\lambda_M^* \geq c_2 + \frac{c_1}{M} > \frac{c_1 + c_2}{M} = \frac{C}{M} \tag{13}$$

In the next section, we construct a heuristic strategy that is implementable in a packet based network and that has a performance close to λ_M^*.

IV. Heuristic Strategy

The flow model (3–8) is a very simple and tractable model to quickly compute an upper bound on the maximum achievable data generation rate for a given network and a given M. As discussed in the last section, there is a need to validate this model, i.e., show that the λ_M^* it computes is close to the maximum achievable data generation rate in the network for the given M. For this, we need to propose a strategy that could be implemented in the network and that would yield a data generation rate close to λ_M^*. By strategy, we mean a set of actions a node executes when it receives or generates a new packet of data. In other words, the strategy helps the node decide if it has to aggregate the new packet and on which outgoing link it has to transmit it. This strategy is not only useful for validating the flow model but it is also useful in itself as a practical method to achieve a near optimal data generation rate in a given network for a given M.

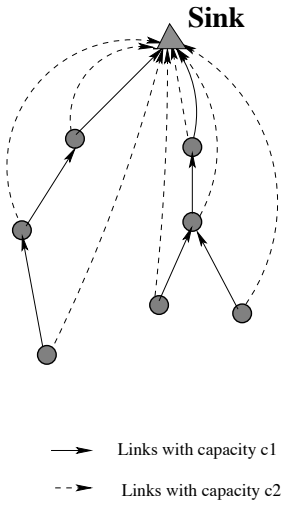


Fig. 3. An example network

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.

For example, consider solving the flow model with $M = 1$ on a 4×4 (directed) grid network, given in Figure 4. 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 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 CPLEX solution 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.

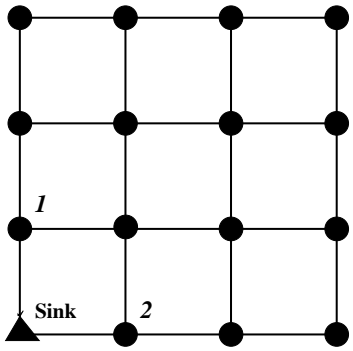
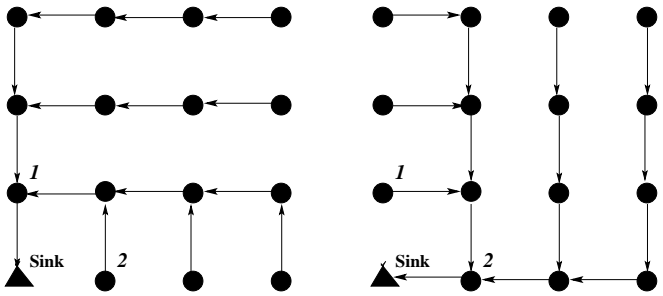


Fig. 4. A grid network: Every link represents two directed links, one in either direction



(a) Tree-1 (b) Tree-2

Fig. 5. Two independent trees of the grid network in Figure 4

In this section, we present a heuristic strategy that could be implemented in a packet-based network and achieves data generation rates close to the value computed by the flow model. This heuristic strategy is based on a set of observations and a set of assumptions. The first observation which is common to both convergecast and in-network computation 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 3 for both convergecast and in-network computation 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 6 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.

For in-network computation, 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 developing a strategy for a general network. This *single tree strategy* is presented in sub-section IV-A.

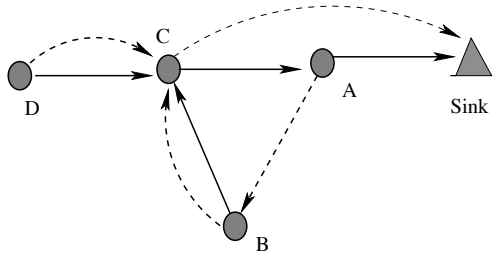


Fig. 6. Multi-path routing

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 IV-B, 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 *wave to tree assignment algorithm* to perform this in sub-section IV-C.

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.

A. Strategy for a Single Tree

Consider a network with a directed tree topology rooted at 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 3). Without loss of generality, assume that some of the nodes (except the sink) are sources and let the data generation rate of each source be λ_M. 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 the G_i sources. To determine if the link (i, j) carries aggregated flows or raw flows, we have to check if $G_i \leq M$. If it is true then the link (i, j) carries only raw data flows, otherwise it carries aggregated flows. Thus, the capacity of the link (i, j) limits the maximum data generation rate of the sources follows.

$$\lambda_M \leq \max\{\frac{c_{i,j}}{M}, \frac{c_{i,j}}{G_i}\} \tag{14}$$

If a node performs in-network computation, then its parent also performs in-network computation according to the rules given in III-A. The link that gives the lowest upper bound on the data generation rate of the sources (computed using equation (14)) forms the bottleneck and determines the maximum data generation rate supported by the network. Thus, we have

$$\lambda_M = \min_{(i,j)}\{\max\{\frac{c_{i,j}}{M}, \frac{c_{i,j}}{G_i}\}\} \tag{15}$$

If a strategy on the tree network supports the λ_M computed by (15), then it is optimal as it an upper bound because of the bottleneck. In calculating this λ_M, there is an implicit assumption that a node always performs in-network computation whenever more than M sources route their data through it. We call such nodes as *aggregators*. This is a best case assumption in terms of the amount of traffic carried. The amount of traffic carried is a function of how much aggregation can be performed. The expected amount of traffic to be carried on a link (i, j) is $\min\{M\lambda_M, G_i\lambda_M\}$. We propose a queue management strategy that ensures that the aggregators always perform in-network computation to minimize the amount of traffic carried and keep it at the expected level.

The nodes first determine if they are aggregators or just forwarders and if they are aggregators, they also determine the total number of packets they expect to receive for a wave from all of their immediate children. This task can be accomplished using either a simple distributed or a centralized algorithm.

At every aggregator node, we assume that there are two buffers. One for processing at the node 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 III-A. 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. If the node is not an aggregator, then the packets are immediately sent to the output queue. This queue management strategy ensures that the traffic on any link does not exceed the expected traffic carried, which in turn does not exceed its capacity because of the way we computed λ_M. 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.

Intuitively, it might seem that the above proposed queue management strategy unnecessarily forces the aggregators to wait for the data from its children and a naive strategy without any waiting could also result in a similar high performance in terms of throughput. However, simulations of this naive strategy on some networks resulted in unstable queues, i.e., the size of the queues increased without bounds. We did not encounter this scenario if there was just a single tree but with multiple trees operating simultaneously, this instability became evident. Thus, we propose the above queue management strategy for achieving a stable higher performance.

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

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 λ for tree T using equation 15
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**

B. 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 from the network.
2) Compute the λ_M supported by it using equation (15).
3) Update the capacities of the links in the network by subtracting the capacity needed by the links in the tree to support λ_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.

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 [15] to extract a tree T from the network $\mathcal{G}'$. DFS($\mathcal{G}'$) [2] 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 [15]) 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.

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

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$(\mathcal{G})'$}

$\forall i \in$ Nodes$(\mathcal{G}')$, color$[i] \leftarrow$ WHITE
$\forall i \in$ Nodes$(\mathcal{G}')$, parent$[i] \leftarrow$ NIL
DFS_VISIT($sink$)

Algorithm 3 Recursive subroutine for DFS: DFS_VISIT(i)

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

color$[i] \leftarrow$ BLACK {Node i is explored}
$m \leftarrow 0$
if i is not $sink$ **then**
  for each vertex j such that $i \in$ Adj(j) **do**
    if color$[j]$ = WHITE **then**
      parent$[j] \leftarrow i$
      color$[j] \leftarrow$ GRAY
      $m \leftarrow m + 1$
      if $m = M$ **then**
        DFS_VISIT(j)
      end if
    end if
  end for
  if $m \neq 0$ **then**
    for each vertex j such that $i \in$ Adj(j) **do**
      if parent$[j] = i$ and color$[j]$ = GRAY **then**
        DFS_VISIT(j)
      end if
    end for
  end if
else
  for each vertex j such that $i \in$ Adj(j) **do**
    if color$[j]$ = WHITE **then**
      parent$[j] \leftarrow i$
      color$[j] \leftarrow$ GRAY
      DFS_VISIT(j)
    end if
  end for
end if

Let λ_M^h 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 λ_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 Section III, 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.

$$\frac{\lambda_M^h}{\lambda_M^*} \geq \frac{c_{min}}{C} \tag{16}$$

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

Algorithm 4 Assignment Algorithm: waves to trees

Input: $\lambda_M^1, \lambda_M^2, \ldots, \lambda_M^k$

$w \leftarrow 0$
for $i = 1$ to k **do**
  $t_i \leftarrow 0$
  $t_i^p \leftarrow 0$
end for
while w $\leq$ `MAX_WAVE` **do**
  for $i = 1$ to k **do**
    $t_i \leftarrow t_i^p + \frac{1}{\lambda_M^i}$
  end for
  $t_{min} \leftarrow \min_i t_i$ and $j \leftarrow \{i : t_i = t_{min}\}$
  $w \leftarrow w + 1$
  Map wave w to Tree j
  $t_j^p \leftarrow t_{min}$
end while

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.

C. Queue Management Strategy for Multiple Trees

Assume that the heuristic in Algorithm 1 has generated k trees with $\lambda_M^1, \lambda_M^2 \ldots \lambda_M^k$ 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

$$\lambda_M^h = \sum_{i=1}^{k} \lambda_M^i. \tag{17}$$

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.

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

TABLE I
THE RANGE OF THE NUMBER OF LINKS IN THE NETWORKS

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

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.

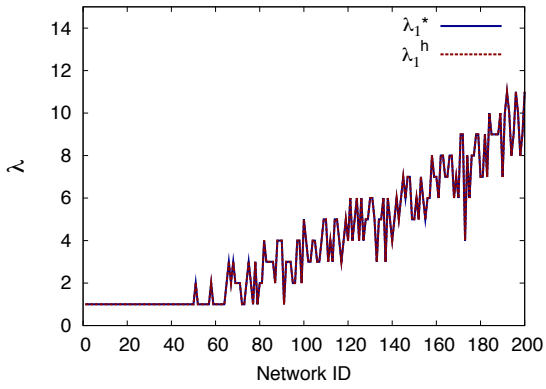
In the next section, we present two types of numerical results. One to show that the heuristic strategy supports a data generation rate that is close to the upper bound computed by the flow model and the other to show the significant performance gains that result from using in-network computation.

V. RESULTS

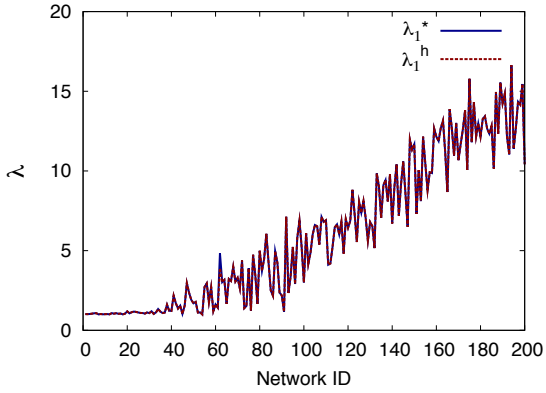
In this section, we first cross validate the flow model and the heuristic strategy by presenting numerical results on four sets of 200 instances of 30-node random networks (two sets with links of unit capacity and two with links of random capacity) that show that the upper bound computed by the flow model is close to the practical achievable data generation rate for different values of M. We, then compare the throughputs of $M = 1$, $M = 2$ and convergecast on the same four sets of 200 instances of 30-node random networks. In all the networks, we assume that all the nodes other than the sink are sources.

A. Validation

We have calculated the data generation rates achieved by our heuristic strategy on four sets of 200 instances of 30-nodes random networks and compared them with the upper bound computed by the flow model. The first two sets of these



(a) With unit capacity links

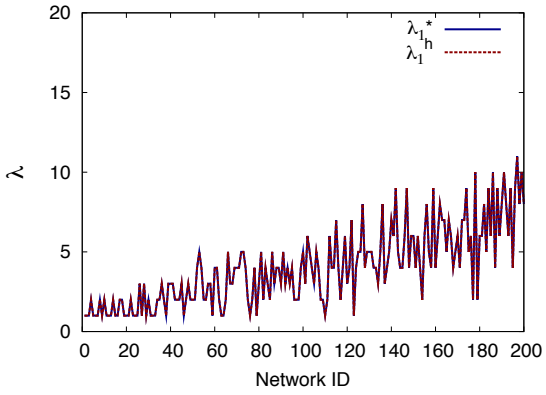


(b) With random capacity links

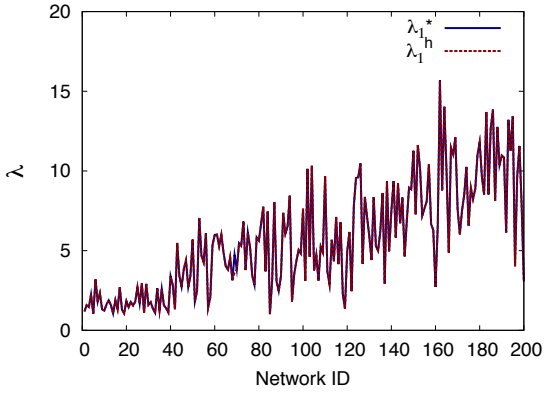
Fig. 7. For the first two sets of 200 instances of 30-node networks, data generation rate obtained by the heuristic strategy vs the upper bound computed by flow model when $M = 1$

random networks are generated 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 discarded the networks that were disconnected). The first set of networks have unit capacity on all their links while the second set have links with random capacity between 1 and 2. For these networks, we assumed that the node 0 is the sink. The next two sets of random networks are generated by distributing the nodes uniformly in a field of dimensions $20m \times 20m$ and assuming that a link exists between two nodes if and only if they are within a fixed distance of each other. For these networks, we have assumed that the sink is at the center of the field. The first set of these networks have unit capacity on all their links while the second set have links with random capacity between 1 and 2. We have ordered the network ids in every set in terms of the number of links in the network (the lower the network id, the lower the number of links). Note that for the 200 networks in a set, multiple networks might have the same number of links. Table I 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 Figures 7 and 8). When $M = 2$, the comparison is given in Figures 9 and 10. For the first two sets of networks, the average difference between the two throughputs is 15.6% for the set of networks with unit capacity

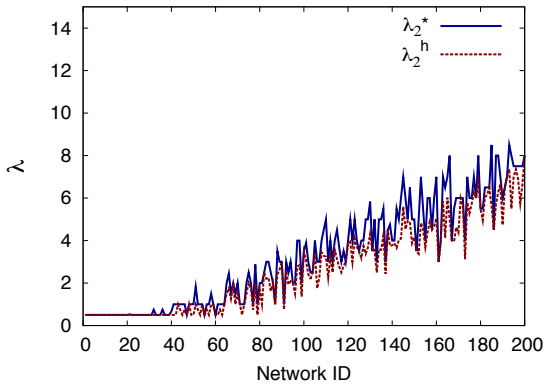


(a) With unit capacity links

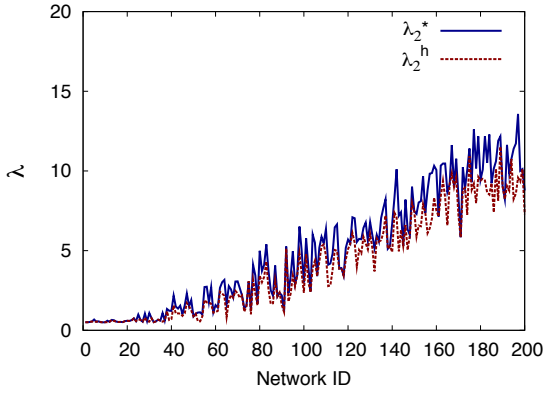


(b) With random capacity links

Fig. 8. For the last two sets of 200 instances of 30-node networks, 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

Fig. 9. For the first two sets of 200 instances of 30-node networks, data generation rate obtained by the heuristic strategy vs the upper bound computed by flow model when $M = 2$

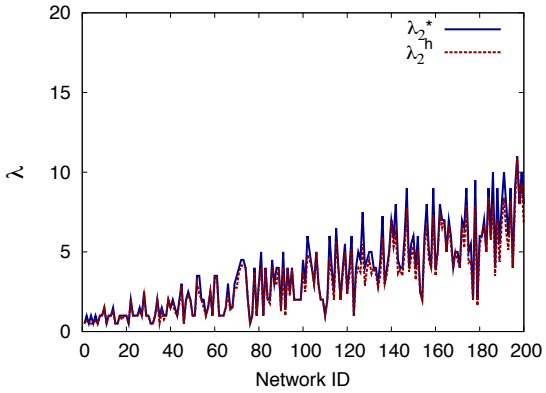
links and it is 14.8% for the set of networks with links of random capacity. For the last two sets of networks, the average difference between the two throughputs is 8.1% for the set with unit capacity links and it is 8% for the set with links of random capacity. As it takes longer to compute the upper bound using the flow model for $M = 3$, we have computed λ_3^* for every fourth network in the first two sets of networks. The comparison for these 50 networks in each set when $M = 3$ is shown in Figure 11. The average difference between the two throughputs is 9.1% 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 very close to the upper bound (irrespective of the method used to generate the random networks). 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.

B. Engineering Insights

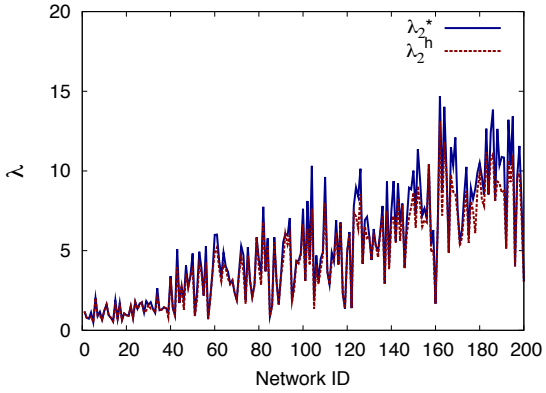
In Figures 12 and 13, we have plotted the throughputs computed by the flow model when $M = 1$, $M = 2$ and for convergecast for all the four sets of networks. The throughput of convergecast is very low when compared to the throughput when in network computation is enabled and hence for ease of illustration, we have magnified its throughput by a factor of ten in the plots in Figures 12 and 13. We see from these plots that in-network computation results in significant improvements in terms of network performance when compared to convergecast (which is a typical store and forward strategy). The achievable throughput when $M = 1$ or $M = 2$ is approximately 10 times higher than that possible with convergecast, especially at higher connectivity.

In Figure 12, 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. However, for the last two sets of networks (see Figure 13) which are generated using a different technique, this is not true and the increase in throughput for convergecast with connnectivity is proportional to the increase in throughput using in-network computation (although lower by a factor of ~ 10). The reason for this is that in the second method, at high connectivity more nodes are directly connected to the sink compared to the number of direct links to the sink in the networks with high connectivity generated using the first method.

In Figure 12, we also 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

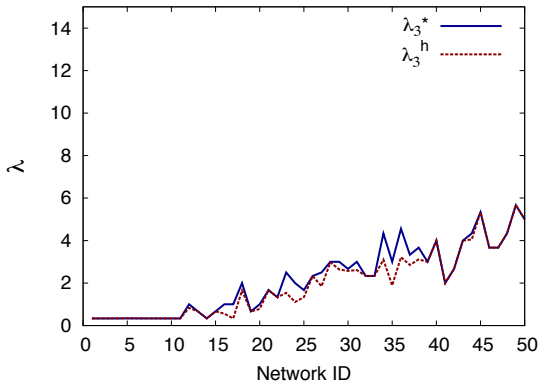


(a) With unit capacity links

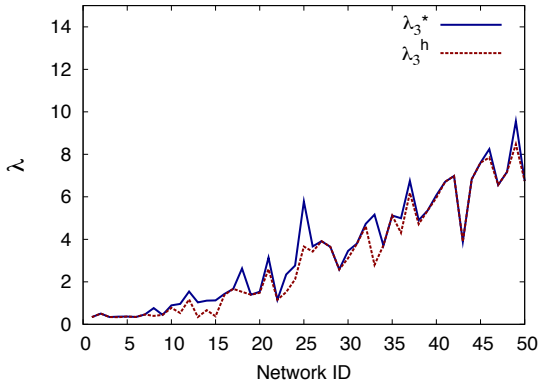


(b) With random capacity links

Fig. 10. For the last two sets of 200 instances of 30-node networks, 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

Fig. 11. For 50 30-node random networks in the first two sets, data generation rate obtained by the heuristic strategy vs the upper bound computed by flow model when $M = 3$

not very different and the average difference between them is only 19.3% for networks with unit capacity links and 19.1% for networks with random capacity links. Likewise, for the last two sets of networks (see Figure 13), the average difference between λ_1^* and λ_2^* is just 12.7% for networks with unit capacity links and 13% for networks with random capacity links.

We conclude this section 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.

In summary, whenever the sink is interested in a statistical function of the data being collected, in-network computation should be used with multiple tree routing.

VI. Conclusion

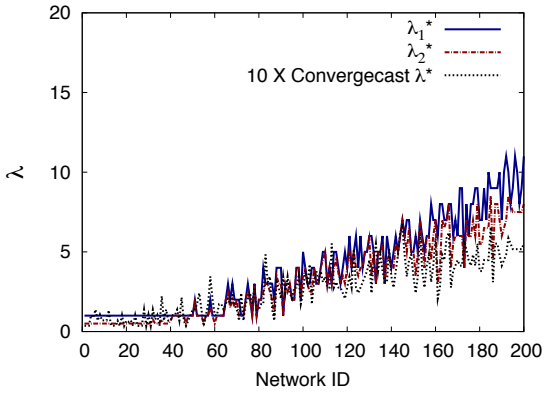
In this paper, we have identified and addressed some of the challenges posed by introducing in-network computation when a sink is interested in a statistical function of the data being collected by some nodes. We have proposed a very simple and compact flow model that computes an upper bound on the maximum achievable data generation rate in a network when the sink is interested only in the first M moments. We have also developed a heuristic strategy that can be implemented in practice in a network. This strategy achieves a data generation rates 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, 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 also note that the network performance when in-network computation is used is substantially higher than the performance of convergecast.

The centralized heuristic strategy proposed in this paper has a near-optimal performance. But a distributed strategy which is not only fault tolerant but also supports a dynamic network topology is desirable. Thus, as part of our future work, we propose to develop such a distributed heuristic strategy.

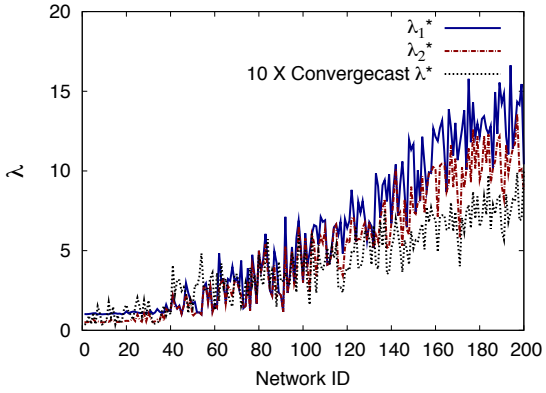
References

[1] R. Sappidi, A. Girard, and C. Rosenberg, "Maximum achievable throughput in a wireless sensor network using in-network computation," *Submitted to IEEE/ACM Trans. Netw.*, 2012. [Online]. Available: http://ece.uwaterloo.ca/∼cath/preprint.pdf

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

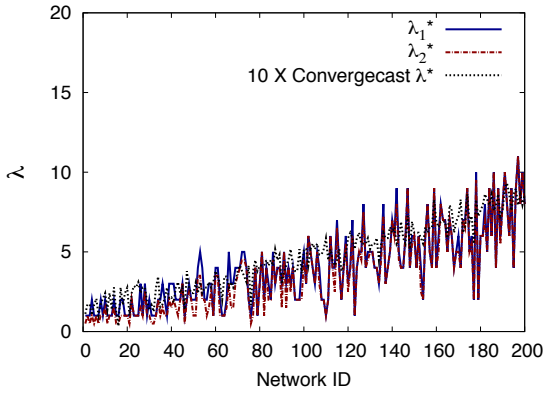


(a) With unit capacity links

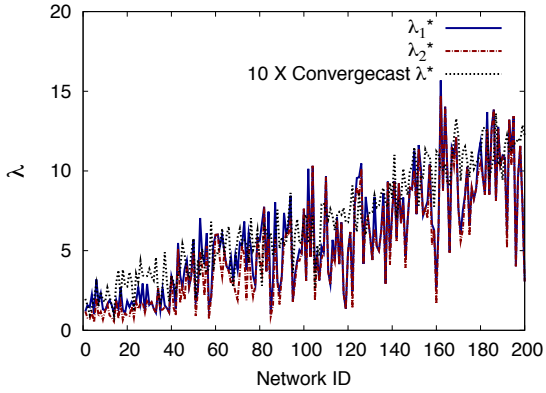


(b) With random capacity links

Fig. 12. For the first two sets of 200 instances of 30-node networks, the comparison of throughputs when $M = 1$, $M = 2$ and convergecast



(a) With unit capacity links



(b) With random capacity links

Fig. 13. For the last two sets of 200 instances of 30-node networks, the comparison of throughputs when $M = 1$, $M = 2$ and convergecast

[3] R. Burkard, K. Dlaska, and B. Klinz, "The quickest flow problem," *Mathematical Methods of Operations Research*, vol. 37, pp. 31–58, 1993. [Online]. Available: http://dx.doi.org/10.1007/BF01415527

[4] N. Kamiyama, N. Katoh, and A. Takizawa, "An efficient algorithm for evacuation problem in dynamic network flows with uniform arc capacity," *IEICE - Trans. Inf. Syst.*, vol. E89-D, no. 8, pp. 2372–2379, 2006.

[5] P. Tiwari, "Lower bounds on communication complexity in distributed computer networks," *J. ACM*, vol. 34, no. 4, pp. 921–938, 1987.

[6] A. Giridhar and P. R. Kumar, "Toward a theory of in-network computation in wireless sensor networks," *IEEE Commun. Mag.*, vol. 44, pp. 98–107, 2006.

[7] R. Gallager, "Finding parity in a simple broadcast network," *IEEE Trans. Inf. Theory*, vol. 34, no. 2, pp. 176–180, Mar. 1988.

[8] A. Giridhar and P. Kumar, "Computing and communicating functions over sensor networks," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 4, pp. 755–764, Apr. 2005.

[9] S. Kamath and D. Manjunath, "On distributed function computation in structure-free random sensor networks," in *Proc. IEEE ISIT*, Jul. 2008, pp. 647–651.

[10] D. Mosk-Aoyama and D. Shah, "Fast distributed algorithms for computing separable functions," *IEEE Trans. Inf. Theory*, vol. 54, no. 7, pp. 2997–3007, Jul. 2008.

[11] O. Ayaso, D. Shah, and M. Dahleh, "Information theoretic bounds for distributed computation over networks of point-to-point channels," *IEEE Trans. Inf. Theoru*, vol. 56, no. 12, pp. 6020 –6039, dec. 2010.

[12] A. Orlitsky and J. R. Roche, "Coding for computing," *IEEE Trans. Inf. Theory*, vol. 47, no. 3, pp. 903–917, 2001.

[13] V. Shah, B. Dey, and D. Manjunath, "Network flows for functions," in *Information Theory Proc. (ISIT), 2011 IEEE International Symposium on*, aug. 2011, pp. 234 –238.

[14] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, ser. STOC '84. New York, NY, USA: ACM, 1984, pp. 302–311. [Online]. Available: http://doi.acm.org/10.1145/800057.808695

[15] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

Rajasekhar Sappidi graduated with B.Tech. in Electrical Engineering from Indian Institute of Technology, Bombay, India in 2007. He worked as a design engineer at Cypress Semiconductors, Hyderabad, India in 2007. Since 2008, he is pursuing Ph.D. in Electrical and Computer Engineering at the University of Waterloo, Canada. His research focus is on modelling and performance evaluation of wireless and wired networks with in-network computation.

Catherine Rosenberg was educated in France (Ecole Nationale Supérieure des Télécommunications de Bretagne, Diplôme d'Ingénieur in EE in 1983 and University of Paris, Orsay, Doctorat en Sciences in CS in 1986) and in the USA (UCLA, MS in CS in 1984), Dr. Rosenberg has worked in several countries including USA, UK, Canada, France and India. In particular, she worked for Nortel Networks in the UK, AT&T Bell Laboratories in the USA, Alcatel in France and taught at Purdue University (USA) and Ecole Polytechnique of Montreal (Canada). Since 2004, Dr. Rosenberg is a faculty member at the University of Waterloo where she now holds a Tier 1 Canada Research Chair in the Future Internet. Her research interests are broadly in networking with currently an emphasis in wireless networking and in smart energy systems. She has authored over 150 papers and has been awarded eight patents in the USA. She is a Fellow of the IEEE. More information can be found at http://ece.uwaterloo.ca/∼cath/.

André Girard received the Ph.D. degree in physics from the University of Pennsylvania, Philadelphia, in 1971. He is an Honorary Professor with INRS-EMT and an Adjunct Professor with École Polytechnique of Montréal, QC, Canada. His research interests all have to do with the optimization of telecommunication networks, and in particular with performance evaluation, routing, dimensioning, and reliability. He has made numerous theoretical and algorithmic contributions to the design of telephone, ATM, IP and wireless networks.