

Distributed Routing in Networks with Time-Variant Edge Capacities: A Modified
Multicommodity Flow Approach

A Thesis

Presented to
the faculty of the School of Engineering and Applied Science
University of Virginia

in partial fulfillment
of the requirements for the degree

Master of Science

by


Kelli Lafferty

August

2013

APPROVAL SHEET

The thesis
is submitted in partial fulfillment of the requirements
for the degree of
Master of Science


AUTHOR

The thesis has been read and approved by the examining committee:

Randy Cogill

Advisor

William T. Scherer

Peter Beling

Accepted for the School of Engineering and Applied Science:



Dean, School of Engineering and Applied Science

August
2013

Abstract

This research is focused on creating an efficient and effective network flow framework for applications in routing protocols for mobile, unicast communications networks with limited delay tolerance in urban canyons. Urban canyons are environments in which man-made objects interfere with the dependability of network reception. To account for fluctuations in reception, networks in this type of environment are modeled with time-variant edge capacities. To limit delays, algorithms implemented for these types of networks must be distributed. Specifically, we focus on modifying multicommodity flow problems to find routing protocols that can (1) effectively send flow through a network with time-varying edge capacities and (2) operate in a distributed setting. To achieve this, we developed candidate algorithms by both manipulating existing fully polynomial time approximation schemes for multicommodity flow for a time-varying setting and developing new algorithms that can produce a static, robust invariant allocation of flow. These candidate algorithms were initially evaluated through simulations in MATLAB. Then, promising algorithms were assessed through a theoretical analysis that validated performance and suitability observations made during the MATLAB simulations. Finally, a distributed implementation plan was developed.

Acknowledgements

I would like to thank Professor Randy Cogill for his guidance in the development and execution of this work. Additionally, I would like to thank Professor Bill Scherer for his invaluable advice, without which I could not have moved forward in my graduate career, Professor Peter Beling for his suggestions on this thesis work, and Jennifer Mauller for her perpetual logistical support. I am forever indebted to my friends and colleagues for sharing in the laughter and grief that comes with graduate life. I would like to especially note Mark Paddrik, who assisted with the formatting and organization of both my proposal and thesis documents. Finally, I would like to thank David for his unwavering support and patience while I finished this degree.

Contents

1	Introduction	1
1.1	Motivation: Mobile Satellite-Terrestrial Networks in Urban Canyons	1
1.2	Problem Statement	3
1.2.1	Project Goals	4
1.2.2	Work Activities	5
2	Literature Review	6
3	Research	8
3.1	Background	8
3.1.1	Multicommodity Flow	8
3.1.2	Fully Polynomial Time Approximation Algorithms . . .	10
3.1.3	Invariant Allocation	13
3.2	Sampling Algorithm	17
3.2.1	Concept	17
3.2.2	Execution	17
3.2.3	Performance: Speed and Accuracy	18
3.3	Lower Bound Approximation	20
3.3.1	Concept	20
3.3.2	Execution: Logarithmic Bound	22
3.3.3	Execution: "Simple" Network Bound	23

3.3.4	Suitability: Maximum Multicommodity Flow vs Maximum Concurrent Flow	25
3.3.5	Performance: Logarithmic Limitations	26
3.4	Stochastic Supergradient Method	28
3.4.1	Concept	28
3.4.2	Execution	29
3.4.3	Suitability: Sampling from a Markov Process	32
3.4.4	Performance: Speed as Compared to FPTAP	37
3.5	Distributed Implementation	38
3.5.1	Concept	38
3.5.2	Execution	38
3.5.3	Suitability: Information Exchange Requirements	47
4	Results	49
4.1	General Approach	49
4.2	Fully Polynomial Time Approximation Algorithms	54
4.3	Sampling Algorithm	55
4.4	Lower Bound Approximation	59
4.4.1	Logarithmic Bound	60
4.4.2	”Simple” Network Bound	61
4.5	Stochastic Supergradient Method	63
5	Conclusions	67
5.1	Contribution	70

5.2	Future Work	70
-----	-----------------------	----

1 Introduction

As society begins to rely more and more on mobile technology as a primary conduit for collecting and relaying information, applications have emerged for mobile devices to be used as a means for improvements in communications, tracking, and monitoring for a myriad of systems. However, mobility presents challenges to information sharing: connectivity changes as nodes move, mobile devices have limited internal capacity, and wireless signals can have limited or unpredictable bandwidth.

Thus, routing protocols in such networks must fulfill unique needs that traditional routing protocols currently cannot handle. We provide mathematical frameworks from which routing protocols can be developed for particular mobile networks.

1.1 Motivation: Mobile Satellite-Terrestrial Networks in Urban Canyons

Our inspiration for developing frameworks for routing protocols in mobile networks comes from a mobile satellite terrestrial network operating in an urban canyon.

Satellite-terrestrial networks are networks in which transmissions are exchanged between satellites and terrestrial nodes [22]. Terrestrial nodes within a certain ground range are also connected and capable of exchanging information when a direct link to a satellite is unavailable. All links between

nodes have a limited transmission rate. Transmissions are unicast, that is, transmissions occur between specific source and sink nodes. Furthermore, these unicast transmissions may occur concurrently, creating the potential for congestion. In the specific networks we consider, we frequently must relay messages containing time-sensitive information, therefore ignoring or delaying transmission of any message is undesirable.

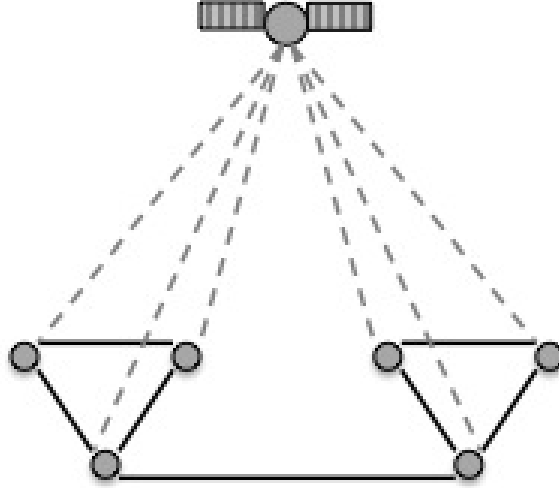


Figure 1: Mobile satellite-terrestrial network

Additionally, the mobile satellite-terrestrial networks we consider operate in an urban canyon. Urban canyons are environments in which man-made objects, such as skyscrapers, cause satellite blockage. As terrestrial nodes move around this environment, links between terrestrial nodes and satellite nodes are lost intermittently. Because of this, it is difficult for a centralized algorithm to gather information and transmit operational instructions to the whole of the network. Thus it is important that any algorithm be able to

operated locally, or in a distributed setting.

Considering these characteristics, any routing framework must be able to address:

- Concurrent transmission of multiple and independently critical unicast flows
- Time-variant edge capacities
- Need for distributed implementation

Existing frameworks for distributed routing protocols in networks of similar type are able to address either multiple, unicast flows in static networks [15] or a single unicast time-variant networks [22], but no work to date has been able to address multiple, unicast flows in time-variant networks.

1.2 Problem Statement

Given the network topology present in mobile satellite-terrestrial networks, this research project sought to determine the most efficient, effective means for routing packets. We define efficient and effective by:

- Flow rate
- Fairness of flow allocation
- Ability to be employed in a distributed manner
- Algorithm speed

A number of different algorithms, based on the multicommodity flow problem, were adapted in an attempt to address the problems of time-varying link capacities, simultaneous unicast communications, and limited local awareness of overall network behavior. Then, these algorithms were run on simulated networks. An initial assessment of the advantages and disadvantages of each algorithm was undertaken by looking at the performance metrics (efficiency and effectiveness, as defined above) in the simulated networks. Theoretical analysis of each algorithm provided further assessment of performance or suitability for the problem. Based on these findings, a distributed implementation plan was developed for the most promising algorithm, for adaptation to a routing protocol.

1.2.1 Project Goals

1. Modify multicommodity flow problems to model the network topology of terrestrial-satellite communications networks
2. Develop algorithms for solving these modified multicommodity flow models
3. Evaluate effectiveness, efficiency, and suitability of algorithms through analysis of simulated networks and theoretical analysis
4. Create a distributed implementation plan for the most promising algorithm for adaptation to communication protocols

1.2.2 Work Activities

1. Review literature on multicommodity flow and possible frameworks (sampling algorithms, lower-bound approximations, stochastic subgradient algorithms) for solving the invariant allocation problem
2. Define performance metrics for routing and distributed implementation
3. Modify existing distributed multicommodity flow algorithms to operate in a time-variant setting
4. Develop invariant allocation (static, robust) algorithms that account for a time-variant network topology
 - (a) Sampling algorithm
 - (b) Lower-bound approximation
 - (c) Stochastic supergradient algorithm
5. Run algorithms in simulated networks
6. Compare performance of algorithms in simulated networks based on defined metrics
7. Compare performance of algorithms through theoretical analysis based on defined metrics
8. Develop a distributed implementation plan

2 Literature Review

Much work has been done on approximately solving the static multicommodity flow problem for large networks, beginning with Dantzig and Wolfe's dual decomposition method in 1960 [12]. Shahrokhi and Matula developed the first fully polynomial time approximation algorithm for the maximum concurrent flow problem in 1990 [21]. Work since has generalized the method for a greater variety of network topologies and has improved upon speed, including work by Andrews [3] and Garg and Könemann [14]. Other work, like Awerbuch and Leighton awerbuch2, Awerbuch and Khandekar [5], and Kamath, Palmon, and Plotkin [15], has sought to find methods for implementing these approximation methods in a variety of distributed settings.

Minimal work has been done on network flow problems in a time-variant setting. Shrader et al [22] built a heuristic routing protocol that seeks to use multiple paths to find maximum flow for a single unicast flow in time-variant networks. Neely, Modiano, and Rohrs [18] looked at allocating power in time varying networks. Andrews [4] showed that his approximation algorithm for static networks was stable for time-variant networks where no edge was critically loaded.

The subgradient method has been used extensively for static multicommodity flow problems. In fact, the Dantzig-Wolfe method is a projected subgradient method, as explained in Martins et al [16]. A proof that the stochastic subgradient method converges appears in notes by Boyd [8] as

well as extensive practical information on projected and constrained subgradient methods by Boyd [7].

The stochastic subgradient method has also been formulated for distributed settings by Ram, Nedic, and Veeravalli [19] as well as Nedic and Ozdaglar [17] for network optimization. Duchi [?] showed that a form of the subgradient method converges when sampling from ergodic processes, a later paper [1] shows that any online optimization method converges when sampling from ergodic processes. *Parallel and Distributed Computation: Numerical Methods* [9] gives general needs and terminology for distributed implementation of network flow problems.

3 Research

In this section, we establish the general approach of this research toward building a framework for routing protocols in networks with our specified topology.

3.1 Background

3.1.1 Multicommodity Flow

There has been much success in adapting network flow problems for routing in cases of unicast flow. Previous research on networks with time-variant edge capacities successfully used the maximum flow problem as a basis for routing protocols [22].

Because we are concerned with networks that have concurrent transmission of multiple, unicast flows, heuristics based on the single commodity, maximum flow problem will fail [22]. However, multicommodity flow problems, the natural extension of maximum flow problems for demands of multiple, concurrent, unicast flows, has been adapted successfully for routing protocols in static networks [3] [14]. Thus, we use a version of multicommodity flow to frame our routing problem throughout this project.

The remainder of this section will describe the multicommodity flow problem in more detail and explicitly define a formulation of the multicommodity flow problem that is the backbone of our analysis.

Multicommodity network flow uses a linear programming formulation to

solve network flow problems where there are multiple routing demands that have specific source-sink pairs. The traditional multicommodity flow problem seeks to maximize total flow across the network, but other formulations seek to minimize total cost, or illicit a fair allocation of flow [21]. Because the networks focused on in this research must relay messages containing time-sensitive information, ignoring or delaying transmission of any message is undesirable. To avoid ignoring any specific message, we use a multicommodity flow formulation that seeks to fairly allocate flow, called maximum concurrent flow [3]. The maximum concurrent flow problem is formally defined as:

$$\begin{aligned}
& \text{maximize: } z \\
& \text{subject to: } zD_i \leq \sum_{j=1}^{n_i} f_{i,j} \quad \text{for all } i \\
& \sum_{(i,j) \in P_e} f_{i,j} \leq c_e \quad \text{for all } e
\end{aligned} \tag{1}$$

where z is the throughput, or fraction of demand, D_i , allocated for each commodity i , $f_{i,j}$ is the total flow that commodity i routes along path j , P_e is the set of paths that span over edge e , and c_e is the available capacity across edge e . The first set of constraints ensures flow conservation, and the second set of constraints are the capacity constraints.

3.1.2 Fully Polynomial Time Approximation Algorithms

Although multicommodity flow problems can be solved in polynomial time using linear programming techniques, this method becomes infeasible as the networks become large [21]. Furthermore, this method requires centralized control [15]. Therefore, approximation algorithms that are capable of solving problems in polynomial time have been developed for a variety of static linear programs. Further developments have modified these algorithms to operate with local control or otherwise in a distributed setting.

Because fully polynomial time approximation algorithms address the need for solving multicommodity flow problems quickly in a distributed setting, these algorithms seemed to be a promising approach for adaptation to the time-variant model. To fully explore these approximation algorithms, we adapted two of the most frequently explored algorithms for multicommodity flow in the literature: Lagrangian relaxation methods and Dantzig-Wolfe methods. In our approach, the algorithms run as usual, but face changing capacities after a set number of iterations.

In 1990, Shahrokhi and Matula [21] created a price-directive algorithm for the maximum concurrent flow model for cases with unit capacity across all edges. Price-directive algorithms use the dual variables associated with capacity constraints to decompose the multicommodity problem into multiple single commodity problems [24]. Since then, many papers have focused on generalizing the algorithm for a wider range of network topologies and improved computation speed. Algorithms fall into two general categories:

Lagrangian relaxation methods and Danzig-Wolfe methods. Lagrangian relaxation methods create an objective function that multiplies the capacity constraints by Lagrangian multipliers, and then solve a minimum cost flow subroutine for each commodity. Danzig-Wolfe methods impose prices on the capacity constraints while running a subroutine that optimizes the flow constraints for each commodity.

Garg and Könemann [14] developed two algorithms that use the Lagrangian relaxation method. One uses a minimum cost subroutine, while the other uses a shortest path subroutine. The method is as follows:

1. Set the initial dual variables assigned to each edge, $\lambda_e^0 = 1/c_e$
2. In iteration k , route each commodity across paths that minimize λ^k
3. Update

$$\lambda_e^{k+1} = \lambda_e^k \left(1 + \epsilon \frac{\sum_i x_e^{k,i}}{c_e} \right)$$

where $x_e^{k,i}$ is the flow across edge e from commodity i

4. Return to 2.

Andrews [3] created a maximum-weight algorithm that uses a Danzig-Wolfe method while taking a moving average of iterative solutions. The method is as follows:

1. Set the initial dual variables assigned to each edge, $\lambda_e^0 = c_e$
2. In iteration k , route D_i units of flow for each commodity across paths minimizing λ^k

3. Maximize z^k subject to $z^k(\lambda^k)^T \sum_i x^{k,i} \leq (\lambda^k)^T c$
4. Update $\lambda^{k+1} = \max\{0, \lambda^k - c + z^k \sum_i x^{k,i}\}$
5. Return to 2.
6. The flow in iteration k is set as the moving average $\frac{1}{k-k'} \sum_{j=k-k'}^k x^{j,i}$

Because the FPTAP methods rely on dual decomposition, that is, "... break[ing] ... the original problem into a set of smaller subproblems...[to]...yield a constrained optimization problem, the Lagrange dual of which is solved using a projected subgradient method" [16], the number of iterations required to converge to the solution within some ϵ is $O\left(\frac{1}{\epsilon^2}\right)$. In fact, the static Max Weight method requires $O\left(\frac{\rho^2}{\epsilon^2}\right)$ where ρ is the width of the problem, found by $\max_x (\lambda_{UB} \sum_i x^i / c)$ for some known upper bound λ_{UB} on the throughput λ . Because converging to a solution to this problem in the static case depends on the width of the problem, it is difficult to quantify convergence in the time-variant case. A change in the capacity realizations changes both the network topology and the width of a problem. When this occurs, the FPTAP acts as if it is solving a new problem. Thus, we can say that when

$$t_{iter} \frac{\rho^2}{\epsilon^2} \geq t_{rate}$$

where t_{tier} is the time it takes for on iteration of the algorithm, and t_{rate} is the time it takes for the the edge capacities to transition, the time-variant problem will not converge for that set of capacity realizations. Additionally,

although the algorithm is solving a new problem every time the capacity realizations change, the new problem usually will not begin with the optimal initial dual variables at time t ,

$$\lambda_e^t = C_e^{(t)}$$

but will instead begin with

$$\lambda_e^t = C_e^{(t-1)}$$

This has the potential to further slow the convergence of the algorithm.

3.1.3 Invariant Allocation

Preliminary simulations of the fully polynomial approximation algorithms for time-variant networks revealed that the algorithm struggled to adjust dual variables quickly enough for the changing problem. As a result, the algorithm was only able to reach 70-80% of the solution upper bound (found by solving the linear program in each iteration with the simplex method). While this phenomenon appears to be related to some ratio of algorithm speed, algorithm convergence rates, and capacity change rate, it is obvious that the solution with existing fully polynomial time approximation algorithms is limited in at least some cases. Thus, we sought another solution whose method could inherently account for changing capacities.

Cogill and Shrader [10] developed an algorithm that provide a static allocation of flow that is robust to time-variant edge capacities. Instead of

optimizing over path flows, the algorithm optimizes over a newly developed value, α , that uses a ratio of flow for a specific commodity and path to the total flow for all commodities and path over a specific edge. This allows for a system to solve a single optimization problem whose solution can be used for the problem even as the edge capacities change over time.

Unfortunately, the objective function of this optimization problem is very difficult to evaluate directly. Thus, we sought to create or adapt alternative methods by which this optimization problem could be solved or approximately solved.

The remainder of this section describes the invariant allocation problem in more detail and the methods by which we attempt to evaluate the objective function.

Consider:

$$\alpha_{i,j,e} = \frac{f_{i,j}}{\sum_{(i,j) \in P_e} f_{i,j}}$$

the ratio of flow for commodity i and path j to the total flow for all commodities and path over edge e . Since we know that:

$$\sum_{(i,j) \in P_e} f_{i,j} \leq c_e$$

$\alpha_{i,j,e}$ can be expressed as the fraction of edge e 's capacity that is allocated to path j for commodity i . Clearly:

$$f_{i,j} \leq \alpha_{i,j,e} c_e$$

Furthermore, we can express:

$$z = \min_i \left\{ \frac{1}{D_i} \sum_{j=1}^{n_i} f_{i,j} \right\}$$

and

$$f_{i,j} = \min_{e \in p_{i,j}} \{ \alpha_{i,j,e} C_e \}$$

Since we are interested in cases where the capacity is changing, we express the capacity realization, c_e as a random variable, C_e . This treatment requires that we look at the expected value, rather than an explicit solution to the problem. Using this and the substitutions above, the invariant allocation problem reduces from the maximum concurrent flow formulation to:

$$\begin{aligned} \text{maximize: } & \mathbf{E} \left[\min_i \left\{ \frac{1}{D_i} \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \{ \alpha_{i,j,e} C_e \} \right\} \right] \\ \text{subject to: } & \sum_{(i,j) \in P_e} \alpha_{i,j,e} = 1 \quad \text{for all } e \\ & \alpha_{i,j,e} \geq 0 \quad \text{for all } e \text{ and } (i,j) \in P_e \end{aligned} \tag{2}$$

where $\alpha_{i,j,e}$ is the fraction of edge e 's capacity that is allocated to path j for commodity i and C_e is the random variable representing the capacity for edge e . In this formulation, the flow conservation constraints have been brought into the objective function, and the capacity constraints are insured by the constraints on α values. It can be shown that this problem is equivalent to the static maximum concurrent flow problem when $C_e = c_e$.

As alluded to earlier, this objective function is very difficult to directly evaluate because one must find the expected value by assessing the probabilistic nature of the random variables C_e under two minimizations and one summation. Because this is impossible for all but trivial network topologies and probability distributions, this work primarily focuses on approximate solutions to this invariant allocation problem that can be found and implemented in a distributed setting. These methods are discussed in the following sections.

3.2 Sampling Algorithm

3.2.1 Concept

One approach to approximating the solution to the invariant allocation problem, developed by Cogill and Shrader, [10] is to sample from possible capacity realizations, and then solve a linear program that maximizes the sample average.

3.2.2 Execution

The linear program is as follows:

$$\begin{aligned}
& \text{maximize: } \frac{1}{S} \sum_{k=1}^S z_k \\
& \text{subject to: } D_i z_k \leq \sum_{j=1}^{n_i} f_{i,j,k} \quad \text{for all } i, k \\
& \quad f_{i,j,k} \leq \alpha_{i,j,e} c_{e,k} \quad \text{for all } e, k \text{ and } (i, j) \in P_e \\
& \quad \sum_{(i,j) \in P_e} \alpha_{(i,j,e)} = 1 \quad \text{for all } e \\
& \quad \alpha_{(i,j,e)} \geq 0 \quad \text{for all } e \text{ and } (i, j) \in P_e
\end{aligned} \tag{3}$$

where k denotes the k -th sample.

This problem can be solved for α prior to implementation. Then, the only requirement on the system for implementation is that each edge or vertex pair know the α values associated with it as well the current value of the capacity.

3.2.3 Performance: Speed and Accuracy

For large enough k , this solution will generate a close approximation of the true solution, however this comes at a computational cost. Each additional sample adds $|E| * |(I, J)| + |I|$ constraints to the problem, where $|E|$ denotes the number of edges, $|(I, J)|$ denotes the total number paths, and $|I|$ denotes the number of commodities. Furthermore, this method requires that the stochastic models for the edge capacities be known prior to implementation, which is not guaranteed in this method.

The sampling algorithm increases in size for an increase in the number of samples. This increase in size comes from the additional $|E| * |(I, J)| + |I|$ constraints that are added to the problem for every additional sample. Now, to arrive at a conservative estimate for the number of samples required, we consider the average time it takes to see all possible $\min_{e \in P_{i,j}} \{\alpha_{i,j,e} C_e\}$ on a path, $\frac{1}{\min_{e \in P_{i,j}} \{p_e, 1 - p_e\}}$. The time it takes to see all combinations of $\min_{e \in P_{i,j}} \{\alpha_{i,j,e} C_e\}$ for all paths and commodities is then:

$$\frac{1}{\prod_i \prod_{j=1}^{n_i} \min_{e \in P_{i,j}} \{p_e, 1 - p_e\}}$$

Even if we use quicker techniques like fully polynomial time approximation algorithms, solution time is still $O\left(\frac{\rho^2}{\epsilon^2}\right)$, where the width of the problem is conservatively $\max_k(\rho_k) * (\text{number of samples})$.

Additionally, even with a less conservative estimate for the number of samples needed, it is difficult to quantify loss of solution accuracy because

the number of samples needed is dictated by the unique topology of the network.

Because the sampling method is a potentially unfeasible solution due to computation time and/or lack of knowledge of the stochastic model, two other approaches to solving the invariant allocation method were explored.

3.3 Lower Bound Approximation

3.3.1 Concept

Since the sampling algorithm increases in size by $|E| * |(I, J)| + |I|$ constraints for each additional sample k , it may become unfeasible to find a close enough approximation in a reasonable computation time. Therefore, we sought a formulation that could approximate the solution to the invariant allocation problem without requiring an increase in computation time for increased accuracy.

The lower bound approximation approach finds a lower bound on the expected value of the objective function of the invariant allocation problem, and then solves a linear program for the invariant allocation problem using the lower bound as the objective function. If the lower bound is tight enough, then the solution for α will closely approximate the α associated with the invariant allocation objective function.

Two lower bound approximations were developed. One, the logarithmic bound, closely follows the Chernoff-type bounds described in [11] for a set of dependent random variables. The other, the "simple" network bound, uses a closed-form solution for the expected value for a simple network topology to form a lower bound on expected value for more complex networks. While these lower bounds allow for one to solve a smaller linear program with $|E| * |(I, J)| + |I|$ constraints rather than $k * (|E| * |(I, J)| + |I|)$ constraints, they both still require prior knowledge of the stochastic model.

The remainder of this section will show the initial derivation required for the two lower bounds.

Recall the objective function in the invariant allocation problem:

$$\mathbf{E} \left[\min_i \left\{ \frac{1}{D_i} \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \{ \alpha_{i,j,e} C_e \} \right\} \right]$$

Ideally, one would like to bring the expected value function further into the problem to facilitate an approximation. Unfortunately, using Jensen's Inequality to bring the expected value into the minimum yields an upper bound rather than a lower bound on the problem. Further research into the problem did not yield any successful approaches to approximation. So, in order to create some approximation, we turned to the maximum multicommodity flow formulation of the problem. This formulation finds the maximum flow rate by summing each commodity's rate, rather than taking the minimum rate as in maximum concurrent flow. In symmetric networks, this formulation yields equivalent α values to the maximum concurrent flow formulation, but it may diverge in asymmetric networks. Using this formulation, we now have the objective function:

$$\mathbf{E} \left[\sum_i \frac{1}{D_i} \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \{ \alpha_{i,j,e} C_e \} \right]$$

which easily simplifies to:

$$\sum_i \frac{1}{D_i} \sum_{j=1}^{n_i} \mathbf{E} \left[\min_{e \in p_{i,j}} \{ \alpha_{i,j,e} C_e \} \right]$$

It is clear to see that if we can find a lower bound on $\mathbf{E} [\min_{e \in p_{i,j}} \{ \alpha_{i,j,e} C_e \}]$ for each $e \in p_{i,j}$, we can find a lower bound for the maximum multicommodity flow formulation of the invariant allocation problem.

3.3.2 Execution: Logarithmic Bound

Adapting the proof for the Chernoff-type bounds found in [11], we find:

$$\begin{aligned} \min_{e \in p_{i,j}} \{ \alpha_{i,j,e} C_e \} &= -\frac{1}{t} \ln \left(\max_{e \in p_{i,j}} \{ e^{-\alpha_{i,j,e} C_e t} \} \right) \\ &\geq -\frac{1}{t} \ln \left(\sum_{e \in p_{i,j}} e^{-\alpha_{i,j,e} C_e t} \right) \end{aligned}$$

Adding expectation:

$$\begin{aligned} \mathbf{E} \left[\min_{e \in p_{i,j}} \{ \alpha_{i,j,e} C_e \} \right] &\geq -\frac{1}{t} \mathbf{E} \left[\ln \left(\sum_{e \in p_{i,j}} e^{-\alpha_{i,j,e} C_e t} \right) \right] \\ &\geq -\frac{1}{t} \ln \left(\mathbf{E} \left[\sum_{e \in p_{i,j}} e^{-\alpha_{i,j,e} C_e t} \right] \right) \\ &= -\frac{1}{t} \ln \left(\sum_{e \in p_{i,j}} \mathbf{E} [e^{-\alpha_{i,j,e} C_e t}] \right) \end{aligned}$$

We see that this inner expectation is simply a moment-generating function. Since the capacities can be expressed as Bernoulli random variables, this expectation becomes

$$\mathbf{E} \left[e^{-\alpha_{i,j,e} C_e t} \right] = p_e e^{-\alpha_{i,j,e} C_e^1 t} + (1 - p_e) e^{-\alpha_{i,j,e} C_e^0 t}$$

where p_e denotes the probability of the capacity being in state 1, and C_e^1 and C_e^0 are capacity realizations in state 1 and state 0, respectively. Combining these results, we have a lower bound approximation for the maximum multicommodity flow invariant allocation problem:

$$\begin{aligned} \text{maximize: } & \sum_i \frac{1}{D_i} \sum_{j=1}^{n_i} -\frac{1}{t} \ln \left(\sum_{e \in p_{i,j}} p_e e^{-\alpha_{i,j,e} C_e^1 t} + (1 - p_e) e^{-\alpha_{i,j,e} C_e^0 t} \right) \\ \text{subject to: } & \sum_{(i,j) \in P_e} \alpha_{i,j,e} = 1 \quad \text{for all } e \\ & \alpha_{i,j,e} \geq 0 \quad \text{for all } e \text{ and } (i,j) \in P_e \end{aligned} \tag{4}$$

3.3.3 Execution: "Simple" Network Bound

For this approximation, we explore a "simple" network, one with identical capacity realizations across all edges where the lower realization is zero, for which we can find a closed-form solution to the objective function for the invariant allocation problem.

If all C_e have identically capacity realizations across all edges and the

lower realization is zero, then:

$$\min_{e \in p_{i,j}} \{\alpha_{i,j,e} C_e\} = \min_{e \in p_{i,j}} \{\alpha_{i,j,e}\} \min_{e \in p_{i,j}} \{C_e\}$$

Now:

$$\begin{aligned} \mathbf{E} \left[\min_{e \in p_{i,j}} \{\alpha_{i,j,e} C_e\} \right] &= \mathbf{E} \left[\min_{e \in p_{i,j}} \{\alpha_{i,j,e}\} \min_{e \in p_{i,j}} \{C_e\} \right] \\ &= \min_{e \in p_{i,j}} \{\alpha_{i,j,e}\} \mathbf{E} \left[\min_{e \in p_{i,j}} \{C_e\} \right] \end{aligned}$$

Because the lower capacity realization is zero, the only instance in which flow will be greater than zero is when all edges in the path have the non-zero capacity realization. This happens with probability $\prod_{e \in p_{i,j}} p_e$. Thus we have

$$\mathbf{E} \left[\min_{e \in p_{i,j}} \{C_e\} \right] = C_e^1 \prod_{e \in p_{i,j}} p_e + C_e^0 (1 - \prod_{e \in p_{i,j}} p_e)$$

Combining these results, we have a closed form solution for the maximum multicommodity flow invariant allocation problem for problems of this network type:

$$\begin{aligned} \text{maximize: } & \sum_i \frac{1}{D_i} \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \{\alpha_{i,j,e}\} (C_e^1 \prod_{e \in p_{i,j}} p_e + C_e^0 (1 - \prod_{e \in p_{i,j}} p_e)) \\ \text{subject to: } & \sum_{(i,j) \in P_e} \alpha_{i,j,e} = 1 \quad \text{for all } e \\ & \alpha_{i,j,e} \geq 0 \quad \text{for all } e \text{ and } (i,j) \in P_e \end{aligned} \tag{5}$$

Note that if the capacities do not fit this framework, then:

$$\min_{e \in P_{i,j}} \{\alpha_{i,j,e} C_e\} \geq \min_{e \in P_{i,j}} \{\alpha_{i,j,e}\} \min_{e \in P_{i,j}} \{C_e\}$$

Thus for any network, this solution is a lower bound on the maximum multicommodity flow invariant allocation problem.

3.3.4 Suitability: Maximum Multicommodity Flow vs Maximum Concurrent Flow

In symmetrical networks, that is, networks where $\sum_{j=1}^{n_i} \min_{e \in P_{i,j}} \{\alpha_{i,j,e} C_e\}$ is the same for all i , the maximum multicommodity flow formulation and the maximum concurrent flow formulation will yield the same α values. But consider networks that are not symmetric. In the most extreme case, if there is any commodity m for which $\sum_{j=1}^{n_m} \min_{e \in P_{m,j}} \{\alpha_{m,j,e} C_e\} = 0$, then the maximum concurrent flow throughput $z = 0$. But the maximum multicommodity flow throughput is still $z = \sum_{i \neq m} \sum_{j=1}^{n_i} \min_{e \in P_{i,j}} \{\alpha_{i,j,e} C_e\}$. Even in cases where one commodity simply has a lower value for $\sum_{j=1}^{n_i} \min_{e \in P_{i,j}} \{\alpha_{i,j,e} C_e\}$, any commodity's flow that is in excess of this value will not be incorporated into the results for maximum concurrent flow, but will be incorporated into the results for maximum multicommodity flow. Even if we consider simply using Jensen's inequality (as an upper bound), the problem still requires a

symmetric network to converge to the optimal solution. Consider:

$$\mathbf{E} \left[\min_i \left\{ \frac{1}{D_i} \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \{ \alpha_{i,j,e} C_e \} \right\} \right] = \sum_r p_r \min_i \left\{ \frac{1}{D_i} \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \{ \alpha_{i,j,e} C_e^r \} \right\}$$

$$\min_i \left\{ \mathbf{E} \left[\frac{1}{D_i} \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \{ \alpha_{i,j,e} C_e \} \right] \right\} = \min_i \left\{ \frac{1}{D_i} \sum_{j=1}^{n_i} \sum_r p_r \min_{e \in p_{i,j}} \{ \alpha_{i,j,e} C_e^r \} \right\}$$

where p_r is the probability of seeing a particular set of realizations r . The first problem, which is the problem we wish to solve, looks at many samples of the minimum flow over all commodities. The second problem, which is closer to the problem we are approximating with the lower bound approximation approaches, looks at the minimum over all commodities given many samples of the flow for individual commodities. The lower bound approximation approach simply tries to push as much flow as possible, rather than pushing as much flow as possible so that each commodity is equally served. Thus, edges where one commodity needs to utilize the edge more than another edge, but produces less overall flow, will not gain a fair share of that edge's capacity.

3.3.5 Performance: Logarithmic Limitations

The logarithmic nature of the first lower bound approximation approach creates some difficulty in finding solutions, namely, the problem is not always concave so we find local maxima rather than a global maximum.

Consider a simplified version of the problem:

$$-\frac{1}{t} \ln (e^{-xt} + e^{-yt})$$

where x and y represent some set of $-\alpha_{i,j,e}C_e$. Examining the Hessian, it can be seen that this problem has principle minors $\Delta_1 < 0$, $\Delta_2 = 0$, and $\Delta_3 = 0$. Thus the problem is concave, but not strictly concave, and there are optimal solutions where $x = y$. But consider a case where there is a coefficient in front of either of the exponents, or a coefficient in front of either x or y . The problem is no longer concave.

3.4 Stochastic Supergradient Method

3.4.1 Concept

Although the lower bound approximation approach is able to limit the size and computation time needed to approximate the solution to the invariant allocation problem, both lower bound approximation formulas still require prior knowledge of the stochastic model. If this information is unavailable, neither the sampling algorithm nor the lower bound approximations could be used in a routing protocol without a "lag" time during which an approximate stochastic model is determined. As messages may contain time sensitive information, these approaches may not always yield desirable results. Thus we sought an approach that could both limit computation time and run without a predetermined stochastic model.

We considered the stochastic supergradient method because it is capable of approximating a solution for stochastic programming problems without a known model and it can easily be implemented in a distributed manner, since each iteration in the method requires limited memory and computation [8]. The supergradient method, rather than gradient descent, is required because the invariant allocation problem has a piecewise objective function that is not differentiable everywhere, but is still concave. Unfortunately, the stochastic supergradient method has only been proven for cases with independent, identically distributed random variables, and in our networks the capacity random variables appear to behave as two state Markov chains.

However, since the steady-state of Markov chains are independent and identically distributed, the algorithm is likely still a good approximation, though theoretical analysis is needed to validate. We use the stochastic supergradient method and develop a distributed implementation plan for use in the network.

3.4.2 Execution

Given the objective function:

$$\mathbf{E} \left[\min_i \left\{ \frac{1}{D_i} \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \{ \alpha_{i,j,e} C_e \} \right\} \right] = \mathbf{E} [f(\alpha, C)]$$

consider the supergradient:

$$f(z, C) \leq f(x, C) + g^T(z - x), \quad g \in \partial f(\alpha, C)$$

Because C is a random variable we are unable to compute the supergradient directly, however, we can compute a noisy, unbiased supergradient:

$$\tilde{g} = \frac{1}{M} \sum_{r=1}^M g(\alpha, C^r), \quad g = \mathbf{E}[\tilde{g}|\alpha]$$

such that $f(z, C) \leq f(x, C) + \mathbf{E}[\tilde{g}|\alpha]^T (z - x)$ almost surely

of a set of M realizations of C , C^r . This supergradient is only unbiased when C is independent and identically distributed.

This noisy, unbiased supergradient allows for an iterative update on α that will eventually converge to the solution to the objective function of the invariant allocation problem.

$$\alpha^{(k+1)} = \alpha^{(k)} + \beta^{(k)} \tilde{g}^{(k)}$$

where $\beta^{(k)}$ is not summable but square summable

$$\beta^{(k)} \geq 0, \quad \sum_{k=1}^{\infty} (\beta^{(k)})^2 < \infty, \quad \sum_{k=1}^{\infty} \beta^{(k)} = \infty$$

Because the invariant allocation problem also has constraints

$$\begin{aligned} \sum_{(i,j) \in P_e} \alpha_{i,j,e} &= 1 && \text{for all } e \\ \alpha_{i,j,e} &\geq 0 && \text{for all } e \text{ and } (i,j) \in P_e \end{aligned}$$

we must incorporate these into the update. The equality constraints are incorporated by using a projection, Π_α , of the constraints onto the update:

$$\begin{aligned} \alpha^{(k+1)} &= \alpha^{(k)} + \beta^{(k)} \Pi_\alpha \tilde{g}^{(k)} \\ \Pi_\alpha &= I_{|E|*|(I,J)|} - A^T (A A^T)^{-1} A \\ A &= [I_{|E|*|(1,J)|} \ I_{|E|*|(2,J)|} \ \cdots \ I_{|E|*|(i,J)|}] \end{aligned}$$

The inequality constraints are handled by creating a penalty supergradient

where the constraints are violated:

$$\tilde{g}^{(k)} = \begin{cases} \frac{1}{M} \sum_{r=1}^M g(\alpha_{i,j,e}, C^r), & \alpha_{i,j,e}^{(k-1)} \geq 0 \\ \partial \alpha_{i,j,e} = 1, & \alpha_{i,j,e}^{(k-1)} < 0 \end{cases}$$

To summarize, the stochastic supergradient method finds solutions to the stochastic programming problem

$$\text{maximize: } \mathbf{E}[f(\alpha, C)]$$

$$\text{subject to: } \sum_{(i,j) \in P_e} \alpha_{i,j,e} = 1 \quad \text{for all } e$$

$$\alpha_{i,j,e} \geq 0 \quad \text{for all } e \text{ and } (i,j) \in P_e$$

as follows:

1. In iteration k , find a noisy unbiased supergradient $\tilde{g}^{(k)}$
2. Update $\alpha^{(k+1)} = \alpha^{(k)} + \beta_{(k)} \tilde{g}^{(k)}$
3. Find $f_{best}^{(k)} = \max\{f(\alpha^{(1)}), \dots, f(\alpha^{(k)})\}$
4. Return to 1.

There are no stopping criteria for the stochastic supergradient method. Therefore, there are two approaches to picking an α to use in the network at time k . We can use $\alpha^{(k)}$, or we can find $\alpha_{best}^{(k)} = \arg \max_{\alpha} \{f_{best}^{k-1}, f(\alpha^{(k)})\}$. If either $\alpha^{(k)}$ or $\alpha_{best}^{(k)}$ is unfeasible, we store the closest previous α iterate which is feasible. While choosing $\alpha_{best}^{(k)}$ will usually yield a better approximation,

if requires greater storage because the objective function must be explicitly calculated in each iteration.

3.4.3 Suitability: Sampling from a Markov Process

We want to establish that the stochastic supergradient method converges for random variables that come from a finite state, irreducible, aperiodic Markov chain. This proof currently only exists for the case where the random variables are i.i.d. Using the stochastic subgradient method proof in [8], we show that the stochastic subgradient method converges for random variables that come from a finite state, irreducible, aperiodic Markov chain, where the variables are constrained to values between 0 and 1. This proof can easily be adapted to the supergradient case.

Using the proof in [8] we know:

$$\begin{aligned}
\mathbf{E} \left[\|\alpha^{(k+1)} - \alpha^*\|^2 | \alpha^{(k)} \right] &= \mathbf{E} \left[\|\alpha^{(k)} - \beta^{(k)} \tilde{g}^{(k)} - \alpha^*\|^2 | \alpha^{(k)} \right] \\
&= \|\alpha^{(k)} - \alpha^*\|^2 - 2\beta^{(k)} \mathbf{E} \left[\tilde{g}^{(k)T} (\alpha^{(k)} - \alpha^*) | \alpha^{(k)} \right] + (\beta^{(k)})^2 \mathbf{E} \left[\|\tilde{g}^{(k)}\|^2 | \alpha^{(k)} \right] \\
&= \|\alpha^{(k)} - \alpha^*\|^2 - 2\beta^{(k)} \mathbf{E} \left[\tilde{g}^{(k)} | \alpha^{(k)} \right]^T (\alpha^{(k)} - \alpha^*) + (\beta^{(k)})^2 \mathbf{E} \left[\|\tilde{g}^{(k)}\|^2 | \alpha^{(k)} \right]
\end{aligned}$$

Now, since the noisy subgradient is taken with samples drawn from a Markov process, specifically the distribution P^k at time k , where P is the transition matrix, we are actually concerned with $\mathbf{E}_{P^k} \left[\tilde{g}^{(k)} | \alpha^{(k)} \right]^T$ which we can expand to:

$$\mathbf{E}_{P^k} [\tilde{g}^{(k)} | \alpha^{(k)}]^T = \mathbf{E}_\pi [\tilde{g}^{(k)} | \alpha^{(k)}]^T + \underbrace{(\mathbf{E}_{P^k} [\tilde{g}^{(k)} | \alpha^{(k)}]^T - \mathbf{E}_\pi [\tilde{g}^{(k)} | \alpha^{(k)}]^T)}_{\mathbf{E}[v]}$$

where π is the steady-state distribution of the Markov process, and $\mathbf{E}[v]$ denotes the expected value of the noise.

We know that since $\mathbf{E}_\pi [\tilde{g}^{(k)} | \alpha^{(k)}]$ is a noisy, unbiased subgradient, we have:

$$\begin{aligned} f(\alpha^*) &\geq f(\alpha^{(k)}) + \mathbf{E}_\pi [\tilde{g}^{(k)} | \alpha^{(k)}]^T (\alpha^* - \alpha^{(k)}) \\ f(\alpha^*) - f(\alpha^{(k)}) &\geq \mathbf{E}_\pi [\tilde{g}^{(k)} | \alpha^{(k)}]^T (\alpha^* - \alpha^{(k)}) \\ f(\alpha^{(k)}) - f(\alpha^*) &\leq \mathbf{E}_\pi [\tilde{g}^{(k)} | \alpha^{(k)}]^T (\alpha^{(k)} - \alpha^*) \end{aligned}$$

Then:

$$\begin{aligned} \mathbf{E}_{P^k} [\tilde{g}^{(k)} | \alpha^{(k)}]^T (\alpha^{(k)} - \alpha^*) &= \mathbf{E}_\pi [\tilde{g}^{(k)} | \alpha^{(k)}]^T (\alpha^{(k)} - \alpha^*) + \mathbf{E}[v]^T (\alpha^{(k)} - \alpha^*) \\ &\geq f(\alpha^{(k)}) - f(\alpha^*) + \mathbf{E}[v]^T (\alpha^{(k)} - \alpha^*) \end{aligned}$$

Plugging in, we have:

$$\mathbf{E} [\|\alpha^{(k+1)} - \alpha^*\|^2 | \alpha^{(k)}] \leq \|\alpha^{(k)} - \alpha^*\|^2 - 2\beta^{(k)} \left(f(\alpha^{(k)}) - f(\alpha^*) - \mathbf{E}[v]^T (\alpha^{(k)} - \alpha^*) \right) + (\beta^{(k)})^2 \mathbf{E} [\|\tilde{g}^{(k)}\|^2 | \alpha^{(k)}]$$

Taking expectation:

$$\mathbf{E} [\|\alpha^{(k+1)} - \alpha^*\|^2] \leq \mathbf{E} [\|\alpha^{(k)} - \alpha^*\|^2] - 2\beta^{(k)} \left(\mathbf{E}[f(\alpha^{(k)})] - f(\alpha^*) - \mathbf{E} [-\mathbf{E}[v]^T (\alpha^{(k)} - \alpha^*)] \right) + (\beta^{(k)})^2 \mathbf{E} [\|\tilde{g}^{(k)}\|^2]$$

Applying recursively:

$$\mathbf{E} \left[\|\alpha^{(k+1)} - \alpha^*\|^2 \right] \leq \mathbf{E} \left[\|\alpha^{(1)} - \alpha^*\|^2 \right] - 2 \sum_{i=1}^k \beta^{(i)} \left(\mathbf{E}[f(\alpha^{(i)})] - f(\alpha^*) \right) + 2 \sum_{i=1}^k \beta^{(i)} \mathbf{E} \left[\mathbf{E}[v]^T (\alpha^* - \alpha^{(i)}) \right] + \sum_{i=1}^k (\beta^{(i)})^2 \mathbf{E} \left[\|\tilde{g}^{(i)}\|^2 \right]$$

To complete the proof, we need to bound:

- $\mathbf{E} \left[\|\alpha^{(k+1)} - \alpha^*\|^2 \right]$
- $\mathbf{E} \left[\|\alpha^{(1)} - \alpha^*\|^2 \right]$
- $\mathbf{E} \left[\mathbf{E}[v]^T (\alpha^* - \alpha^{(i)}) \right]$
- $\mathbf{E} \left[\|\tilde{g}^{(i)}\|^2 \right]$

Clearly, $\mathbf{E} \left[\|\alpha^{(k+1)} - \alpha^*\|^2 \right] \geq 0$. In our problem α elements lie between 0 and 1, $|\alpha - \eta| < \vec{1}$, so we can bound $|\alpha - \eta| \leq R$, $\mathbf{E} \left[\|\alpha^{(1)} - \alpha^*\|^2 \right] \leq R^2$. Furthermore, since α elements lie between 0 and 1 and the random variables take on discrete, finite values, we also see $|\tilde{g}^{(i)}| \leq G$, $\mathbf{E} \left[\|\tilde{g}^{(i)}\|^2 \right] \leq G^2$. Finally, we must consider:

$$\mathbf{E} \left[\mathbf{E}[v]^T (\alpha^* - \alpha^{(i)}) \right] = \mathbf{E} \left[\left(\mathbf{E}_{P^i} \left[\tilde{g}^{(i)} | \alpha^{(i)} \right]^T - \mathbf{E}_\pi \left[\tilde{g}^{(i)} | \alpha^{(i)} \right]^T \right) (\alpha^* - \alpha^{(i)}) \right]$$

Since $\mathbf{E}_{P^i} \left[\tilde{g}^{(i)} | \alpha^{(i)} \right]^T$ and $\mathbf{E}_\pi \left[\tilde{g}^{(i)} | \alpha^{(i)} \right]^T$ are expected values of functions of random variables with realizations ω , we can write:

$$\mathbf{E}_{P^k} \left[\tilde{g}^{(i)} | \alpha^{(i)} \right] = \sum_{\Omega} \tilde{g}(\alpha^{(i)}, \omega) P^i(\omega)$$

and

$$\mathbf{E}_\pi \left[\tilde{g}^{(i)} | \alpha^{(i)} \right] = \sum_{\Omega} \tilde{g}(\alpha^{(i)}, \omega) \pi(\omega)$$

where $\tilde{g}(\alpha^{(i)}, \omega)$ is the supergradient assessed at $\alpha^{(i)}$ and realization ω .

Now we can write:

$$\begin{aligned}
\left(\mathbf{E}_{P^i} [\tilde{g}^{(i)} | \alpha^{(i)}]^T - \mathbf{E}_\pi [\tilde{g}^{(i)} | \alpha^{(i)}]^T \right) (\alpha^* - \alpha^{(i)}) &= \sum_{\Omega} \tilde{g}(\alpha^{(i)}, \omega)^T P^i(\omega) (\alpha^* - \alpha^{(i)}) - \sum_{\Omega} \tilde{g}(\alpha^{(i)}, \omega)^T \pi(\omega) (\alpha^* - \alpha^{(i)}) \\
&= \sum_{\Omega} \tilde{g}(\alpha^{(i)}, \omega)^T (\alpha^* - \alpha^{(i)}) (P^i(\omega) - \pi(\omega)) \\
&\leq \sum_{\Omega} \left| \tilde{g}(\alpha^{(i)}, \omega)^T (\alpha^* - \alpha^{(i)}) (P^i(\omega) - \pi(\omega)) \right| \\
&\leq \sum_{\Omega} \left\| \tilde{g}(\alpha^{(i)}, \omega)^T \right\| \left\| \alpha^* - \alpha^{(i)} \right\| |P^i(\omega) - \pi(\omega)| \\
&\leq GR \sum_{\Omega} |P^i(\omega) - \pi(\omega)|
\end{aligned}$$

where the second to last step follows from the Cauchy-Schwarz inequality.

We still need explicit bounds on the difference $|P^i(\omega) - \pi(\omega)|$ in order to complete the proof. Consider the total variation distance, d_{TV} , defined as:

$$d_{TV}(P, Q) = \sup_{A \in \mathcal{F}} |P(A) - Q(A)| = \frac{1}{2} \sum_S |P(S) - Q(S)|$$

for P and Q on probability space (S, \mathcal{F}) . We can see:

$$\left(\mathbf{E}_{P^i} [\tilde{g}^{(i)} | \alpha^{(i)}]^T - \mathbf{E}_\pi [\tilde{g}^{(i)} | \alpha^{(i)}]^T \right) (\alpha^* - \alpha^{(i)}) \leq 2GR d_{TV}(P^i, \pi)$$

Taking expectation:

$$\mathbf{E} \left[\left(\mathbf{E}_{P^i} [\tilde{g}^{(i)} | \alpha^{(i)}]^T - \mathbf{E}_\pi [\tilde{g}^{(i)} | \alpha^{(i)}]^T \right) (\alpha^* - \alpha^{(i)}) \right] \leq \mathbf{E} [2GR d_{TV}(P^i, \pi)] = 2GR d_{TV}(P^i, \pi)$$

Since P^i and π come from a finite state, irreducible, aperiodic Markov chain we have:

$$d_{TV}(P^i, \pi) \leq C\rho^i$$

for constants $\rho < 1$ and $C < \infty$.

Now we have bounded all the terms necessary to complete the proof, and we have:

$$\begin{aligned}
0 &\leq R^2 - 2 \sum_{i=1}^k \beta^{(i)} \left(\mathbf{E}[f(\alpha^{(i)})] - f(\alpha^*) \right) + 4GRC \sum_{i=1}^k \beta^{(i)} \rho^i + G^2 \sum_{i=1}^k (\beta^{(i)})^2 \\
2 \sum_{i=1}^k \beta^{(i)} \left(\mathbf{E}[f(\alpha^{(i)})] - f(\alpha^*) \right) &\leq R^2 + 4GRC \sum_{i=1}^k \beta^{(i)} \rho^i + G^2 \sum_{i=1}^k (\beta^{(i)})^2 \\
\min_{i=1, \dots, k} \left(\mathbf{E}[f(\alpha^{(i)})] \right) - f(\alpha^*) &\leq \frac{R^2 + 4GRC \sum_{i=1}^k \beta^{(i)} \rho^i + G^2 \sum_{i=1}^k (\beta^{(i)})^2}{2 \sum_{i=1}^k \beta^{(i)}} \\
\mathbf{E} \left[\min_{i=1, \dots, k} \left(\mathbf{E}[f(\alpha^{(i)})] \right) \right] - f(\alpha^*) &\leq \frac{R^2 + 4GRC \sum_{i=1}^k \beta^{(i)} \rho^i + G^2 \sum_{i=1}^k (\beta^{(i)})^2}{2 \sum_{i=1}^k \beta^{(i)}} \\
\mathbf{E} \left[f_{best}^{(k)} \right] - f(\alpha^*) &\leq \frac{R^2 + 4GRC \sum_{i=1}^k \beta^{(i)} \rho^i + G^2 \sum_{i=1}^k (\beta^{(i)})^2}{2 \sum_{i=1}^k \beta^{(i)}}
\end{aligned}$$

where the second to last step is a result of Jensen's inequality. Set:

$$\lim_{n \rightarrow \infty} \sum_{i=1}^n \beta^{(i)} = \infty, \quad \lim_{n \rightarrow \infty} \sum_{i=1}^n (\beta^{(i)})^2 < \infty$$

We can see that the denominator diverges. Since $\sum_{i=1}^k \rho^i$ is a geometric progression that converges and $\beta^{(i)} \rho^i \leq \rho^i$ for all i , by the direct comparison test, $\sum_{i=1}^k \beta^{(i)} \rho^i$ converges. Thus the numerator converges to some constant and $\mathbf{E} \left[f_{best}^{(k)} \right]$ converges to $f(\alpha^*)$.

3.4.4 Performance: Speed as Compared to FPTAP

This method always offers quicker solutions than FPTAP. Since both are supergradient methods, they appear to have the same running time, but FPTAP must build a new solution for every capacity change while the stochastic supergradient method improves on a single solution in every iteration.

3.5 Distributed Implementation

3.5.1 Concept

Any routing protocol for our specific problem requires a distributed implementation plan, in which nodes can communicate needs for the entire network simply through messaging between neighboring nodes. This is a requirement because the stochastic model may not be known, and connectivity between nodes is often limited. After preliminary analysis, it became clear that the stochastic supergradient model was best suited for distributed implementation. The following describes the requirements and pseudocode for distributed implementation of the stochastic supergradient method.

3.5.2 Execution

Using the following notation, we delineate pseudocode for running the stochastic supergradient algorithm in a distribution setting.

notation	description	
\mathbf{s}_i	source	
\mathbf{d}_i	sink	
\mathbf{u}_i	head node	where $(u_i, v_i) \in p_{i,j}$
\mathbf{v}_i	tail node	
$\mathbf{u}_{\{i\}}$	bottleneck head node	where $\{i\}$ is the set of commodities $i \in \{i\}$ for which $(u_{\{i\}}, v_{\{i\}}) \in p_{i,j}$
$\mathbf{v}_{\{i\}}$	bottleneck tail node	
$\mathbf{c}_{(\mathbf{u}_i, \mathbf{v}_i), k}$	edge capacity at time k	
\mathbf{t}_{ch}	time since last edge capacity change	
$\{\mathbf{j}\}_i$	set of all paths j for commodity i	
$\hat{\mathbf{g}}_i^{(k)}$	supergradient of	
	$\min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\}$	for all $j \in \{\mathbf{j}\}_i$

Ideally, a distributed implementation plan for the stochastic supergradient algorithm would only require that nodes be able to communicate with neighboring nodes. Our plan requires a few additional assumptions:

- Each node knows the commodities i and the paths j to which it belongs
- Each node knows which node comes prior to and which node follows it in the path $p_{i,j}$

- Each source node s_i knows how many bottleneck head nodes $u_{\{i\}}$ are such that $i \in \{i\}$
- Each bottleneck head node $u_{\{i\}}$ knows how many bottle neck head nodes are in the network
- Nodes can send and receive capacity information with neighboring nodes without saturating the incident edge

The distributed supergradient algorithm requires three subroutines:

- Establish the transition rate of the edge capacities
- Update α (supergradient method)
- Push flow

To establish the transition rate of the edge capacities, the algorithm asks each set of neighboring nodes to check for a change in incident edge capacity at a fast rate. This rate is predetermined based on empirical evidence or theoretical assumptions about the transition rates. When a change in capacity is detected, the nodes send the new rate of change information to the source nodes, which then compile and average the detected rates for some number of samples. The source nodes must send this information to all bottleneck head nodes along available paths; then the bottleneck head nodes must find a current minimum rate amongst all collected values and send it back along all available reverse paths. This process must be repeated $|I| - 1$ times to

ensure that an overall network minimum has been found. This new value is used to establish a rate at which each iteration of the other two subroutines occurs. Using the minimum rate found across all commodities ensures that the network is checking for changes conservatively. Alternatively, this subroutine does not have to be included in the algorithm if a good estimate for the transition rate can be established a priori, or the subroutine can be incorporated and updated within the other two subroutines.

To update α , the algorithm asks the sink nodes to send $\alpha_{i,j,e}^{(k-1)}$ and $c_{e,k}$ to the previous node in the path. Then each node along the reverse path computes the $\min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\}$ found up to that point. When the information about the values arrives at the source, the source calculates a local supergradient; that is, supergradient of $\min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\}$ for all paths $j \in \{j\}_i$, and sends it along to the bottleneck head nodes along with $\sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\}$. The bottleneck head nodes then assess a minimum of these amongst all commodities that the bottleneck head node shares. This information is then sent back to the source node to find a minimum amongst all the bottleneck head nodes that it shares. This process must be repeated $|I| - 1$ times to ensure that an overall network minimum has been found. The bottleneck head node then has the information to compute the supergradient and a local update on all α values for all $i \in \{i\}$. These updated values are then propagated to their respective nodes along the relevant paths and reverse paths.

To push flow, the algorithm must again ask the sink nodes to send $\alpha_{i,j,e}^{(k)}$ and $c_{e,k}$ to the previous node in the path to compute the $\min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\}$ found up to that point. However, this only needs to be done for paths on which the supergradient value has been changed; otherwise $\sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\} = \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\}$. Once the source receives this information, the source sends it along to the bottleneck head nodes. The bottleneck head nodes then assess a minimum of these amongst all commodities that the bottleneck head node shares. This information is then sent back to the source node to find a minimum amongst all the bottleneck head nodes that it shares. This process must be repeated $|I| - 1$ times to ensure that an overall network minimum has been found amongst all commodities i , which is equivalent to the desired flow rate, $z^{(k)}$. The source nodes push this flow until the next iteration of the algorithm.

What follows is the pseudocode associated with the methods described above.

Establish the transition rate of the edge capacities

- At time t , u_i **requests** capacity across edge (u_i, v_i) for all v_i
- u_i :
 - **evaluates**:
 - * if $c_{(u_i, v_i), t} = c_{(u_i, v_i), t - t_{ch}}$
 - discard** $c_{(u_i, v_i), t}$
 - * if $c_{(u_i, v_i), t} \neq c_{(u_i, v_i), t - t_{ch}}$
 - store** $c_{(u_i, v_i), t}$ and t

- **sends** t_{ch} along a reverse path $p_{i,j}$ to s_i
- When desired number of samples, $\#s_{amp}$ of t_{ch} is at s_i , s_i :
 - **sends** signal for all nodes along paths $p_{i,j}$, $j \in \{j\}_i$ to stop requesting capacity across edges
 - **evaluates** $t_{s_{amp},i} = \frac{\sum t_{ch}}{\#s_{amp}}$
- The following steps are repeated $|I| - 1$ times:
 - s_i :
 - * **waits** to receive $t_{s_{amp},i}$ from all $j \in \{j\}_i$
 - * **evaluates** $\min \{t_{s_{amp},i}\}$
 - * **sends** $t_{s_{amp},i} = \min \{t_{s_{amp},i}\}$ to all $u_{\{i\}}$, $i \in \{i\}$ along paths $p_{i,j}$, $j \in \{j\}_i$
 - $u_{\{i\}}$:
 - * **waits** to receive $t_{s_{amp},i}$ from all $i \in \{i\}$
 - * **evaluates** $\min \{t_{s_{amp},i}\}$
 - * **sends** $t_{s_{amp},i} = \min \{t_{s_{amp},i}\}$ to all s_i , $i \in \{i\}$ along reverse paths $p_{i,j}$
- $t_{rate} = t_{s_{amp},i}$

Update α

- s_i **sends** signal along paths $p_{i,j}$, $j \in \{j\}_i$ for d_i to **begin** at time k
- u_i :
 - **evaluates** $\min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\}$ for all $e \in p_{i,j} \setminus \{1_i, \dots, u_i - 1\}$, $(u_i - 1, u_i) \in p_{i,j}$
 - **sends** $\min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\}$, $\alpha_{i,j,e}^{(k-1)}$, $c_{e,k}$, and e to $u_i - 1$
- s_i :

- **waits** to receive $\min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\}$, $\alpha_{i,j,e}^{(k-1)}$, $c_{e,k}$, and e from all $j \in \{j\}_i$
- **evaluates** $\hat{g}_i^{(k)}$ and $\sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\}$
- **sends** $\hat{g}_i^{(k)}$ and $\sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\}$ to all $u_{\{i\}}$, $i \in \{i\}$ along paths $p_{i,j}$, $j \in \{j\}_i$
- $u_{\{i\}}$:
 - **waits** to receive $\hat{g}_i^{(k)}$ and $\sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\}$ to all $u_{\{i\}}$ from all $i \in \{i\}$
 - **evaluates** $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\} \right\}$ among available i
 - **sends** $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\} \right\}$ and associated i value to all $u_{\{i\}}$, $i \in \{i\}$ along reverse paths $p_{i,j}$, $j \in \{j\}_i$
- The following steps are repeated $|I - 2|$ times:
 - s_i :
 - * **waits** to receive $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\} \right\}$ and associated i value from all $u_{\{i\}}$ such that $i \in \{i\}$
 - * **evaluates** $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\} \right\}$ among available i
 - * **sends** $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\} \right\}$ and associated i value to all $u_{\{i\}}$, $i \in \{i\}$ along paths $p_{i,j}$, $j \in \{j\}_i$
 - $u_{\{i\}}$:
 - * **waits** to receive $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\} \right\}$ from all $i \in \{i\}$
 - * **evaluates** $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\} \right\}$ among available i
 - * **sends** $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\} \right\}$ and associated i value to all $u_{\{i\}}$, $i \in \{i\}$ along reverse paths $p_{i,j}$, $j \in \{j\}_i$

- $u_{\{i\}}$:
 - **evaluates** $i_{min}^{(k-1)} = \arg \min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\} \right\}$
 - **evaluates**:
 - * if $i = i_{min}^{(k-1)}$

$$\tilde{g}_i^{(k)} = \hat{g}_i^{(k-1)}$$
 - * if $i \neq i_{min}^{(k-1)}$

$$\tilde{g}_i^{(k)} = \vec{0}$$
 - **evaluates** local update $\alpha_{\{i\}}^{(k)} = \alpha_{\{i\}}^{(k-1)} + \Pi_{\alpha, \{i\}} \tilde{g}_{\{i\}}^{(k)}$
 - **sends** $\alpha_{i,j}^{(k)}$ along relevant paths and reverse paths $p_{i,j}$, $j \in \{j\}_i$ for all $i \in \{i\}$

Push flow

- $u_{\{i\}}$ **sends** signal along paths $p_{i,j}$, $i = i_{min}^{(k-1)}$, $j \in \{j\}_i$ for d_i to **begin**
- u_i :
 - **evaluates** $\min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\}$ for all $e \in p_{i,j} \setminus \{1_i, \dots, u_i - 1\}$, $(u_i - 1, u_i) \in p_{i,j}$
 - **sends** $\min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\}$ to $u_i - 1$
- if $i = i_{min}^{(k-1)}$, s_i :
 - **waits** to receive $\min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\}$ from all $j \in \{j\}_i$
 - **evaluates** $\sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\}$
 - **sends** $\sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\}$ to all $u_{\{i\}}$, $i \in \{i\}$ along paths $p_{i,j}$, $j \in \{j\}_i$
- else, s_i :
 - **sends** $\min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\} = \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k-1)} c_{e,k} \right\}$ to all $u_{\{i\}}$, $i \in \{i\}$ along paths $p_{i,j}$, $j \in \{j\}_i$

- $u_{\{i\}}$:

- **waits** to receive $\sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\}$ from all $i \in \{i\}$
- **evaluates** $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\} \right\}$ among available i
- **sends** $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\} \right\}$ to all $s_i, i \in \{i\}$ along reverse paths $p_{i,j}$

- The following steps are repeated $|I - 2|$ times:

- s_i :

- * **waits** to receive $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\} \right\}$ from all $u_{\{i\}}$ such that $i \in \{i\}$
- * **evaluates** $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\} \right\}$ among available i
- * **sends** $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\} \right\}$ to all $u_{\{i\}}, i \in \{i\}$ to all $u_{\{i\}}, i \in \{i\}$ along paths $p_{i,j}, j \in \{j\}_i$

- $u_{\{i\}}$:

- * **waits** to receive $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\} \right\}$ from all $i \in \{i\}$
- * **evaluates** $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\} \right\}$ among available i
- * **sends** $\min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\} \right\}$ to all $s_i, i \in \{i\}$ along reverse paths $p_{i,j}$

- s_i **pushes** $z^{(k)} = \min_i \left\{ \sum_{j=1}^{n_i} \min_{e \in p_{i,j}} \left\{ \alpha_{i,j,e}^{(k)} c_{e,k} \right\} \right\}$ along each path $p_{i,j}, j \in \{j\}_i$ until time $k + 1$, which occurs at t_{rate}

3.5.3 Suitability: Information Exchange Requirements

In order to find a minimum on the overall network, where we seek the minimum of a set of values where each value comes from a specific commodity, the source nodes must send the values to all bottleneck head nodes along available paths, then the bottleneck head nodes must send all collected values back along all available reverse paths. This process must be repeated $|I| - 1$ times to ensure that an overall network minimum has been found.

To further explore this concept, without loss of generality suppose x_1 is the overall network minimum of the set $\{x_1 \dots x_I\}$ where $|I|$ is the total number of commodities in the network.

If s_1 sends x_1 to all bottleneck head nodes along paths $p_{1,j}$, $j \in \{j\}_1$, then all $u_{\{i\}}$ such that $1 \in \{i\}$ have received x_1 .

If all $u_{\{i\}}$ such that $1 \in \{i\}$ send x_1 to all source nodes along all available reverse paths $p_{m,j}$, $\{1, m\} \subseteq \{i\}$, then all s_l such that $\{1, m\} \subseteq \{i\}$ have received x_1 .

Now, it is possible that there are commodities m such that $m \notin \{i\}$ for any $\{i\}$ such that $1 \in \{i\}$. However, for the network to be connected, there must be some commodity l_n for which $\{m, l_n\} \subseteq \{i\}$ for some set of commodities $\{i\}$, and furthermore, there must be some set of pairs of commodities in some bottleneck head node commodity sets $\{i\}$ by which $\{1, l_1\}, \{l_1, l_2\}, \dots, \{l_{n-1}, l_n\}, \{l_n, m\}$. This guarantees that if the two step process of exchanging the current minimum between all available sources and all available bottleneck head nodes back to all available sources is repeated,

s_m will eventually have received x_1 . This process takes at most $|I| - 1$ steps as the most conservative connectivity case is a network where each bottleneck head node serves only two commodities.

4 Results

4.1 General Approach

The following subsection describes the general approach of this research toward exploring the efficiency and effectiveness of each solution concept toward a framework for a routing protocol.

Concepts were explored through both empirical and theoretical analysis. Empirical analysis provided measures of efficiency and effectiveness of the method for specific example networks which in turn helped to inform the direction of the theoretical analysis for each method. Theoretical analysis sought to either confirm the suitability of the method for our problem or to validate performance seen in the empirical results.

Simulations of each method were run in MATLAB on example networks to gain an understanding of the advantages and limitations of each algorithm. Performance measures considered were:

- Flow rate
- Fairness of flow allocation
- Ability to be employed in a distributed manner
- Algorithm speed

Through analysis of field measurements, Shrader et al [22] determined that two-state Markov models are sufficient to simulate edge capacity changes

over time. Markov models were used in MATLAB simulations for this purpose.

Example networks were built according to topology that was cited in multi-commodity flow literature, met specific system requirements, and/or was theoretically intriguing. The main example networks are described at the end of this section.

Theoretical analysis, as shown in the research section, was based upon sources discovered in the literature review. Some proofs were necessary to determine the suitability of the approach for our problem. Specifically, suitability theoretical analysis: determined how much a lower bound approximation solution based upon maximum multicommodity flow would deviate from the maximum concurrent flow solution, determined whether or not the stochastic supergradient method could be used when random variables were Markov processes as opposed to independent and identically distributed, and determined the information sharing necessary for distributed implementation of the stochastic supergradient method. Information gleaned from simulations helped guide the direction and scope of the performance theoretical analysis to validate observations made during simulations. Performance theoretical analysis: determined in which scenarios fully polynomial time approximation algorithms could not adjust dual variables before edge capacity transitions, determined how slowly the sampling algorithm would perform given a particular network and accuracy of approximation required, and determined when a duality gap would occur in the logarithmic lower bound approach and how

it would affect the solution.

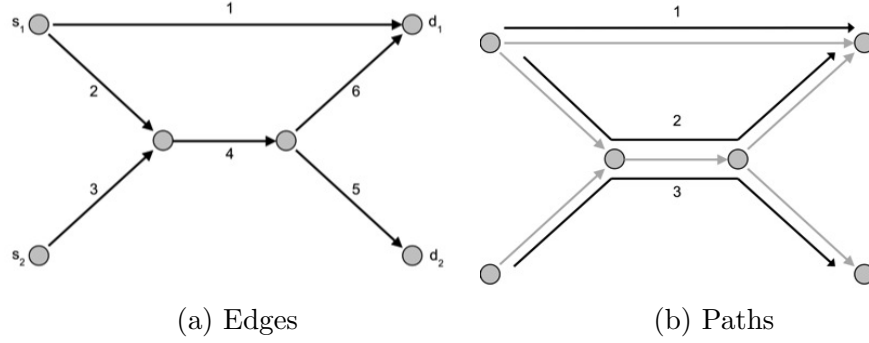


Figure 2: Example Network 1

Example Network 1 has two commodities. The flow from commodity 1 moves from the source node s_1 to the sink node d_1 via path 1 or 2. The flow from commodity 2 moves from the source node s_2 to the sink node d_2 via path 3. The capacities of all edges switch between values of 1 and 2 according to the transition matrix

$$P = \begin{bmatrix} .8 & .2 \\ .2 & .8 \end{bmatrix}$$

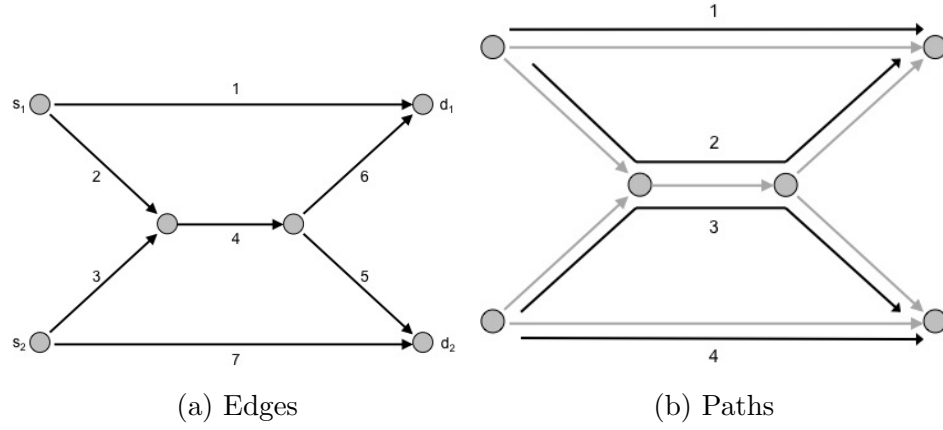


Figure 3: Example Network 2

Example Network 2 has two commodities. The flow from commodity 1 moves from the source node s_1 to the sink node d_1 via path 1 or 2. The flow from commodity 2 moves from the source node s_2 to the sink node d_2 via path 3 or 4. The capacities of all edges switch between values of 1 and 2 according to the transition matrix

$$P = \begin{bmatrix} .8 & .2 \\ .2 & .8 \end{bmatrix}$$

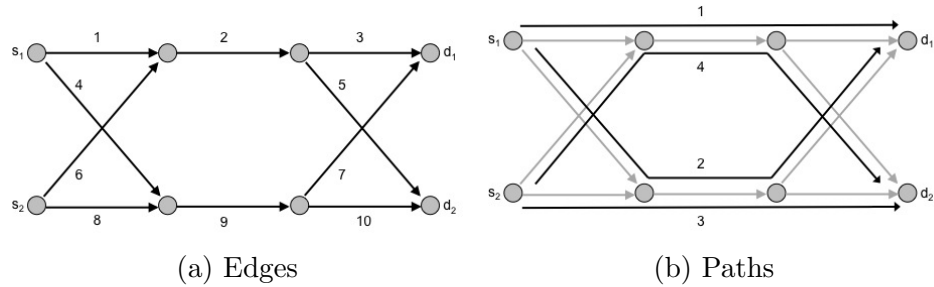


Figure 4: Example Networks 3 and 4

Example Network 3 has two commodities. The flow from commodity 1 moves from the source node s_1 to the sink node d_1 via path 1 or 2. The flow from commodity 2 moves from the source node s_2 to the sink node d_2 via path 3 or 4. The capacities of edges 1, 2, 3, 8, 9, and 10 switch between values of 1 and 2 according to the transition matrix:

$$P_1 = \begin{bmatrix} .8 & .2 \\ .2 & .8 \end{bmatrix}$$

and the capacities of edges 4, 5, 6, and 7 switch between values of .5 and 1 according to the transition matrix:

$$P_2 = \begin{bmatrix} .7 & .3 \\ .1 & .9 \end{bmatrix}$$

Example Network 4 has two commodities. The flow from commodity 1 moves from the source node s_1 to the sink node d_1 via path 1 or 2. The flow from commodity 2 moves from the source node s_2 to the sink node d_2 via path 3 or 4. The capacities of edges 1, 2, 3, 8, 9, and 10 switch between values of 0 and 2 according to the transition matrix:

$$P_1 = \begin{bmatrix} .8 & .2 \\ .2 & .8 \end{bmatrix}$$

and the capacities of edges 4, 5, 6, and 7 switch between values of 0 and 1

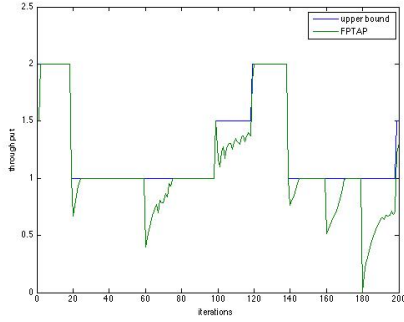
according to the transition matrix:

$$P_2 = \begin{bmatrix} .7 & .3 \\ .1 & .9 \end{bmatrix}$$

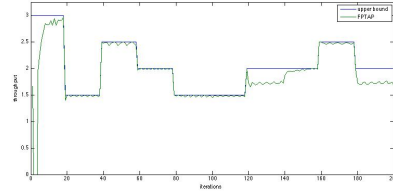
4.2 Fully Polynomial Time Approximation Algorithms

We ran a FPTAP algorithm (specifically Andrews' Max Weight Algorithm without a moving average) for 200 iterations with capacity realizations changing every 20 iterations. It is at this ratio of iterations to transition rate that one begins to see the inability of the algorithm to update for the current network topology. The following graphs show an average example of the performance of the algorithm as compared to solving the linear program directly for the current network topology for each example network.

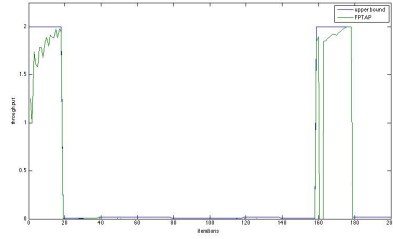
Example Network 1 had a degradation of 10.5% as compared to directly solving the linear program, Example Network 2 had a degradation of 5.6%, Example Network 3 had a degradation of 24.8%, and Example Network 4 had a degradation of 16.6%.



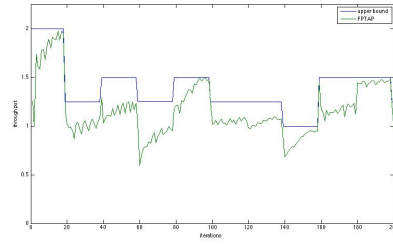
(a) Example Network 1



(b) Example Network 2



(c) Example Network 3



(d) Example Network 4

Figure 5: FPTAP Performance

4.3 Sampling Algorithm

In each instance, the sampling algorithm was run with 10, 50, 100 and 200 samples. The following charts show the α values found on edges where multiple paths, (i, j) , share the edge (otherwise, the algorithm finds $\alpha = 1$ where the edge on the path or $\alpha = 0$ when the edge is not in the path).

Example Network 1

10 samples				50 samples			
	(1,1)	(1,2)	(2,1)		(1,1)	(1,2)	(2,1)
edge 4	0	0	1	edge 4	0	0	1

100 samples				200 samples			
	(1,1)	(1,2)	(2,1)		(1,1)	(1,2)	(2,1)
edge 4	0	0	1	edge 4	0	0	1

Example Network 2

10 samples					50 samples				
	(1,1)	(1,2)	(2,1)	(2,2)		(1,1)	(1,2)	(2,1)	(2,2)
edge 4	0	.5	.5	0	edge 4	0	.5	.5	0

100 samples					200 samples				
	(1,1)	(1,2)	(2,1)	(2,2)		(1,1)	(1,2)	(2,1)	(2,2)
edge 4	0	.5	.5	0	edge 4	0	.5	.5	0

Example Network 3

10 samples					50 samples				
	(1,1)	(1,2)	(2,1)	(2,2)		(1,1)	(1,2)	(2,1)	(2,2)
edge 2	.75	0	0	.25	edge 2	.5	0	0	.5
edge 9	0	.5	.5	0	edge 9	0	.5	.5	0

100 samples					200 samples				
	(1,1)	(1,2)	(2,1)	(2,2)		(1,1)	(1,2)	(2,1)	(2,2)
edge 2	.5	0	0	.5	edge 2	.5	0	0	.5
edge 9	0	.5	.5	0	edge 9	0	.5	.5	0

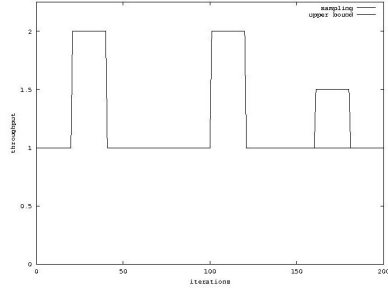
Example Network 4

10 samples					50 samples				
	(1,1)	(1,2)	(2,1)	(2,2)		(1,1)	(1,2)	(2,1)	(2,2)
edge 2	.5	0	0	.5	edge 2	.5	0	0	.5
edge 9	0	.5	.5	0	edge 9	0	.5	.5	0

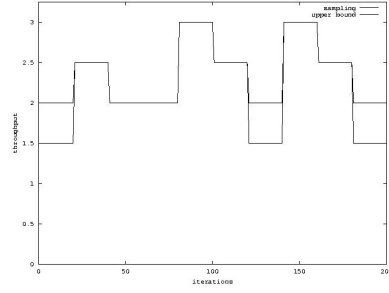
100 samples					200 samples				
	(1,1)	(1,2)	(2,1)	(2,2)		(1,1)	(1,2)	(2,1)	(2,2)
edge 2	.5	0	0	.5	edge 2	.5	0	0	.5
edge 9	0	.5	.5	0	edge 9	0	.5	.5	0

We find in all cases that 200 samples is sufficient for stability of the

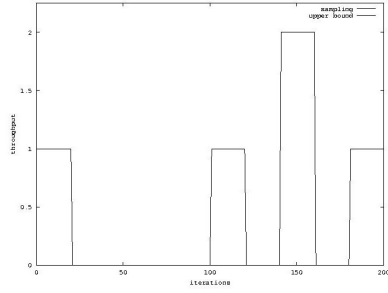
algorithm results. The following graphs show how the sampling algorithm performs in each example instance as compared to the known upper bound on the solution, calculated by solving the maximum concurrent flow problem directly for each change in capacity (occurring on every 10th iteration).



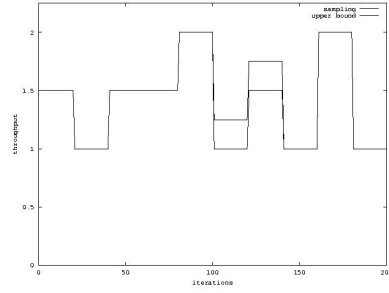
(a) Example Network 1



(b) Example Network 2



(c) Example Network 3



(d) Example Network 4

Figure 6: Sampling Algorithm Performance

In these instances, the sampling algorithm reaches 96.7% of the output of the upper bound for Example Network 1, 92.5% of the output of the upper bound for Example Network 2, 100% of the output of the upper bound for Example Network 3, and 96.6% of the output of the upper bound for Example

Network 4.

Recall the conservative estimate for the number of samples required,

$$\frac{1}{\prod_i \prod_{j=1}^{n_i} \min_{e \in P_{i,j}} \{p_e, 1 - p_e\}}$$

Using this estimate, Example Network 1 needs 8 samples, Example Network 2 needs 16 samples. and Example Networks 3 and 4 need 64 samples. Therefore, Example Network 1 will have $20 * 8 = 160$ more constraints than the original problem, Example Network 2 will have $30 * 16 = 480$ more constraints than the original problem, and Example Networks 3 and 4 will have $42 * 64 = 2688$ more constraints than the original problem. It is easy to see that even for small networks the optimization problem quickly becomes unwieldy.

4.4 Lower Bound Approximation

The goal of the lower bound approximation approach is to provide an approximation to the solution of the invariant allocation problem. The sampling algorithm provides a "hard-coded" solution to the invariant allocation problem; therefore empirical results for the lower bound approximation approach consist of comparisons to the results for the sampling algorithm.

4.4.1 Logarithmic Bound

The logarithmic lower bound approach was run for all example instances. The following charts show how the α values found in the logarithmic lower bound approximation approach compare to those found in the sampling algorithm. If the value differs from the values found in the sampling algorithm with 200 samples, then the value is in boldface.

Example Network 1

	(1,1)	(1,2)	(2,1)
edge 4	0	.5	.5

Example Network 2

	(1,1)	(1,2)	(2,1)	(2,2)
edge 4	0	.5	.5	0

Example Network 3

	(1,1)	(1,2)	(2,1)	(2,2)
edge 2	.5016	0	0	.4984
edge 9	.5017	0	0	.4983

Example Network 4

	(1,1)	(1,2)	(2,1)	(2,2)
edge 2	.5031	0	0	.4969
edge 9	0	.5031	.4969	0

If the α values are the same, clearly the average throughput remains the same as well. In Example Network 1, the long run average throughput (run for 200 samples) was .76 as compared to 1.05 for the sampling algorithm, a degradation of 28%. Because the network is not symmetrical, using the maximum multicommodity flow approach does not yield optimal results, as explored in greater detail in the research section. Network Examples 3 and 4 are within 10^{-3} of optimal results, thus they experience minimal degradation. This suggests that the algorithm is capable of finding the optimal result, but more precise parameters are required. It is also interesting to note that different types of optimization algorithms yield different results for the logarithmic bound. When run with an interior-point algorithm, the method yields values close to optimal, but when run with an active-set algorithm, Network Example 3 and 4 results converge to varying degrees from the optimal values. This suggests (as shown in the research section) that the logarithmic lower bound method is finding local maxima rather than a global maximum.

4.4.2 "Simple" Network Bound

The "simple" network lower bound approach was run for all example instances. The following charts show how the α values found in the "simple" network lower bound approximation approach compare to those found in the sampling algorithm. If the value differs from the values found in the sampling algorithm with 200 samples, then the value is in boldface.

Example Network 1

	(1,1)	(1,2)	(2,1)
edge 4	0	.5	.5

Example Network 2

	(1,1)	(1,2)	(2,1)	(2,2)
edge 4	0	.5	.5	0

Example Network 3

	(1,1)	(1,2)	(2,1)	(2,2)
edge 2	.5	0	0	.5
edge 9	0	.5	.5	0

Example Network 4

	(1,1)	(1,2)	(2,1)	(2,2)
edge 2	.5	0	0	.5
edge 9	0	.5	.5	0

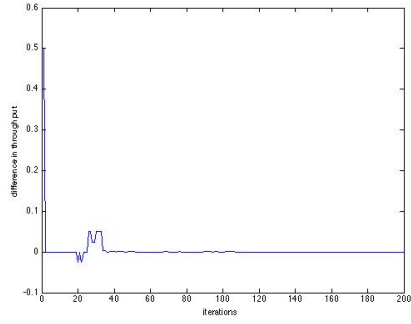
If the α values are the same, clearly the average throughput remains the same as well. In Example Network 1, the long run average throughput (run for 200 samples) was .76 as compared to 1.05 for the sampling algorithm, a degradation of 28%. Because the network is not symmetrical, using the maximum multicommodity flow approach does not yield optimal results. This issue was explored in greater detail in the research section.

4.5 Stochastic Supergradient Method

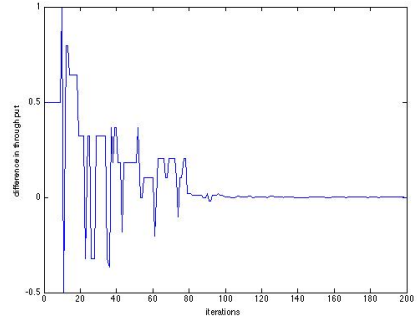
The goal of the stochastic supergradient method is to provide an approximation to the solution of the invariant allocation problem. The sampling algorithm provides a "hard-coded" solution to the invariant allocation problem; therefore empirical results for the stochastic supergradient method consist of comparisons to the results for the sampling algorithm. The stochastic supergradient method was run for all example instances. The following chart shows how many iterations it consistently took the stochastic supergradient method to find the α values within 10^{-3} from the sampling algorithm. This is based on an algorithm where a new capacity realization is sampled from in each iteration. The graphs show the convergence of the results to the sampling algorithm throughput by showing the difference between the sampling algorithm throughput and the stochastic supergradient method throughput in each iteration.

Number of Iterations Required for Convergence

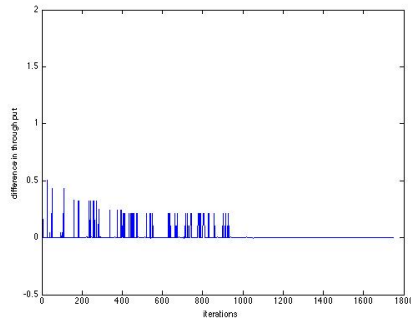
Example	1	2	3	4
	200	200	1750	1750



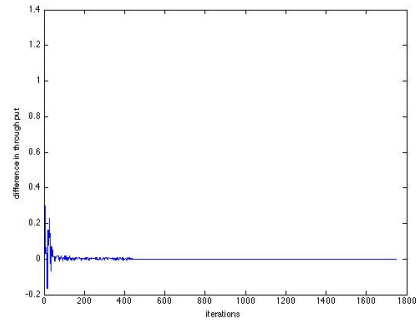
(a) Network Example 1



(b) Network Example 2



(c) Network Example 3



(d) Network Example 4

Figure 7: Convergence to Sampling Algorithm Results

Furthermore, in the following graphs one can see how the α values converge.

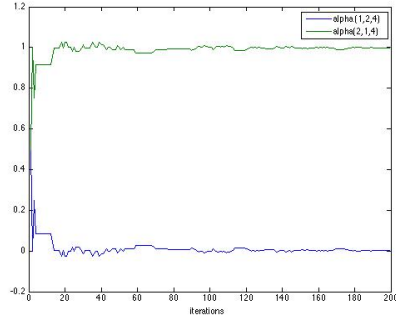


Figure 8: Example Network 1

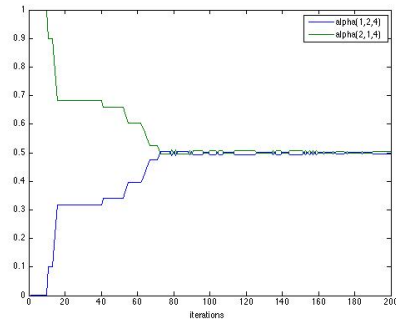


Figure 9: Example Network 2

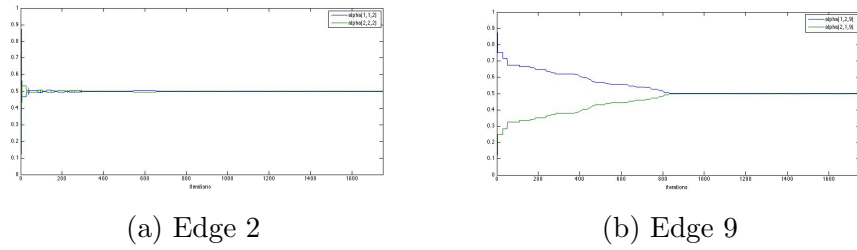
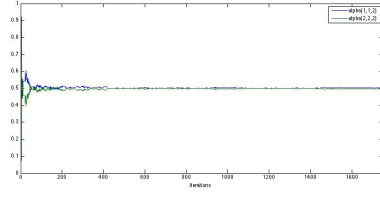
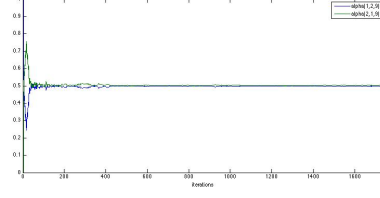


Figure 10: Example Network 3



(a) Edge 2



(b) Edge 9

Figure 11: Example Network 4

Although the stochastic supergradient method is not strictly a descent method, we can see that as the number of iterations increase, the α values generally move in the direction of the optimal solution.

5 Conclusions

We have explored different means for solving a network flow problem with multiple commodities and time-variant capacities in a distributed setting.

Adapting the standard approximation algorithms (FPTAP) for multicommodity network flow problems resulted in optimal solutions only when the transition rate was slower than the time to compute the logarithmic number of iterations needed to converge within some ϵ given the width of a problem ρ . Thus we sought solutions that could reach optimal solutions under these conditions by creating a problem that could be solved once and was robust to changes in capacity over time. The resulting problem, the invariant allocation problem, optimizes over a ratio of flow that is the fraction of an edge e 's capacity that is allocated to path j for commodity i , rather than over path flows. When tested through extensive sampling, this method yielded a throughput that was only 5% or less worse than the longterm throughput of solving directly in all example instances. Unfortunately, the objective function of this problem is difficult to assess directly because it is an expected value, so we sought other methods of approximating the solution to the invariant allocation problem.

The first method, the sampling algorithm, which samples from possible capacity realizations and then solves a large linear program, is effective in finding the optimal solution. However, the linear program becomes extremely large even with a small increase in network size, so the sampling method

is not feasible to use in a time-sensitive setting. Additionally, the method requires prior knowledge of the stochastic model, so it cannot be run in a distributed setting.

The second method, the lower bound approximation approach, sought to decrease the size of the optimization problem in the sampling algorithm by finding a tight lower bound on the objective function that would allow us to solve a simpler linear program. Unfortunately, it is difficult to find a lower bound on the objective function because we have an expected value of a minimum, and Jensen’s Inequality only provides an upper bound. We used the maximum multicommodity flow formulation as a hopeful approximation of the problem. This approach yielded two lower bound approximations. The first approximation, the logarithmic bound, uses the Chernoff-type bounds in [11] for a set of dependent random variables. The second approximation, the ”simple” network bound, uses a closed-form solution for the expected value for a simple network topology that is a lower bound for more complex networks. The ”simple” network bound found optimal solutions in all example instances except where the network was not symmetrical (i.e. $\sum_{j=1}^{n_i} \min_{e \in P_{i,j}} \{\alpha_{i,j,e} C_e\}$ is not the same for all i). This is due to the simplification of the objective function by using the maximum multicommodity flow formulation. The logarithmic bound experienced varying degradation of longterm throughput as compared to the optimal solution when using certain optimization methods because the objective function is not concave in all cases, so it is possible that local maxima will be found rather than

a global maximum. Although the lower bound approximation approach requires solving a much smaller linear program than the sampling algorithm, it still requires prior knowledge of the stochastic model, so it has limitations in a distributed setting. However, the "simple" network bound is still a viable choice in cases where networks are symmetrical or fairness is not a primary concern and the linear program can be solved prior to implementation.

The final method, the stochastic supergradient method, sought to find a solution that did not require prior knowledge of the stochastic model and could operate in a distributed setting. The stochastic supergradient method updated the decision variables in each iteration by using a sample to build a noisy, unbiased supergradient. This approach was previously only proven in cases where random variables were independent and identically distributed. We proved that the method can be used for Markov processes. We tested the method using Markov process as variables and successfully converged to the optimal solution within 2000 iterations for all example networks. In fact, this method offered quicker solutions than FPTAP. Since both are supergradient methods, they appear to have the same running time, but FPTAP must build a new solution for every capacity change while the stochastic supergradient method improves on a single solution in every iteration. The stochastic supergradient approach is the best overall approach to use in networks of this type. Therefore we built a distributed implementation plan for this method that performed well in all example networks.

5.1 Contribution

This work has been executed on behalf of Lincoln Labs and supports endeavors in mobile wireless networks, specifically for military and emergency response communications. Deliverables include results from this thesis document and code for the lower bound and stochastic supergradient methods that can take inputs of network size, connectivity, and a stochastic model to give α results for any network.

Furthermore, a journal publication is under preparation. This article will expound upon concepts and results from invariant allocation, the sampling algorithm, and the stochastic supergradient method. We are exploring publication in IEEE Transactions on: Communications; Information Theory; Mobile Computing; Systems, Man and Cybernetics; or Wireless Communications.

5.2 Future Work

Future work could include more extensive empirical results, a deeper theoretical exploration into the types of networks for which each method is suited, and further development and testing of distributed implementation.

The methods were tested on a limited number of small networks mostly due to the computational limitations as well as difficulty of verifying the accuracy of results. Future work could explore larger networks of more varied topology, including networks with more than two demands.

With more empirical results, one could hopefully glean the types of networks for which each method is most suitable. This could lead to a greater theoretical understanding of the methods as well. For example, through theoretical understanding of optimization methods and duality gaps, one could ascertain why particular optimization methods (interior-point vs active-set) appear to perform better for the logarithmic bound.

Although pseudocode has been developed for the distributed implementation of the stochastic supergradient method, this plan has gone through limited testing. A development of hard code and testing on example networks could further show the performance of the distribution implementation. More extensive literature review into information sharing could also help streamline the distribution implementation plan to improve speed.

References

- [1] Agarwal, Alekh, and J. Duchi. "The generalization ability of online algorithms for dependent data." (2011): 1-1.
- [2] Aldous, David, Lszl Lovsz, and Peter Winkler. "Mixing times for uniformly ergodic Markov chains." *Stochastic Processes and their Applications* 71.2 (1997): 165-185.
- [3] Andrews, M. "Optimization via communication networks." *42nd Annual Conference on Information Sciences and Systems*. 19-21 March 2008. 1205-1209.
- [4] Andrews, Matthew, Kyomin Jung, and Alexander Stolyar. "Stability of the max-weight routing and scheduling protocol in dynamic networks and at critical loads." *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. ACM, 2007.
- [5] Awerbuch, Baruch, and Rohit Khandekar. "Greedy distributed optimization of multi-commodity flows." *Distributed Computing* 21.5 (2009): 317-329.
- [6] Awerbuch, Baruch, and Tom Leighton. "A simple local-control approximation algorithm for multicommodity flow." *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*. IEEE, 1993.

- [7] Boyd, Stephen, Lin Xiao, and Almir Mutapcic. "Subgradient methods." *lecture notes of EE392o*, Stanford University, *Autumn Quarter* 2004 (2003).
- [8] Boyd, Stephen, and Almir Mutapcic. "Stochastic Subgradient Methods." *Notes for EE364b*, Stanford University, 2008.
- [9] Bertsekas, Dimitri, and John Tsitsik. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997
- [10] Cogill, Randy and Brooke Shrader. "2012 Year-end Report: Distributed Algorithms for Dynamic Networks." (2012).
- [11] Cogill, Randy. "Randomized Load Balancing with Non-Uniform Task Lengths." *Proceedings of the 2007 Allerton Conference on Communication, Control, and Computing*. Monticello, Illinois: 2007.
- [12] Dantzig, George B., and Philip Wolfe. "Decomposition principle for linear programs." *Operations research* 8.1 (1960): 101-111.
- [13] Duchi, John C., et al. "Ergodic mirror descent." *SIAM Journal on Optimization* 22.4 (2012): 1549-1578.
- [14] Garg, Naveen, and Jochen Knemann. "Faster and Simpler Algorithms for Multicommodity Flow and other Fractional Packing Problems." *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*. 1998.

- [15] Kamath, Anil, Omri Palmon, and Serge Plotkin. "Simple and fast distributed multi-commodity flow algorithm." Unpublished manuscript, Dec 1993.
- [16] Martins, Andre FT, et al. "Alternating Directions Dual Decomposition." (2012).
- [17] Nedic, Angelia, and Asuman Ozdaglar. "Distributed subgradient methods for multi-agent optimization." *Automatic Control, IEEE Transactions on* 54.1 (2009): 48-61.
- [18] Neely, Michael J., Eytan Modiano, and Charles E. Rohrs. "Dynamic power allocation and routing for time-varying wireless networks." *Selected Areas in Communications, IEEE Journal on* 23.1 (2005): 89-103.
- [19] Ram, S. Sundhar, A. Nedi?, and V. V. Veeravalli. "Distributed stochastic subgradient projection algorithms for convex optimization." *Journal of optimization theory and applications* 147.3 (2010): 516-545.
- [20] Roberts, Gareth O., and Jeffrey S. Rosenthal. "Geometric ergodicity and hybrid Markov chains." *Electron. Comm. Probab* 2.2 (1997): 13-25.
- [21] Shahrokhi, Farhad, and D.W. Matula. "The maximum concurrent flow problem." *Journal of the ACM*. 37.2. 1990. 318-334.
- [22] Shrader, B., Shake, T.H., Funk, J., Babikyan, A., and Worthen, A.P., "Routing and rate control for coded cooperation in a satellite-terrestrial network," *Military Communications Conference*. 2011. 735-740.

- [23] Trichakis, N., A. Zymnis, and S. Boyd. "Dynamic Network Utility Maximization with Delivery Contracts." *Proceedings IFAC World Congress*. Seoul: 2008. 29072912.
- [24] Wang, I-Lin. "Shortest Paths and Multicommodity Network Flows." Diss. Georgia Institute of Technology, 2003.