Towards a Queueing-Based Framework for In-Network Function Computation

Siddhartha Banerjee
Department of ECE
The University of Texas at Austin
Email: sbanerjee@mail.utexas.edu

Piyush Gupta
Mathematics of Networks and Communications
Bell Labs, Alcatel-Lucent
Email: pgupta@research.bell-labs.com

Sanjay Shakkottai
Department of ECE
The University of Texas at Austin
Email: shakkott@mail.utexas.edu

***Abstract*—We seek to develop joint aggregation, routing, and scheduling algorithms that, for any graph topology and a large class of functions, have analytically provable performance benefits due to in-network computation as compared to simple data forwarding. To this end, we define a class of functions, the Fully-Multiplexible functions, which includes several functions such as parity, k-th order statistic and range, and for which we can exactly characterize the maximum achievable refresh rate of the network in terms of an underlying graph primitive, the min-mincut. In wireline networks, we show that the maximum refresh rate is achievable by a simple algorithm that is dynamic, distributed, and only dependent on local information. In the case of wireless networks, we provide a MaxWeight-like algorithm with dynamic flow splitting that is shown to be throughput-optimal.**

I. Introduction

In-network function computation is a fundamental paradigm that increases the efficiency of sensor networks vís-a-vís conventional data networks. Sensor nodes, in addition to sensing and communication capabilities, are often equipped with basic computational capabilities. Depending on the task for which they are deployed, a sensor network can be viewed as a distributed platform for computing a specific function of the sensor data. For example, a sensor network for environment monitoring may be concerned with keeping track of the average temperature and humidity in a region. Similarly 'alarm' networks, such as those for detecting forest fires, care only for the maximum temperature. The baseline approach for performing such tasks is to aggregate all the data at a central node and then perform off-line computations; the premise of in-network computation is that distributed computation schemes can greatly increase the performance of the network. However, from the perspective of designing network algorithms, in-network function computation poses a greater challenge than data networks: the freedom to combine and compress packets, as long as the desired information is preserved, destroys the flow conservation laws central to data networks.

Our focus here is to develop a queue-based framework for function computation in sensor networks, and use it to design and analyze network algorithms. By network algorithms, we refer to cross layer algorithms that jointly perform:

1) *Aggregating* data at nodes via in-network computation.
2) *Routing* packets between nodes.
3) *Scheduling* links between nodes for packet transmission.

Cross-layer algorithms for data networks are concerned only with the scheduling and routing aspects. In order to incorporate aggregation, there is a need for a new framework and algorithms for in-network function computation. Keeping in mind the lessons learnt from the success of data networks, our aim is to design algorithms that are *dynamic* (i.e., the algorithm should not assume static network parameters, but rather, use the network state to adaptively learn the network parameters), *robust* (i.e., the algorithm adapts to temporal changes in traffic and network topology), *capable of dealing with a large class of functions* (i.e. if the function being computed by the network changes, then one should only need to make minor changes to the scheduling and routing algorithms) and *applicable to all network topologies*.

Due to the wide range of applications, there are many existing models for such networks. The pioneering work of Giridhar and Kumar [1] considers the function computation problem from the point of view of the capacity scaling for certain classes of functions. Other papers consider the function computation problem from the point of view of information theory [2], [3], communication complexity [4] and capacity for wired networks [5], characterizing various metrics for functions in terms of properties of the graph, function, etc. These works take a 'bottom-up' approach to the problem and are very sensitive to system assumptions; in particular, they consider wireline networks, and focus mainly on obtaining bounds on the capacity rather than designing network algorithms [4], [5]. In contrast, Krishnamachari et al. [6] adopt a more 'top-down' approach whereby they formulate models that abstract out some of the complexity while allowing quantification of performance gains. Their model does not however allow for the design of dynamic algorithms.

For data networks, queue-based models have proved to be an essential tool in developing intuition and designing algorithms for such systems. Such models have provided a common framework for understanding various aspects of network performance such as throughput [7], [8], delay, flow utility maximization, network utility maximization, etc. (for an overview, refer to [9]). In addition, these algorithms have been implemented in real systems [10], including in sensor networks [11], with good results. However, these algorithms are designed for data networks, and can not exploit any potential benefit from in-network computation. In a recent

work, Zhao et al. [12] extend this framework to study fork-and-join processing networks. However their focus is on resource partitioning, and they assume fixed routing.

Using fixed routing in a network usually leads to suboptimal operations as the routes may not be designed to optimize the network performance; in general, even choosing the single best fixed route can perform abritrarily worse than with dynamic routing (see example in Section III). Further, static routing is not robust to temporal changes in the network. However, introducing dynamic routing with in-network computation destroys the flow conservation equations that exist in data networks and networks with fixed flows, as the flow out of a node depends both on inflow as well as (dynamic) packet aggregation at that node. There is a need to come up with new queue-based frameworks and algorithms for function computation in sensor networks, and our paper is a step in this direction.

A. Main Contributions

- We identify a class of functions, the Fully-Multiplexible or FMux functions, for which we provide a tight characterization of the maximum refresh rate with in-network computation.
- Leveraging the results of Massoulié et al. [13], we outline a decentralized, throughput-optimal algorithm for FMux function computation in a wireline network.
- For wireless networks, we develop an alternate algorithm based on dynamic allocation of routes and MaxWeight-type scheduling, and show that this is throughput-optimal for FMux functions.

Note on notation: Throughout the paper, we use calligraphic fonts ($\mathcal{Q}, \mathcal{A}$, etc.) to denote sets and the corresponding capital letter (Q, A, etc.) to denote their cardinality.We also use the shorthand notation $[N] \triangleq \{1, 2, \ldots, N\}$.

II. System Model

At a high level, the system consists of a network of N nodes, one of which is an aggregator, and the rest sensors. Sensor nodes are capable of three tasks: sensing the environment, transmitting to and receiving data from other nodes, and performing computations on the data. Each sensor periodically records the state of its local environment, which is assumed to take values in a finite set. Furthermore, all sensors record values *in a synchronous manner*, and the overall purpose of the system is to repeatedly compute a specific function of the synchronously generated sensor data, and forward it to the aggregator. The metric used to quantify the efficacy of an algorithm is the maximum synchronous rate at which the sensors can record data such that the required function of the data can be forwarded to the aggregator in a stable manner[1]. This rate is henceforth referred to as the *maximum refresh rate* of the network.

Communication Graph: We model the topology of the sensor network as a directed graph $G(\mathcal{N}, \mathcal{L})$ with N nodes with a designated aggregator a, and L directed links which determine the connectivity between nodes. Directed link $(u, v) \in \mathcal{L}$ represents a communication channel from node u to node v (in wireline this corresponds to a physical channel, while in wireless it represents that the nodes are in radio range).

Transmission Model: Following the convention in literature [8], [13], we consider a continuous time model for wireline systems, whereas in the case of wireless networks, we assume that time is slotted[2]. In wireline networks, we define a vector of link rates $\hat{\mathbf{c}} = \{c_{uv}\}_{(u,v)\in\mathcal{L}}$; one bit is assumed to traverse a link $(u, v) \in \mathcal{L}$ with a random transit time with distribution *Exponential*(c_{uv}). The transit times are independent across links and across packets crossing the same link.

In the case of wireless networks, in addition to capacity constraints, there are interference constraints. $\mathcal{I} \subseteq 2^L$ is defined to be the set of independent sets, i.e., valid schedules that obey the interference constraints. Given an independent set I, $\mathbf{c}(I) = \{c_{uv}(I)\}_{(u,v)\in L}$ is said to be *admissible* if the link-rates can be achieved simultaneously in a time slot. Γ is the set of all such admissible rate vectors and is assumed to be time invariant. Further, we assume that $c_{uv}(I) \leq c_{\max}\ \forall\ (u, v) \in \mathcal{L}, I \in \mathcal{I}$. Finally, $\hat{\mathbf{c}}$ is said to be *obtainable* if $\hat{\mathbf{c}} \in \mathcal{CH}(\Gamma)$, the convex hull of Γ. An obtainable link-rate vector can be achieved by time-sharing over admissible link-rate vectors. For details, refer to [14].

Up to this point, the system is identical to one considered for data networks. Now in order to highlight the unique features of the model and its connections to a physical sensor network performing function computation, we consider the following example. In the process, we also indicate the gains achievable via in-network computation versus data download and processing at the aggregator.

Example: Consider a wireline grid of N temperature sensors, with a single aggregator at the center, engaged in recording the maximum temperature in the area. Each node is connected to its four immediate neighbors in the grid via links with a fixed capacity c. Every node senses the temperature *synchronously*, and the aggregator desires the MAX of these synchronous measurements. Suppose the network operates by transferring all the data to the aggregator, and then calculating the MAX offline; the maximum rate at which the measurements can be made is then $\Theta(\frac{1}{N})$, as all the packets must pass through one of the 4 links entering the aggregator. On the other hand, if we allow in-network computation, wherein nodes on receiving multiple packets can discard all but the one with highest value, then the network can operate at a rate of $\Theta(1)$, as the bottleneck is now the minimum-cut of the graph (again the 4 links entering the aggregator). In subsequent sections, we show that for certain functions like MAX, and any network, the maximum possible refresh rate can be related thus to minimum-cuts in the network. Further, there are dynamic algorithms that support rates up to the maximum refresh rate.

[1]By stability, we refer to the standard queueing notion of finite average queue backlogs [8], [9]

[2]These assumptions are made so that the results presented here align with existing work, and are not crucial to the analysis. It is possible to convert either to discrete or continuous time

With this example in mind, we outline the rest of our system model.

Traffic Model: We consider a symmetric arrival rate, where each sensor node senses the environment synchronously at a rate λ (the ***refresh rate*** of the network). The aim of network algorithm design is to support the maximum possible λ while ensuring that the network is stable.

Suppose the sensor readings take values in a finite set $\mathcal{X}$ (stored by the sensor in a packet of size $\log_2|\mathcal{X}|$). In case of wireline networks, sensing is performed (and hence, packets are generated) synchronously *at all nodes* following a Poisson process with rate λ. In case of wireless networks, the arrival process of packets (due to sensing) $A_i[t]$ in time slot t consists of a number of packets per time slot generated in a synchronous manner , i.e., $A_i[t] = A_j[t] = A[t]\,\forall\, i,j \in \mathcal{N}$, and further $A[t]$ is i.i.d across time. In this case, we define the refresh rate as $\lambda = \mathbb{E}[A[t]]$, and also define $A_{\max}$ to be the maximum allowed value of $A[t]$.

We associate a set of simultaneously generated packets with a unique identifier called the ***round number***, which represents the time when the packet was generated. The arrival of round r is equivalent to the generation at each node $i \in [N]$ of a packet containing the sensor reading $x_i^r \in \mathcal{X}$. Thus $\{x_i^r\}_{i\in[N]}$ is the collection of variables corresponding to measurements at all the sensors for round r, and the aim of the network is to compute a function $f(\{x_i^r\}_{i\in[N]})$ and relay it to the aggregator.

Now in order to develop a queueing model, we need a framework to capture data aggregation operations. As mentioned before, our primary goal is to explore the benefits of in-network computation versus data-download. To this end, we restrict our attention to a specific class of functions, the FMux functions, for which we can exactly quantify the gains from in-network computation. The intuition behind the FMux class is that these functions support *maximum compression upon aggregation*; when two (or more) packets combine at a node, the resultant packet has the same size as the original packets. We now define it formally.

Computation Model: We assume that the function f is *divisible* [1] (intuitively, this means that for any partition of the nodes, f can be computed by performing a local computation over each set in the partition, and then aggregating them together), and use f_k to denote the function operating on k inputs, i.e., $f_k : \mathcal{X}^k \to \mathcal{R}(f,k)$, where $\mathcal{R}(f,k)$ denotes the range of function when it takes k inputs.

A function f is said to be ***Fully-Multiplexible or FMux*** if $\mathcal{R}(f,k) = \mathcal{R}(f,j) = \mathcal{R}(f)$ for all $j,k \in [n]$. In other words, the output of a FMux function lies in the same set independent of the number of inputs. Some important examples of FMux functions are range, k-th order statistics, parity, etc.. As mentioned before, in this work we will focus on FMux functions as they most clearly exhibit the effects of in-network computation (in that we have tight bounds for their refresh rate). However this framework can be extended to a more general class of functions (Refer to [14] for details).

As a representative example of FMux functions for defining the queueing model and algorithms, consider the parity function; $\mathcal{X} = \{0,1\}, f(\{x_1,x_2,\ldots,x_N\}) = x_1 \oplus x_2 \oplus \ldots \oplus x_N$, where $\oplus$ represents the binary XOR operator. Upon sensing, node i stores the value x_i as a packet of size $\log_2|\mathcal{X}| = 1$ bit. Next, when two or more packets of the same round arrive at a node, they are combined using the XOR operation. Finally, the aggregator obtains the parity by taking XOR of all the packets of a given round that it receives. We now develop a queueing model for FMux functions.

Queueing Model: Each node maintains a single queue, with packets corresponding to different rounds (recall this means that each packet in the queue has a unique time-stamp representing when it was generated). When a packet corresponding to round r arrives at node i from any other neighboring node, it is combined with node i's own packet corresponding to round r to result in a single packet of the same size (using the FMux property in general, e.g. by taking XOR for parity). In the case where node i does not have a packet of round r in queue, it needs to store the new packet (see [14] for details).

III. Maximum Refresh Rate and Tree Packing

Given the above system, it is unclear what routing structures are required for efficient in-network computation. Existing works assume that routing is done on a single *aggregation tree*, where each node aggregates data from its children before relaying it to its parent. However it's not *a priori* evident that a single optimal tree, or a collection of optimal trees exists (or indeed that acyclic aggregation structures are sufficient), and if it does, how it can be found dynamically.

In this section, we derive an algorithm-independent upper bound on the refresh rate for FMux computation. By focusing on the flow of information from sensor nodes to the aggregator, we are able to express the bound in terms of an underlying graph primitive- the min-mincut of the graph. Next we construct a class of throughput-optimal randomized policies, thereby obtaining a tight characterization of the maximum refresh rate. In the process, we show the existence of an optimal collection of aggregation trees. To understand the import of this result, consider the following example.

Example: Let G be the complete graph on N nodes, with every edge having capacity 1. If we use a single aggregation tree for routing, then the maximum possible refresh rate is 1, as every edge is a bottleneck. However, by using a collection of aggregation trees (in fact, it can be shown that a particular set of $N-1$ trees are sufficient), one can achieve a refresh rate of $N-1$, which matches the min-mincut of the graph.

Given a rate vector $\hat{\mathbf{c}} \in \mathcal{CH}(\Gamma)$ and any node $i \in \mathcal{N}$, we define the min-cut between the node i and the aggregator a as $\delta_i(\hat{\mathbf{c}}) \triangleq \min_{\{S\subset\mathcal{N}: i\in S, a\notin S\}} \sum_{u\in S, v\notin S} \hat{c}_{uv}$. Further, we define the min-mincut of the network under rate vector $\hat{\mathbf{c}} \in \mathcal{CH}(\Gamma)$ as $\delta^*(\hat{\mathbf{c}}) = \min_{i\in\mathcal{N}} \delta_i(\hat{\mathbf{c}})$. It can be shown (for details, see [14]) that for stability, a necessary condition on the refresh rate is:

$$\lambda > (\log_2|\mathcal{R}(f)|)^{-1} \max_{\hat{\mathbf{c}}\in\mathcal{CH}(\Gamma)} \delta^*(\hat{\mathbf{c}}). \tag{1}$$

Now we can use a classical theorem of Edmonds [15] to derive an algorithm to achieve this rate[3]. Let $\mathcal{T}$ be the set of all directed spanning trees of G aggregating at a (i.e., a set of directed, acyclic subgraphs such that each node has a unique directed path to a). The max-spanning-tree-packing number, $\Lambda^*(G)$ is defined to be the solution to the following optimization problem:

$$\begin{array}{ll} \text{Maximize} & \sum_{\tau\in\mathcal{T}} \lambda_\tau, \\ \text{subject to} & \sum_{\tau\in\mathcal{T}:(u,v)\in\tau} \lambda_\tau \le c_{uv} \ \ \forall\,(i,j)\in E, \\ \text{and} & \lambda_\tau \ge 0 \forall\ \ \tau\in\mathcal{T}. \end{array}$$

For such a system, Edmonds' theorem [15] states that $\delta^*(G) = \Lambda^*(G)$, i.e., there exists a tree packing which has the same weight as the min-mincut of the graph. Using this and the techniques of Andrews et al. [8], we can construct a static, randomized throughput-optimal scheme as follows: given the optimal rate point $\tilde{\mathbf{c}}^*$ from equation 1, we can use Edmonds' theorem to construct a tree packing, and split the incoming flow according to that packing to obtain a stable routing (i.e., associate each round with an aggregation tree) and scheduling (using $\tilde{\mathbf{c}}^*$) algorithm (refer to [14] for details). Combined with equation 1, this gives the following tight characterization of the maximum refresh rate of the network. We state this theorem for wireless networks, as an equivalent theorem for wireline networks can be obtained as a special case.[4]

Theorem 1. *Consider a network performing in-network computation for an FMux function f. The maximum refresh rate is defined as:*

$$\lambda^* = (\log_2|\mathcal{R}(f)|)^{-1} \max_{\hat{\mathbf{c}}\in\mathcal{CH}(\Gamma)} \delta^*(\hat{\mathbf{c}}). \tag{2}$$

Then a refresh rate of λ can not be stabilized by any algorithm if $\lambda > \lambda^*$, and there exists a static, randomized algorithm to stabilize it if $\lambda < \lambda^*$.

The problem with a static algorithm is that it needs prior calculation of the min-mincut and associated optimal rate point (to find the optimal aggregation tree packing). A better alternative is to use queues as proxy for learning these through dynamic algorithms based on the current system state (similar to the Backpressure algorithm [7], [8] for data networks). The rest of the paper deals with developing such algorithms.

IV. Scheduling with Random Packet Forwarding in Wireline Networks

In this section we give a routing algorithm for wireline networks based on random packet forwarding with aggregation. This algorithm is based on a remarkable algorithm for one-to-all network broadcast in wireline networks by Massoulié et al. [13], which demonstrates that performing random 'useful' packet forwarding by the nodes achieves the min-mincut bound. We come up with an analogous notion of a useful packet for in-network aggregation, and thereby obtain a 'dual' version of their algorithm applicable to FMux function computation in wireline networks.

A critical aspect of the broadcast algorithm is that the trace of a round *always follows a spanning tree*. For a similar property to hold in aggregation, one needs to ensure that a transmitted packet is always aggregated (i.e., combined with another packet from the same round, e.g., using XOR for parity), and more importantly, *does not isolate*[5] *any other neighbor's packet*. To this end, we define a packet in node i to be useful to neighbor j if *(a)* j has a packet of the same round (hence ensuring aggregation); and *(b)* transferring the packet to j does not result in an isolated neighbor k of i (for details, refer to [14]). The routing algorithm now performs random useful packet forwarding with aggregation in a work conserving manner (i.e., whenever a link is idle). Formally we have:

Input: An idle link (u, v), i.e., a link with no packet transmitting on it currently.
Step 1: If $\nexists$ useful packets across (u, v), leave link idle.
Step 2: Otherwise, pick a useful packet uniformly at random and start transmitting.

Algorithm 1: Random useful packet forwarding.

And finally we have the main theorem for the stability of the algorithm. For the proof, refer to [14].

Theorem 2. *Under algorithm 1, the network is stable if $\lambda < (\log_2|\mathcal{R}(f)|)^{-1}\delta^*$, where min-mincut $\delta^* \triangleq \min_{S\in\mathcal{S}} \sum_{v\in S}\sum_{u\notin S} c_{uv}$.*

V. Scheduling With Aggregation-Tree Routing in Wireless Networks

The presence of interference in wireless networks necessitates efficient scheduling of independent sets in addition to routing. Dynamic scheduling in order to achieve the optimal refresh rate now requires an alternate routing approach, which we outline here. Unlike the previous section where routing (over the wireline network) was performed via random packet forwarding, this algorithm is based on allocating the route to be followed by the packets of each round, and then scheduling under these routing constraints. Building on the intuition of Section III, the routes we use here are aggregation trees. The algorithm thus consists of two components:

- The routing component maps incoming rounds of packets to aggregation trees. Once a round is loaded on a tree, all packets of that round follow the edges of the tree to reach the aggregator.
- The scheduling component uses the routing information (aggregation tree) of each packet to determine an optimal independent set for transmission.

Let $\mathcal{T}$ be the set of aggregation trees used for routing. The routing algorithm allocates each incoming packet to a tree, and we define $A_i^\tau[t]$ to be the resultant arrivals on tree τ. Before

[3] Although Edmonds' theorem is originally for broadcast, we can modify it for aggregation by reversing the directions of all edges.

[4] A similar framework has been studied in [5] for a network-coding based formulation in wireline networks. There, similar bounds on the refresh rates have been obtained but no explicit algorithms have been developed.

[5] A packet is said to be isolated if none of its neighbors have a packet from the same round, thus preventing aggregation of this packet.

specifying the general algorithm, we first define the queueing dynamics for a single aggregation tree. Given an aggregation tree and a round of packets constrained to be routed along it, we impose that *a node only transmits a packet belonging to that round after aggregating all the packets from its children on that tree* (analogous to non-isolation in random packet forwarding). Each node i on an aggregation tree τ thus maintains two queues:- $\mathcal{Q}_i^{\tau,u}[t]$ corresponding to unaggregated packets (i.e., those waiting for same round packets from children nodes of i on τ), and $\mathcal{Q}_i^{\tau,a}[t]$ corresponding to aggregated packets (i.e., those which have aggregated all corresponding packets from children nodes of i on τ). When node i receives packets corresponding to round r from all its children nodes, it shifts an aggregate round r packet (e.g., taking XOR for parity) from the unaggregated queue to the aggregated queue.

The routing is performed using a *greedy tree-loading policy*, wherein all incoming rounds in a time slot are loaded on the tree with smallest sum queue. Formally, we have:

Input: Time slot t, queues $\{\mathcal{Q}_i^{\tau,u}[t], \mathcal{Q}_i^{\tau,a}[t]\}_{i\in\mathcal{N},\tau\in\mathcal{T}}$, incoming rounds.
Step 1: Calculate $W_\tau = \sum_{i\in\mathcal{N}} (Q_i^\tau[t])$ for all $\tau \in \mathcal{T}$.
Step 2: Find the minimum loaded tree $\tau^*[t]$ as:

$$\tau^*[t] = \arg\min_{\tau\in\mathcal{T}} W_\tau[t].$$

Step 3: Assign all incoming rounds to aggregation tree τ^* and store the packets in the appropriate queues.

Algorithm 2: Greedy tree-loading algorithm.

The scheduling algorithm is similar to the MaxWeight policy [8], in that it picks a maximum independent set with weights given by the product of the rate and the maximum queue across an edge. Formally we have:

Input: Time slot t, queues $\{\mathcal{Q}_i^{\tau,u}[t], \mathcal{Q}_i^{\tau,a}[t]\}_{i\in\mathcal{N},\tau\in\mathcal{T}}$, incoming packets $\mathcal{A}_i[t]$, admissible rate region Γ.
Step 1: Place packets arriving on tree τ at node i in $\mathcal{Q}_i^{\tau,u}[t]$ for non-leaf nodes, and $\mathcal{Q}_i^{\tau,a}[t]$ for leaf nodes.
Step 2: Calculate $P_{ij}[t] = \max_{\tau\in\mathcal{T}:(i,j)\in\tau} Q_i^\tau[t]$. Also define $\tau(i,j)$ as the tree which maximizes $P_{ij}[t]$.
Step 3: Compute schedule $\mathbf{c}^*[t]$ as:

$$\mathbf{c}^*[t] = \arg\max_{\mathbf{c}\in\Gamma} \sum_{(i,j)\in\mathcal{L}} P_{ij}[t]c_{ij}[t].$$

Step 4: Consider link (i,j). If $c_{ij}^*[t] > 0$, then transmit the first $\min(c_{ij}^*[t], Q_i^{\tau(i,j),a}[t])$ packets from $Q_i^{\tau(i,j),a}[t]$.

Algorithm 3: MaxWeight scheduling algorithm.

Finally we have a theorem stating the throughput-optimality of this algorithm. The proof is given in [14].

Theorem 3. *The dynamic queue based policy consisting of greedy tree loading (Algorithm 2) and MaxWeight scheduling (Algorithm 3) stabilizes the system for any refresh rate λ that is less than the maximum refresh rate λ^*.*

VI. Conclusions

We have presented a queue-based framework for in-network function computation, and used it to gain insights into designing dynamic algorithms for such systems, and quantify the gains over data-download. For wireline networks, we have extended the random routing scheme of Massoulié et al. [13] for aggregation. For wireless networks, we have provided a fixed-routing via dynamic flow splitting along with MaxWeight-like scheduling that is shown to be throughput-optimal. The results have been presented for a specific class of functions (the FMux functions). For extensions to a more general class of functions, refer to [14].

Acknowledgments

This work was supported in part by AFOSR under grant FA9550-09-1-0317 and by NSF under grants CNS-0519535, CNS-0721380 and CNS-096439.

References

[1] A. Giridhar and P. R. Kumar, "Computing and communicating functions over sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 4, pp. 755–764, 2005.

[2] A. Orlitsky and J. R. Roche, "Coding for computing," *IEEE Transactions on Information Theory*, vol. 47, no. 3, pp. 903–917, 2001.

[3] N. Ma, P. Ishwar, and P. Gupta, "Information-theoretic bounds for multiround function computation in collocated networks," *CoRR*, vol. abs/0901.2356, 2009.

[4] H. Kowshik and P. R. Kumar, "Optimal computation of symmetric boolean functions in tree networks," in *IEEE International Symposium on Information Theory — ISIT 2010*, IEEE, July 2010.

[5] R. Appuswamy, M. Franceschetti, N. Karamchandani, and K. Zeger, "Network coding for computing: Cut-set bounds," *IEEE Transactions on Information Theory*, vol. 57, no. 2, pp. 1015–1030, 2011.

[6] B. Krishnamachari, D. Estrin, and S. B. Wicker, "The impact of data aggregation in wireless sensor networks," in *Proc. the 26th IEEE Internat. Conf. Distributed Computing Systems*, pp. 575–578, 2002.

[7] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 4, pp. 1936–1948, December 1992.

[8] M. Andrews, K. Kumaran, K. Ramanan, A. L. Stolyar, R. Vijayakumar, and P. Whiting, "Scheduling in a queueing system with asynchronously varying service rates," *Probability in Engineering and Informational Sciences*, vol. 14, pp. 191–217, 2004.

[9] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Foundations and Trends in Networking*, vol. 1, pp. 1–144, 2006.

[10] U. Akyol, M. Andrews, P. Gupta, J. D. Hobby, I. Saniee, and A. L. Stolyar, "Joint scheduling and congestion control in mobile ad-hoc networks," in *Proc. IEEE Infocom.*, pp. 619–627, 2008.

[11] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali, "Routing without routes: the backpressure collection protocol," in *Internat. Workshop on Inform. Processing in Sensor Networks*, pp. 279–290, 2010.

[12] H. Zhao, C. H. Xia, Z. Liu, and D. Towsley, "A unified modeling framework for distributed resource allocation of general fork and join processing networks," in *Proc. Ann. ACM SIGMETRICS Conf.*, 2010.

[13] L. Massoulié, A. Twigg, C. Gkantsidis, and P. Rodriguez, "Randomized decentralized broadcasting algorithms," in *Proc. IEEE Infocom.*, pp. 1073–1081, 2007.

[14] S. Banerjee, P. Gupta, and S. Shakkottai, "Towards a queueing-based framework for in-network function computation," tech. rep., 2010. `http://arxiv.org/abs/1105.5651`.

[15] J. Edmonds, "Edge-disjoint branchings." in *Combinatorial Algorithms*, 1972.