Streaming Big Data meets Backpressure in Distributed Network Computation

Apostolos Destounis, Georgios S. Paschos
Huawei Technologies France Research Center

Iordanis Koutsopoulos
Athens University of Economics and Business

***Abstract*—We study network response to a stream of queries that require computations on remotely located data, and we seek to characterize the network performance limits in terms of maximum sustainable query rate that can be satisfied. The available network setup consists of (*i*) a communication network graph with finite-bandwidth links over which data is routed, (*ii*) computation nodes with certain computation capacity, over which computation load is balanced, and (*iii*) network nodes that need to schedule raw and processed data transmissions. Our aim is to design a universal methodology and distributed algorithm to adaptively allocate resources in order to support *maximum query rate*.The proposed algorithms extend in a nontrivial way the backpressure (BP) algorithm to take into account computations carried out in the presence of query streams. They contribute to the fundamental understanding of network computation performance limits when the query rate is limited by both the communication bandwidth and the computation capacity, a classical setting that arises in streaming big data applications in network clouds and fogs.**

***Index Terms*—Backpressure (BP) routing, Cloud Computing, Fog Computing, In-network Computation, Resource Allocation.**

I. Introduction

In recent years, the gamut of services and applications that rely on big data analytics and computations has significantly expanded. The game-changer in these platforms lies in their ability to perform computations and deliver results in real time, in the form of a service or an application. This proliferation is much attributed to the advent of smart-phones and wearable devices with multi-modal embedded sensors that facilitate data collection, and it has created the need for impromptu service delivery to the mobile user. For instance, mobile augmented-reality apps rely on real-time data retrieval from distributed data sources to offer a sense of an augmented world, supplemental to the real one. In mobile crowd-sensing apps, smart-phones contribute data which is aggregated, and the aggregate is provided in real time as a service to app subscribers. Further, the mobile health sector supports real-time personalized medical advice based on analytics on dynamic diverse data collected from smart-phones and wearable devices to help people self-manage their health.

Data computations and analytics may be performed either (*i*) at the back-end i.e. at large-scale computation platforms or high-performance computing clouds of interconnected nodes with computation and storage capabilities, or (*ii*) at network-edge components i.e. mobile devices or base stations, according to the newly coined concept of edge computing and the "fog" [1] wherein nodes with computation and storage resources are wirelessly connected.

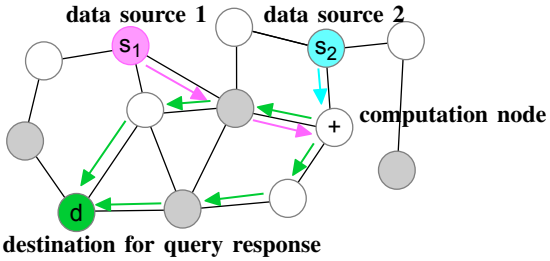


Fig. 1. Illustration of network computations. White nodes are computation nodes and shaded nodes are forwarders. Colorful nodes are sources and destinations, and colorful arrows denote routing of raw and processed data.

A unifying model that captures the scenarios above is the following. A set of nodes are connected in a network through links of certain communication bandwidth, and each node has some computation capacity resources. Sequences of queries for computation are generated in a streaming fashion. Each query sequence is characterized by a type of computation, the sources where the data are collected from, and the destination where results are to be delivered. In order to satisfy each computation request, an algorithm is needed to perform the following tasks: (*i*) first, *retrieval* of data pertinent to the query, possibly from multiple source nodes in the network. These may be either nodes that hold stored data such as databases in a computing cluster or mobile devices that provide data on the spot. (*ii*) Next, *determination of computation nodes* in the network that will do computations on the data; these nodes may have diverse computation resources. Computation may involve aggregates, functions of or statistics on the data. (*iii*) *Multi-hop routing* of the unprocessed (raw) data through the network from the source nodes to computation nodes, and *multi-hop routing* of the computation results (processed data) from computation nodes to the destination, (*iv*) *scheduling* of traffic streams of unprocessed and processed data corresponding to different queries through computation nodes of limited computational capacity and through links of limited bandwidth. In this work, we ask the following question: Given a network graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$ with links of limited communication bandwidth and nodes of limited computation resources, what are the *performance limits* of in-network computation throughput? Namely, what is the *maximum rate* with which computation results can be conveyed to the destination when computations take place in response to a stream of queries for data computation in the network? This question is a fundamental one to resolve in order to efficiently handle the large volume of data analytics requests by optimally utilizing system resources.

A. Motivating Example

Consider a simple query involving three nodes (two sources $1, 2$ and destination d), which form a fully connected undirected graph. Let the node set be $\{1, 2, d\}$, and let the link set be $\{(1, 2), (1, d), (2, d)\}$. Let C_i be the available computation capacity of node $i \in \{1, 2, d\}$, measured in number of processed packets per second and R_{ij} be the available communication bandwidth of link (i, j), in packets/sec. Let x_i, $i = 1, 2$ be a datum of source i, $i = 1, 2$. Consider a stream of queries with rate λ queries/sec where each query seeks to compute, say the sum of a datum of source 1 and a datum of source 2, and deliver the result to d.

If we restrict ourselves to single-path routing, the query stream can be handled in three different ways:

1) Source 1 sends data x_1 to source 2 over link bandwidth R_{12}. This leads to incoming data rate $\min\{\lambda, R_{12}\}$ to source 2. Source 2 performs addition with its own data x_2 (of rate λ) and generates sums $x_1 + x_2$ at rate $\min\{C_2, \lambda, R_{12}\}$. It then sends the sums to the destination d over link bandwidth R_{2d}. Here the computation is performed at node 2, and the rate with which sums are received at d is $\min\{C_2, \lambda, R_{12}, R_{2d}\}$.
2) Source 2 sends data x_2 to source 1 over link bandwidth R_{12}. Source 1 performs addition with its data x_1 (of rate λ), it generates sums $x_1 + x_2$ and sends them to the destination d over link bandwidth R_{1d}. Here, the computation is performed at node 1, and the rate with which sums are received at d is $\min\{C_1, \lambda, R_{12}, R_{1d}\}$.
3) Source 1 sends data x_1 to d over link of bandwidth R_{1d}, and source 2 sends data x_2 to d over link bandwidth R_{2d}. The destination d performs the addition and generates sums $x_1 + x_2$. Thus, the computation is performed at d, and the rate with which the sums are generated is $\min\{C_d, \lambda, R_{1d}, R_{2d}\}$.

Clearly, the maximum rate of received sums depends on which computation node was used, and how data are routed on the network. The problem becomes further complicated if we allow routing through multiple paths. Moreover, considering a stream of similar queries, it is possible to load balance queries over the different options, and hence the problem obtains a multi-commodity form. The static scenario described above serves as a prelude to the dynamic problem that arises in the presence of unknown dynamic query arrivals and accumulated traffic loads at various queues in the network. The decisions in the dynamic scenario concern determination of the node to perform the computation for each query, as well as queue management through traffic routing, link bandwidth sharing and computation capacity allocation, and must be made adaptively.

B. Related Work

The problem of in-network computation has attracted a lot of attention recently. If network coding is allowed, cut-set bounds for the computational capacity of networks defined on Directed Acyclic Graphs (DAGs) are proven in [2]. These cut-set bounds cannot be achieved by routing alone, and proper network codes need to be used. However, in this paper we restrict ourselves to routing-only policies, which simplify adaptivity and distributed implementation. Prior works pertaining to routing-only approaches study the problem in a static setup, with network flows as variables [3]. The problem of finding an optimal flow when there is a computation cost at each node is considered in [4]. Steiner-tree packings are examined in [5] for solving function computation jointly with multicasting, albeit without considering limitations on computation capacity of nodes. A line of work also deals with scaling laws of network computational capacity, cf. [6] and followup papers.

In dynamic setups, [4] examines the problem of function computation in cloud computing and use intuition from the Lagrangian relaxation to derive a dynamic queue-based algorithm. The work in [7] deals with the problem of computing a function of data generated at *all* nodes in a network, a problem that is mainly motivated by sensor network applications. The authors relate the problem to the network broadcast (in the reverse manner) and they propose a scheme based on the *Random Useful Policy* (adaptive broadcast policy [8]) to achieve maximum query rate. On the contrary, in our scenario where data stem (possibly) from a strict subset of the nodes, the corresponding (reverse) multicast method is not successful in general, indicating that the consideration of computation capabilities at all nodes is crucial for such a methodology.

A common underlying assumption, at least implicitly, in the aforementioned works is that there are no limitations on how nodes may process data. Constraints on packet combinations are considered in the literature of *processing networks* which is very related to our work, see e.g. [9] for a recent review. These networks model the industrial assembly of components, whereby the network blueprint determines where the combination of various types of components takes place. Recent works on allocation of resources and utility optimization in processing networks include the work in [10], where the use of "dummy components" is made to get around the processing restrictions, and [11], where the authors advocate minimizing the drift of a suitably perturbed quadratic Lyapunov function. Our problem setup generalizes the processing networks framework in the following manner: instead of combining any two components of the same type (e.g. any bottle with any cork), here each query has a tag and we need to combine pairs of data with the exact same tags.

C. Our Contribution

We study the dynamic resource allocation problem that arises in network computation with the aim to reach the maximum achievable query response rate. We design a universal methodology and algorithm to solve this problem for a broad class of operations on data encountered in practice such as arithmetic, logical, database-related or other types of operations. We abstract the operation as "summation", with the understanding that it stands for any operation of that broad class.

We consider a scenario with two data sources 1 and 2, and a stream of dynamically arriving queries, each of which seeks to compute the sum of a datum from source 1 and a datum from

source 2 and deliver the sum to a destination. The process takes place in a communication network with diverse computation and bandwidth resources. The restriction to two sources and a query stream is deemed necessary for presentation purposes, but it will become apparent that the analysis in the paper can be easily extended to multiple sources and multiple queries via a simple multiclass queueing extension.

We design algorithms that orchestrate utilization of computation and bandwidth resources by performing (*i*) dynamic load balancing of computations on available nodes, (*ii*) unprocessed (raw) and processed data routing from source nodes to computation nodes and from computation nodes to the destination respectively, (*iii*) scheduling of data from different queries on communication links and computation nodes. The proofs of algorithm optimality require non-trivial modifications of the well-known Backpressure (BP) routing and scheduling, including computation thresholding for capturing the tag constraint, randomization for decoupling routing and computation, and the use of stochastic coupling. Our contributions are:

- We formulate the problem of max-throughput distributed computation and derive necessary conditions for queue stability, which correspond to an upper bound on the maximum attainable query rate.
- For the optimal policy, we deploy our approach progressively. For a pre-specified computation node, we first derive the optimal policy under the assumption that the network infrastructures for communicating raw and processed data are non-overlapping. We then relax this assumption and extend our approach. The optimal policy involves *Backpressure* for scheduling and routing, as well as appropriate packet combining at computation nodes, and the optimality is derived through a novel queueing structure abstraction at those nodes.
- We extend the problem in the case of multiple computation nodes where computations need to be load-balanced across the available nodes. The extended optimal policy relies on the *join-the-shortest-sum-of-queues* rule.

II. Model and Problem Statement

A. Network, Resources and Query Streams

We consider a network abstracted as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. We assume there exist two source nodes $s_1, s_2 \in \mathcal{N}$ and a destination node d. Edge $(m, l) \in \mathcal{E}$ between nodes m and l has a fixed capacity of R_{ml} packets per slot. A network example is given in Fig. 1.

We consider a *stream* of queries, where each query concerns the computation of the sum of a datum from source 1 and a datum from source 2, while the network is agnostic to specificities of data [1]. This situation is abstracted as follows. Upon arrival of each query, a corresponding packet (datum) is generated at each of the two source nodes, and both packets are given the same *tag* (e.g. the identification number of the corresponding query). These packets need to be summed somewhere in the network, and the result needs to be delivered to the destination d. Time is slotted, and at each slot t there are $A(t)$ newly arrived queries belonging to the same stream, where the process $A(t)$ is assumed to be independent and identically distributed with time, with $\mathbb{E}[A(t)] = \lambda$.

[1] An extension here is to consider networks that are aware of data specificities and can exploit them to improve performance; e.g. use caching or multicast.

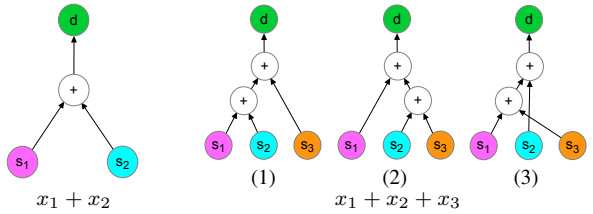


Fig. 2. Computation DAGs. In this paper we focus on a single operation with two sources and a unique computation DAG (shown left).

Combination of packets corresponding to a query may take place in one among a subset of nodes, denoted by $\mathcal{N}_C = \{n_1, n_2, ..., n_{N_C}\} \subseteq \mathcal{V}$; these are referred to as the *computation nodes*. Node $n_i \in \mathcal{N}_C$ has computational capacity of C_{n_i}, measured in number of produced processed packets per slot, where each processed packet represents the sum of two raw packets with the same tag when both are available to the computation node.

B. Operations and Embeddings

For demonstration purposes, our analysis is focused on a simple operation $x_1 + x_2$, but it is useful to discuss a possible generalization. Each computation task is associated with a set of sources whose data are involved in the computation, and the operation to be performed on their data. For instance $x_1 + x_2 + x_3$ describes retrieval of one datum from each of the sources $1, 2, 3$ and their addition. From the network computation point of view however, the description of the operation is completely specified only when we are given the entire order of how data are combined. One way to provide such a description is the so-called *computation graph*, which is a directed acyclic graph (DAG) whose nodes are the sources, the destination, and the operations themselves. The ordering of nodes in this graph gives a description of the operation. Some operations are associated with a unique computation DAG while some others not. For example, the operation $x_1 + x_2$ is associated with a unique DAG with nodes 1, 2 and "+" denoting the summation, see Fig. 2-(left). On the other hand, operation $x_1 + x_2 + x_3$ has more than one computation DAGs, each of which stems from the outcome of the associativity property of the addition operator. In this work, we assume that each task is associated with a *unique* computation DAG.

An embedding of the computation graph on the network graph is a mapping of DAG operation nodes to computation nodes of the network. Prior work has studied the problems of finding an embedding that minimizes delay and cost and has shown that they are both NP-complete problems [12]. In this paper we focus on one query stream that can be computed at N_C nodes, and hence there are $|\mathcal{N}_C|$ possible embeddings of our computation DAG onto G. Instead of finding the best embedding, we use all embeddings available to load-balance

computation so as to achieve the maximum attainable query rate.

We remark that our analysis can be generalized to multiple query streams and multiple computation DAGs by using a multiclass queueing approach, which we omit here for brevity. Moreover, the case we study entails all nontrivialities that arises from integrating routing and computation.

C. Queueing Model

Data packets at each node may be (*i*) unprocessed (raw) source data on their way from the source to the computation node, or (*ii*) processed data on their way from the computation node to the destination. We introduce a packet classification with respect to the computation node that a particular packet will be (or was) computed. The raw packets can be further classified according to the source where they stem from. To capture all packet classes we define the following queues:

- $\mathcal{Q}_k^{(i,n)}(t), i = 1, 2$: Data queue at node k, containing tagged raw packets generated at node s_i that have to undergo computation at node n; $Q_k^{(i,n)}(t)$ denotes its length. Raw packets assigned to n do not wait in the data queues but go directly to the queues keeping packets to be combined at this node (see below). Therefore, $Q_n^{(i,n)}(t) = 0$.
- $\mathcal{X}_n^{(i)}(t), i = 1, 2$: Computation queue at node n, containing tagged raw packets generated at node s_i that have to undergo computation at *this node*; $X_n^{(i)}(t)$ denotes its length.
- $\mathcal{Q}_k^{(0,n)}(t)$: Data queue at k, containing processed packets from computing node n, that have to be delivered at the destination node; $Q_k^{(0,n)}(t)$ denotes its length. We make the convention that $Q_d^{(0,n)}(t) = 0, \forall n \in \mathcal{N}_C$.

Moving packets between queues corresponds to control decisions to be taken each slot as follows:

- The set of raw packets with tags $\mathcal{U}_{mk}^{(i,n)}(t)$ originated from node s_i and destined to computation node n, that are transmitted from node m to node k; $U_{mk}^{(i,n)}(t)$ is the number of packets of this decision. We allow to allocate more service than packets waiting in the queue, in which case "zero" packets are transmitted (these packets will be dropped at the other side of the link).
- The pairs of raw packets to be combined at each computation node n. Let $\mathcal{Z}_n(t)$ be the set of corresponding tags and $Z_n(t)$ be the number of combined packets.
- The set of processed packets, combined at node n, $\mathcal{U}_{mk}^{(0,n)}(t)$, to be transmitted from node m to node k; $U_{mk}^{(0,n)}(t)$ is the number of such packets.

We have the following constraints. The total number of transmitted packets over a link are limited by link capacity

$$\sum_{\substack{i\in\{0,1,2\},\\ n\in\mathcal{N}_C}} \left(U_{ml}^{(i,n)}(t) + U_{lm}^{(i,n)}(t)\right) \leq R_{ml}, \quad \forall (ml) \in \mathcal{E}. \quad (1)$$

Further, the number of combined pairs cannot exceed the computation capacity or any of the individual raw packet queue lengths,

$$0 \leq Z_n(t) \leq \min\left[C_n, X_n^{(1)}(t), X_n^{(2)}(t)\right], \quad \forall n \in \mathcal{N}_C\,. \quad (2)$$

Moreover, two packets with the same tag can be combined only if they have both arrived at the computation node, i.e.,

$$\mathcal{Z}_n(t) \subseteq \mathcal{X}_n^{(1)}(t) \cap \mathcal{X}_n^{(2)}(t), \quad \forall n \in \mathcal{N}_C. \quad (3)$$

We point out that the last constraint involves consideration of packet tags and would complicate the description of the system state. However, our approach will be to define a simpler system state with queue lengths only, and then establish that the considered policies indeed satisfy (3).

We define the set of permissible policies in our system Π_C as mappings of the network state (queue lengths) to control variables for routing $U_{ml}^{(i,n)}(t)$ and computation $Z_n(t)$, subject to capacity and computation constraints (1)-(3).

D. Problem Formulation

We say that *the system is stable* under a policy π if all queues in the system are strongly stable, i.e. if

$$\limsup_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left\{Q_k^{(i,n)}(t)\right\} < \infty, \ \forall i \in \{0,1,2\}, \forall k \in \mathcal{N}$$

$$\limsup_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left\{X_n^{(i)}(t)\right\} < \infty, \quad \forall i \in \{1,2\}, \ \forall n \in \mathcal{N}_C.$$

We are interested to find the maximum attainable query rate, λ^* that can be delivered by some policy in the class Π_C subject to system stability, as well as to find a policy $\pi^* \in \Pi_C$ that stabilizes the system for *every* query rate $\lambda < \lambda^*$.

It is important to note that strong (or at least steady-state) stability of all queues is actually necessary to ensure that all computations are made and results are delivered to the destination. Indeed, if some queues are only mean-rate- or rate-stable (these are weaker notions of stability), there may be a growing number of queries in time that are never executed.

III. Fixed Computation Node

We begin our analysis by fixing attention to one computation node, say node n. This special case contains the crux of the problem, which is to deal with (*i*) the challenging constraint (3), and (*ii*) integration routing and computation.

A. Query Rate Upper Bound λ^*

First we revisit the standard multicommodity flow problem. For a set of commodities $\mathcal{C}$, consider the multicommodity flow feasibility region $\Lambda_{\mathcal{G}}(\mathcal{C})$ of network G which is defined as the set of arrival rate vectors $(\lambda^{(c)})_{c\in\mathcal{C}}$ for which *there exists a feasible flow that successfully decomposes the arrivals*. Feasibility in this case includes, (*i*) flow conservation constraints $\forall c \in \mathcal{C}$

$$\sum_{k\in\mathrm{OUT}(m)} f_{mk}^{(c)} - \sum_{k\in\mathrm{IN}(m)} f_{km}^{(c)} = \begin{cases} \lambda^{(c)} & m = \text{src.} \\ -\lambda^{(c)} & m = \text{dest.} \\ 0 & \text{otherwise,} \end{cases} \quad (4)$$

where $\mathrm{OUT}(m)$ are the out-neighbours of node m and $\mathrm{IN}(m)$ its in-neighbours, (*ii*) capacity constraints,

$$\sum_{c\in\mathcal{C}} f_{mk}^{(c)} \leq R_{mk}, \quad \forall (m,k) \in \mathcal{E}, \quad (5)$$

and (*iii*) standard flow constraints,

$$\begin{cases} f_{mk}^{(c)} = 0 & m = \text{dest.} \\ f_{mk}^{(c)} \geq 0 & \text{otherwise} \end{cases} \quad \forall (m,k) \in \mathcal{E},\ \forall c \in \mathcal{C}. \qquad (6)$$

The feasibility region, given by the convex polytope

$$\Lambda_{\mathcal{G}}(\mathcal{C}) = \{(\lambda^{(c)})_{c\in\mathcal{C}} \text{ s.t. } (4)-(6) \text{ hold}\}$$

is also the set of arrival rates for which the system with dynamic routing policies (for the same network and commodity setting) is stable, under mild assumptions for arrival processes [13].

Consider now the standard multicommodity routing problem with three commodities $\mathcal{C}_3 = \{(s_1,n),(s_2,n),(n,d)\}$, and corresponding feasibility region $\Lambda_{\mathcal{G}}(\mathcal{C}_3)$. We have:

Theorem 1. *For a query stream from sources s_1, s_2, destined to d and computed at node n, the following are necessary conditions for system stability:*

$$(\lambda,\lambda,\lambda) \in \Lambda_{\mathcal{G}}(\mathcal{C}_3), \quad \lambda \leq C_n.$$

Proof: Recall that the policy set Π_C, whose stability region we are interested in bounding, involves the challenging constraint (3). To obtain the upper bound on the performance of this class, we relax this constraint in the following way. Consider a new set of control policies Π, whereby we select routing variables $U_{ml}^{(i,n)}(t)$ subject to instantaneous capacity constraint R_{ml}, and computation variable $Z_n(t)$ subject to capacity C_n, while constraint (3) is relaxed and any two raw packets can be combined. Note that policies in set $\Pi \setminus \Pi_C$ may interleave raw data from different queries, which harms the system.

The conditions in the statement of the theorem are necessary for stability for any policy $\pi \in \Pi$; the first condition is necessary for routing all raw packets from the sources to n and the combined packets from n to the destination and the second is necessary for performing all computations. Then the proof is complete by noting that if $\pi \in \Pi_C$ then $\pi \in \Pi$ as well, due to the fact that Π is obtained from Π_C by relaxing a constraint; hence it also includes policies that satisfy the constraint. ■

Equivalently, for a stable system the query rate is upper bounded by

$$\lambda^* = \max_{\substack{(\lambda,\lambda,\lambda)\in\Lambda_{\mathcal{G}}(\mathcal{C}_3) \\ \lambda\leq C_n.}} \lambda.$$

In the next subsections we propose policies in Π_C that provably stabilize any $\lambda < \lambda^*$, thus establishing that in fact λ^* is the maximum query rate.

B. Case I. Non-overlapping topology from source to computation nodes and from computation nodes to destination

We examine first the special case where the network connecting the sources to the computation node is non-overlapping with that connecting the computation node to the destination, see Fig. 3. When this is true, the networks containing s_1, s_2, n and n, d do not have common nodes (except n) and hence the routing is decoupled from computation, and the only remaining nontrivial issue is the challenging constraint (3).

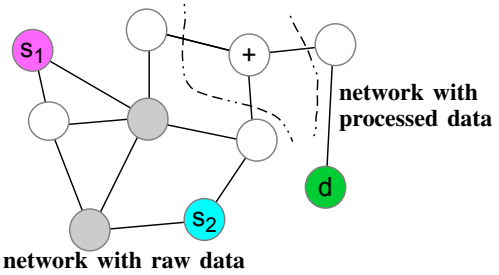


Fig. 3. Example where the networks of raw and combined data are non-overlapping.

We first consider the case where the destination is also the node that performs the computations. The results will be generalized in a straightforward way at the end of the section. Consider the following policy π_1:

- The controls $U_{mk}^{(i,n)}(t)$ are obtained by applying *BP routing and scheduling*; see the text box and [14], [15] for details.
- At node n, let $P_n(t)$ denote the number of packet tags that satisfy (3). Combine the maximum number of available paired packets i.e. $\min\{P_n(t), C_n\}$ pairs.

Review on Backpressure (BP) Routing and Scheduling [14]: BP is a dynamic algorithm that reacts on current data queue backlogs and decides on the number of packets to be routed across each link in order to balance queues. For wired multicommodity routing, the BP algorithm to be executed at each link/slot is:
Select the commodities that maximize differential backlog:

$$c_{mk}^*(t) = \arg\max_{c\in\mathcal{C}} \left| Q_m^{(c)}(t) - Q_k^{(c)}(t) \right|, \ \forall (m,k) \in \mathcal{E}$$

Route any $U_{ij}^{(c)}(t)$ packets from commodity c, where:

$$U_{mk}^{(c)}(t) = \begin{cases} R_{mk} & \text{if } c = c_{mk}^*(t) \text{ and } Q_m^{(c)}(t) > Q_k^{(c)}(t) \\ 0 & \text{otherwise.} \end{cases}$$

BP is a maximum-throughput policy for wired multicommodity routing.

The main result of this section follows:

Theorem 2. *For non-overlapping networks and a query stream computed at any one node n, policy π_1 is stable for any query rate $\lambda < \lambda^*$.*

We prove the result by tackling constraint (3). To avoid tracking packet tags, define $X(t)$ as the number of raw packets in the system excluding node n, and $S_n^{(1)}, S_n^{(2)}$ as the number of raw packets with unique tags at node n, whose counterpart is still in $X(t)$. Recall that $P_n(t)$ is the number of raw pairs with same tag waiting at node n. Breaking down the packets at node n we have

$$\begin{aligned} X_n^{(1)}(t) + X_n^{(2)}(t) &= 2P_n(t) + S_n^{(1)}(t) + S_n^{(2)}(t) \\ &\leq 2P_n(t) + X(t) \end{aligned}$$

where inequality becomes strict equality if all packets $X(t)$ have different tags. Thus the number of pairs waiting at n to

be computed is lower-bounded by

$$P_n(t) \geq \frac{X_n^{(1)}(t) + X_n^{(2)}(t) - X(t)}{2},$$

from which we observe that, if

$$X_n^{(1)}(t) + X_n^{(2)}(t) \geq 2C_n + X(t)\,, \tag{7}$$

then there are at least C_n pairs of packets with the same tag in the computation node. Hence we may directly replace constraint (3) with the restriction (7). The idea behind the proof is to use a slightly different restriction from (7), where, instead of using $X(t)$ (which is correlated with network events), we make use of a large constant $\overline{X}$.

Consider a policy π_1' that works as follows:

- The controls $U_{mn}^{(i,n)}(t)$ are obtained by applying backpressure routing and scheduling, as in π_1.
- At node n: choose the computations as follows:

$$Z_n(t) = \begin{cases} C_n & \text{if } X_n^{(1)}(t) + X_n^{(2)}(t) \geq 2C_n + \overline{X} \\ 0 & \text{otherwise} \end{cases}$$

The main idea is to prove the statement of Theorem 2 for policy π_1' and then prove the theorem itself by showing that π_1' is stochastically dominated by π_1. First, we prove that π_1' satisfies the largest possible query rate:

Lemma 1. *For any $\lambda < \lambda^*$, there exists a threshold $\overline{X}$ that depends on the parameters $\mathcal{G}, C_n, \lambda$, such that the network is stable under policy π_1'.*

Proof: We give a sketch of the proof. The complete version, as well as all the complete versions of other proofs, is in [16]. Since BP routing stabilizes the network queues, and leads to a stationary distribution for their lengths, we pick a large enough value $\overline{X}$ so that the probability that $X(t) > \overline{X}$ is made sufficiently small. Then we consider the T-slot drift for the computation queues and show that they are also stable, using the fact that the restriction (7) is violated only very rarely. ∎

Policy π_1' achieves the computational capacity of the network but has some shortcomings. First, the computation of an appropriate threshold $\overline{X}$ can be tedious and requires the statistics of the query arrival processes. Second, policy π_1' adds delays in delivering the result to destination, since due to large $\overline{X}$ used, the computing node does not perform any computations until many packets have arrived. Both issues above are resolved by policy π_1, which does at least as good as π_1' in terms of stability, as implied by the following result:

Lemma 2. *For all thresholds $\overline{X}$, we have*

$$X_n^{(i),\pi_1}(t) \leq_{st} X_n^{(i),\pi_1'}(t), \quad \forall i \in \{1,2\}.$$

Proof: We compare what happens when both policies start with the same network state at time $t = 0$ and the sample paths of routing decisions and arrival processes are the same. The sample paths of the computations made, however, are different between the two policies. Let t' the time slot index where the constraint for π_1' to be active holds for the first time. The main point is that, in the meantime, policy π_1 can serve packets, therefore all packets that will get combined by π_1' at t' will have been combined earlier, or at t' if its the first time pairs of packets appear. That is, the C_n packets that are combined by π_1' have either been combined already by π_1 if some of these were paired earlier than t', or they have been combined by π_1 if all of these are paired on t', so $X_n^{(i),\pi_1}(t') \leq X_n^{(i),\pi_1'}(t')$. In addition, since for $t < t'$ no packets are served by π_1', while maybe served by π_1, so $X_n^{(i),\pi_1}(t) \leq X_n^{(i),\pi_1'}(t), \forall t < t'$. We can show, using similar arguments that this inequality holds also for $t > t'$.

Since this is proven for every sample path for arrivals and routing decisions, the statement in the Lemma follows. ∎

Finally, the proof of Theorem 2 follows from the above lemmas.

C. Case II. Overlapping topology from source to computation nodes and from computation nodes to destination

Next, we allow processed packets to compete with raw packets for the same network resources. The arising technical issue here is the dependence of network state on the outputs of the computation node, which means that certain steps used to prove Lemma 1 can not be used directly.

In order to overcome this technicality, we randomize the output of the computation node by purposefully inserting dummy packets at random. In particular, we introduce an additional queue $\mathcal{Y}_n(t)$ in the computing node that keeps the results of the computation and forwards $F_n(t)$ processed packets to the queue $\mathcal{Q}_n^{(0,n)}(t)$, where $F_n(t)$ includes both useful and dummy packets. The network treats these dummy packets as real ones, delivering them to the destination (Fig. 4).

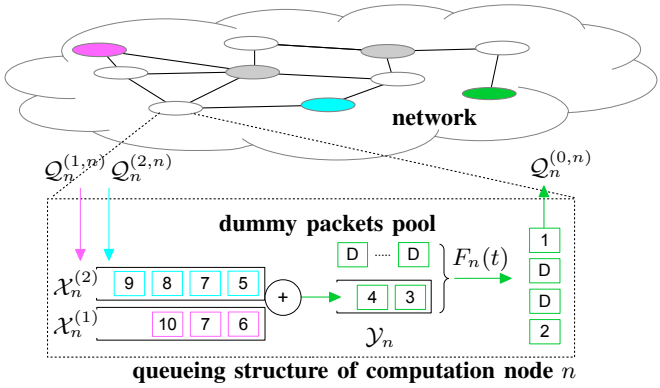


Fig. 4. Illustration of queueing structure for computation node n. Numbered packets are either raw (purple and blue) or processed (green) useful packets, while packets noted with "D" are dummy packets. At slot t, $F_n(t)$ packets depart the queue $\mathcal{Y}_n$ and arrive at the network queue $\mathcal{Q}_n^{(0,n)}$, where $F_n(t)$ potentially includes both dummy and useful packets.

The structure above leads to a situation where the number of packets going into $\mathcal{Q}_n^{(0,n)}(t)$ does not depend on the network queues, which facilitates the analysis. To perform the analysis, however, we additionally need to design carefully the number of generated dummy packets to ensure that the adjusted packet rate remains inside region $\Lambda_{\mathcal{G}}(\mathcal{C}_3)$, and that the virtual queue $\mathcal{Y}(t)$

is served enough to be strongly stable. These two properties then imply that the useful processed packets will eventually be delivered.

We will consider the policy π_2. At each time slot t:

- Controls $U_{mk}^{(i,n)}(t)$ are obtained by applying backpressure routing and scheduling.
- At node n, all paired packets that can be combined (i.e. at most C_n pairs) are combined and pushed to the queue $\mathcal{Y}_n(t)$.
- $F_n(t)$ packets are pushed from the queue $\mathcal{Y}_n(t)$ to the queue $\mathcal{Q}_n^{(0,n)}(t)$. $F_n(t)$ is a random variable with mean $\lambda' \in (\lambda, \lambda^*)$. If there are not enough packets, then dummy packets are used. These dummy packets are routed exactly as the normal processed packets.

For the process that serves queue $\mathcal{Y}_n(t)$ we use

$$F_n(t) = (1 + B^{(n)}(t))A(t), \tag{8}$$

where $B^{(n)}(t)$ is a Bernoulli random variable, independent of everything in the network, with success probability ϵ_B; this parameter can take an arbitrarily small positive value.

The main result here is that the policy described above achieves a computation rate arbitrarily close to the upper bound:

Theorem 3. *For one query stream computed at n, policy π_2 is stable for any query rate $\lambda < \left(1 - \frac{\epsilon_B}{1+\epsilon_B}\right)\lambda^*$.*

This theorem can be proved in a similar way as Theorem 2, by defining a policy π_2' that is the same as π_2, except that it does computations only if $X_n^{(1)}(t) + X_n^{(2)}(t) \geq 2C_n + \overline{X}$. Similarly to the previous subsection, we first prove stability under π_2':

Lemma 3. *For any $\lambda < \left(1 - \frac{\epsilon_B}{1+\epsilon_B}\right)\lambda^*$, there is a threshold $\overline{X}$ such that the the network is stable under policy π_2'.*

Proof Outline: The crucial observation is that, due to the randomized input to queue $\mathcal{Q}_n^{(0,n)}(t)$ and the use of dummy packets, $\left(Q_k^{(0,n)}(t)\right)_{k\in\mathcal{N}}$ do not depend on the state of the queues with raw packets. For the network excluding node n, then, it is as if we had three commodities: $s_1 \to n, s_2 \to n, n \to d$ with rates $(\lambda, \lambda, (1+\epsilon_B)\lambda)$ inside the stability region of the system and correlated arrival processes, but i.i.d. in time. This implies that $Q_k^{(i,n)}(t), \forall i \in 0, 1, 2, \forall k \in \mathcal{V} \setminus \{n\}$ are strongly stable under π_2' (since backpressure routing is applied).

In addition, strong stability implies steady-state stability of the aforementioned queues, so $X(t)$ has a steady-state distribution with zero limit as it goes to infinity. We can then apply the same methodology as in the proof of Theorem 2 to show that there exists a threshold $\overline{X}$ for which $\mathcal{X}_n^{(i)}(t), i \in \{1, 2\}$ are strongly stable.

The final step is to show that queue $Y_n(t)$ is stable as well. Indeed, since $\mathcal{Q}_n^{(i)}(t), i \in \{1, 2\}$ are stable, the input to this queue is a Markov modulated process with mean λ. Since the service process of $\mathcal{Y}_n(t)$ is i.i.d. over time with mean $\left(1 - \frac{\epsilon_B}{1+\epsilon_B}\right)\lambda^* > \lambda$, strong stability of $\mathcal{Y}_n(t)$ follows. ∎

Then, we prove that the threshold is not really needed for the algorithm, in the same way as Lemma 2.

Lemma 4. *For all thresholds $\overline{X}$, $X_n^{(i),\pi_2}(t) \leq_{st} X_n^{(i),\pi_2'}(t), \forall i \in \{1, 2\}$*

Theorem 3 is then a consequence of Lemmas 3, 4 and the observation that the number of packets (including dummy ones) injected in queue $\mathcal{Q}_n^{(0,n)}(t)$ at every time slot t is the same for both policies π_2, π_2'. As a final remark, note that policy π_2 can achieve any fraction of the computational capacity upper bound of Theorem 1 by requiring only access to the number of requests at every slot. In fact it can be proven that even delayed information about the number of requests is sufficient for this policy to achieve any fraction of the computational capacity of the network [13, §4.7].

In summary, we have shown that for one stream with two sources and one destination, where the streams need to be combined at specified node n, the maximum query rate is characterized by Theorem 1, and achieved in a distributed way by policy π_2. In the next section we generalize to multiple computation nodes.

IV. Multiple Computation Nodes

In this section, we consider the more general case where a summation can take place at any computation node in $\mathcal{N}_C = \{n_1, \ldots, n_{N_C}\}$.

A. Query Rate Upper Bound λ^*

In this scenario, we have three commodities for every one of the N_C computation nodes, hence we need to define the set $\widetilde{\mathcal{C}}_3$ with $3N_C$ unicast commodities, as follows: there are three commodities for each computation node $n \in \mathcal{N}_C$, $(1, n)$ delivering packets from s_1 to n, $(2, n)$ delivering packets from s_2 to n, and (n, d) delivering packets from n to d. Consider the multicommodity flow with rates $\boldsymbol{\lambda} = (\lambda_1^1, \lambda_1^2, \lambda_1^0, \ldots, \lambda_{N_C}^1, \lambda_{N_C}^2, \lambda_{N_C}^0)$, where $\lambda_m^1 = \lambda_m^2 = \lambda_m^0 = \lambda_m$, and

$$\sum_{m\in\mathcal{N}_C} \lambda^m/\lambda = 1$$

i.e., the quantities $(\frac{\lambda_m}{\lambda})_{m\in\mathcal{N}_C}$ can be seen as the time-share coefficient for queries computed at node m. Then we have the following upper bound on the query rate:

Theorem 4. *For a query stream with sources s_1, s_2, destined to d and computed at the set of computation nodes $\mathcal{N}_C$, the following is a necessary condition for stability:*

$$\boldsymbol{\lambda} \in \Lambda_{\mathcal{G}}(\widetilde{\mathcal{C}}_3), \quad \lambda_m \leq C_m, \ \forall m \in \mathcal{N}_C, \quad \sum_{m\in\mathcal{N}_C} \lambda_m = \lambda.$$

The upper bound characterized by Theorem 4 can be actually achieved arbitrarily close by a dynamic policy, as discussed in the next subsection.

B. Achieving maximum sustainable query rate with Multiple Computation Nodes

In addition to the queues specified in II-C, we need to define N_C other queues, denoted with $H_n(t)$, whose role is to ensure that each computing node does not receive more computational load than its capacity. Queues $H_n(t)$ evolve as

$$H_n(t+1) = \left[H_n(t) + \tilde{A}^{(n)}(t) - C_n\right]^+,$$

where $\tilde{A}^{(n)}(t)$ will be defined in eq. (10). The dynamic policy π_3 we consider here is the following:

- **Load Balancing:** At each slot, choose $n^*(t)$ equal to

$$\arg\min_{n\in\mathcal{N}_C}\left[(1+\epsilon_B)Q_n^{(0,n)}(t)+\sum_{i=1,2}Q_i^{(i,n)}(t)+H_n(t)\right] \tag{9}$$

  where $\epsilon_B \in (0,1)$ is a control parameter. The first term reflects congestion of combined and dummy packets at the computation node, the second term stands for congestion of raw data packets at the source nodes, and the last term reflects the computation load at node n.Then the newly arrived queries are assigned to the class that corresponds to this computation node,

$$\tilde{A}^{(n)}(t)=\begin{cases}A(t), n=n^*(t)\\ 0, \text{otherwise.}\end{cases} \tag{10}$$

- **Routing and scheduling:** Use BP over class pairs. For every link $(m,k)\in\mathcal{E}$ choose the class pair

$$(i^*_{mk}(t),n^*_{mk}(t))=\arg\max_{\substack{i\in\{0,1,2\}\\ n\in\mathcal{N}_C}}\left|Q_m^{(i,n)}(t)-Q_k^{(i,n)}(t)\right|,$$

  where $i^*_{mk}(t)$ is the best class of packets between raw packets and processed packets, and $n^*_{mk}(t)$ is the best class of packets w.r.t. the computation node. Then choose the routing variables as,

$$U_{mk}^{(i,n)}(t)=\begin{cases}R_{mk} & \text{if} \quad (i,n)=(i^*(t),n^*(t))\\ 0 & \text{otherwise.}\end{cases}$$

- **Computation:** At every node $n\in\mathcal{N}_C$, all possible computations are done. If there are more pairs than the computation capacity of this node, then C_n pairs are selected using any tie breaking rule (e.g. priority can be given to the oldest queries).
- **Randomization with dummy packets:** $F_n(t)=\tilde{A}^{(n)}(t)\left(1+B^{(n)}(t)\right)$ packets resulting from a computation are pushed to queue $\mathcal{Q}_n^{(0)}(t)$, where $B^{(n)}(t)$ are an i.i.d. Bernoulli random variables with mean ϵ_B. If there are not enough processed packets available at queue $\mathcal{Y}_n$, dummy packets are used.

The main result is that the policy above satisfies almost every query demand rate below λ^*, according to the choice of the control parameter ϵ_B:

Theorem 5. *Policy π_3 stabilizes the network with multiple computing nodes for any query rate $\lambda<\left(1-\frac{\epsilon_B}{1+\epsilon_B}\right)\lambda^*$.*

Next we provide a sketch of the proof of Theorem 5. By exploiting the randomization that decouples routing from computation, we use classical Lyapunov drift techniques [15], [13, Theorem 4.5] to prove strong stability of network queues, excluding the ones that take part in the computation. In particular we obtain the following bound in the Lyapunov drift expression:

$$\begin{aligned}\Delta L(\mathbf{Q}(t),\mathbf{H}(t))\le B-\sum_{n\in\mathcal{N}_C}C_nH_n(t)+\sum_{n\in\mathcal{N}_C}\mathbb{E}\left\{\tilde{A}^{(n)}(t)\right\}\\ \times\left((1+\epsilon_B)Q_n^{(0,n)}(t)+\sum_{i=1,2}Q_i^{(i,n)}(t)+H_n(t)\right)\\ -\sum_{\substack{(m,k)\in\mathcal{E}\\ n\in\mathcal{N}_C\\ i=0,1,2}}\left(Q_m^{(i,n)}(t)-Q_k^{(i,n)}(t)\right)\left(\mathbb{E}\left\{U_{mk}^{(i,n)}(t)-U_{km}^{(i,n)}(t)\right\}\right)\end{aligned}$$

where we observe that the right-hand side above is minimized by the load balancing, routing and scheduling actions of our policy π_3. Combining the expressions above with an optimal randomized routing policy, we have

Lemma 5. *Under policy π_3, all queues $Q_k^{(i,n)}(t)$, $H_n(t)$, $\forall k,i,n$ are strongly stable for any $\lambda<\left(1-\frac{\epsilon_B}{1+\epsilon_B}\right)\lambda^*$.*

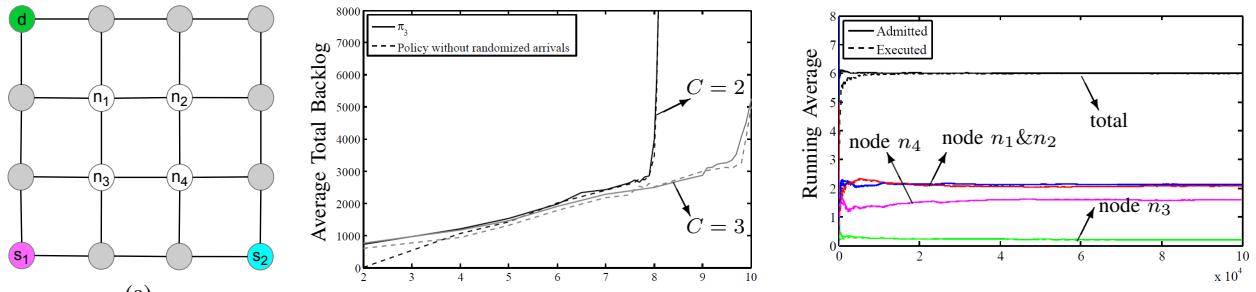
To complete the proof of Theorem 5, it remains to show strong stability for queues $X_n^{(i)}(t), Y_n(t)$. For this we extend the methodology of the previous section in a straightforward manner, e.g. Lemmas 3, 4.

C. Discussion on π_3

The proposed policy π_3 is adaptive, it requires only local information for routing, and it can react to changes in the environment, providing robust efficient network computations. Using prior work [13], we may extend π_3 to the wireless case, covering applications within the range of edge and fog computing. Also note that the thresholds are used only for the proofs and are not necessary in the implementations of policies.

Load balancing queries on different computation nodes requires some coordination, since there an agreement needed to be made and communicated between remote sources on the exact computation node that each query is using. This can be achieved by using an information exchange mechanism like [17]. To facilitate timely coordination, it is possible to modify load balancing in the following way. Sources agree on a weighted round robin policy on how queries are assigned to computation nodes, and then frequently update weights in a coordinated fashion in order to balance the terms in π_3.

The main idea of the algorithms used is the intermediate queues $\mathcal{Y}_n(t)$ whose services are independent of the queues and routing controls in the network. This decouples the problem in a computation one (on which packets to be combined at the nodes), a "load balancing" one (on which embedding to use for new requests) and a communication (on routing and scheduling at links). The cost is using dummy packets in the network and being able to satisfy slightly smaller request rate than the maximum. A policy $\bar{\pi}_3$ that does not use the intermediate queues and does any computations possible at every slot is conjectured to achieve any $\lambda<\lambda^*$.



(a) (b) (c)

Fig. 5. (a) The grid topology used for simulations. (b) Average total queue lengths for π_3 (solid lines), $\bar{\pi}_3$(dashed lines) vs. query rate for $C = 2, C = 3$.(c) Running averages of the query rate, allocations among embeddings and total computations made for a simulation run of π_3 with $C = 2, \lambda = 6$.

V. Simulation Results

In order to illustrate the impact of computational capacity on performance and the network behaviour under our algorithms, we examine a 4×4 grid topology with four computation nodes as shown in Fig. 5(a).

Each edge has a capacity of $R = 5$ packets per slot and computation nodes have capacity, C. The average total queue lengths versus the incoming query rate for policies π_3 with $\epsilon_B = 0.01$ and a policy $\bar{pi}_3$ that does not use the randomized inputs are plotted in Fig. 5 (b) for $C = 2, C = 3$. We can observe that the two policies achieve the same computational capacity. This supports our conjecture that the randomized inputs to $\mathcal{Q}_n^{(0,n)}$ are needed to overcome technicalities the proofs, but an algorithm that does not use them work as well. In addition, $\bar{\pi}_3$ has fewer packets in the network in light loads. We can also note that when $C = 2$, the computational capacity of the network is $\lambda^* = 8$, therefore limited by the capacity of the computing nodes, while for $C = 3$ it is around 9.8, therefore limited by the communication capacity of the network.

Finally, Fig. 5(c) shows the empirical averages of the queries arriving to the system, computations allocated per computing node (i.e. on the load balancing phase of the algorithm) and computations executed. For this simulation, $C = 2$ and $\lambda = 6$, so it is an achievable computation rate by the network. We can observe that as time passes, the average computations made in the network matches the average query demand.

VI. Conclusions

This paper is a first step towards understanding the performance limits of the interaction of Big Data with the underlying network resources. The Big Data challenge materializes as one of dealing with so-called 5 Vs (volume, variety, velocity, variability, complexity). We attempted here to deep-dive into two of the five Vs, namely volume and velocity of data that stems from query streams. Our goal is to characterize the fundamental limits of the volume of queries that can be processed in the presence of limited resources in a networked setting. The study also aims to provide an understanding of the velocity dimension above i.e. how fast the generated volume of data can be processed.

There exist several directions for future work. A non-trivial extension of our work includes the case of computation queries with multiple possible computation graphs (DAGs) to choose from, where each computation graph may have several embeddings in the network graph. Another interesting twist is the scenario where the computation graph involves several types of operations, some of which are harder to perform than others. In that case, the computations at computation nodes would also depend on the type of operation to be performed.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. MCC*, 2012.

[2] R. Appuswamy, M. Franceschetti, N. Karamchandani, and K. Zeger, "Network coding for computing: Cut-set bounds," *IEEE Trans. Inf. Theory*, pp. 1015–1030, 2011.

[3] V. Shah, B. Dey, and D. Manjunath, "Network flows for functions," in *Proc. ISIT*, 2011.

[4] J. Liu, C. H. Xia, N. B. Shroff, and X. Zhang, "On distributed computation rate optimization for deploying cloud computing programming frameworks," *Proc. ACM SIGMETRICS*, 2013.

[5] S. Kannan and P. Viswanath, "Multi-session function computation and multicasting in undirected graphs," *IEEE J. Sel. Areas. Commun.*, pp. 702–713, 2013.

[6] A. Giridhar and P. Kumar, "Computing and communicating functions over sensor networks," *IEEE J. Sel. Areas. Commun.*, pp. 755–764, 2005.

[7] S. Banerjee, P. Gupta, and S. Shakkottai, "Towards a queueing-based framework for in-network function computation," *Queueing Syst.*, pp. 219–250, 2012.

[8] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez, "Randomized decentralized broadcasting algorithms," in *Proc. IEEE INFOCOM*, 2007.

[9] H. C. 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. ACM SIGMETRICS*, 2010.

[10] L. Jiang and J. Walrand, "Stable and utility-maximizing scheduling for stochastic processing networks," in *Proc. Allerton*, 2009.

[11] L. Huang and M. J. Neely, "Utility optimal scheduling in processing networks," *Perform. Evaluation*, pp. 1002–1021, 2011.

[12] P. Vyavahare, N. Limaye, and D. Manjunath, "Optimal embedding of functions for in-network computation: Complexity analysis and algorithms," *CoRR*, vol. abs/1401.2518, 2014.

[13] M. J. N. L. Georgiadis and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Foundations and Trends in Networking*, vol. 1, 2006.

[14] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Trans. Autom. Control*, pp. 1936–1948, 1992.

[15] M. Neely, E. Modiano, and C. Rohrs, "Dynamic power allocation and routing for time-varying wireless networks," *IEEE J. Sel. Areas. Commun.*, pp. 89–103, 2005.

[16] A. Destounis, G. S. Paschos, and I. Koutsopoulos, "Streaming Big Data meets Backpressure in Distributed Network Computation," Tech. Rep., 2016. [Online]. Available: http://arxiv.org/abs/1601.03876

[17] D. G. Thaler and C. V. Ravishankar, "Using name-based mappings to increase hit rates," *IEEE/ACM Trans. Netw.*, pp. 1–14, 1998.