Coordinated Sampling sans Origin-Destination Identifiers: Algorithms, Analysis, and Evaluation

Vyas Sekar[1], Anupam Gupta[1], Michael K. Reiter[2], Hui Zhang [1]

Mar 4, 2009
CMU-CS-09-104

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1]⋆Carnegie Mellon University, [2]UNC-Chapel Hill

This work was supported in part by NSF awards CNS-0326472, CNS-0433540, and ANI-0331653.

Keywords: Network Monitoring, Submodularity, Sampling, Coordination, Approximation Algorithms

Abstract

Flow monitoring is increasingly used for a wide range of network security and anomaly detection applications. These applications require that flow monitoring infrastructures provide high flow coverage and be able to support fine-grained network-wide objectives. Coordinated Sampling (cSamp) is a recent proposal for improving the flow monitoring capabilities of ISPs to address these demands. In this paper, we address a key deployment impediment for cSamp-like solutions – the requirement that each router must determine the Origin-Destination (OD) pair of each packet it observes. We cast cSamp in a new framework called cSamp-T that enables us to apply powerful results from the theory of maximizing submodular set functions to build effective flow monitoring solutions in which each router works with *only local information*. We show that cSamp-T provides near-ideal performance in maximizing the total flow coverage in the network. Further, with a small amount of additional targeted provisioning or upgrading a small number of ingress routers to add OD-pair identifiers, cSamp-T obtains near-optimal maximization of the minimum fractional coverage across all OD-pairs. We demonstrate these results on a range of real topologies. From a practical perspective, these results are promising since they expand the applicability of cSamp-like solutions to ISPs where OD-pair identification is challenging and also provides an incremental deployment path for ISPs. Additionally, we believe that many of the techniques we develop here are more broadly applicable to other aspects of network management and measurement.

1 Introduction

Applications of flow monitoring in ISPs have far exceeded the scope of traditional traffic engineering and accounting applications [8]. Today, flow monitoring supports several critical network management tasks such as anomaly detection [23], identifying unwanted application traffic [6], understanding traffic structure at various granularities [41, 39], botnet analysis [28], and even forensic analysis [38]. These applications impose significantly greater demands on flow monitoring infrastructures: greater flow coverage (number of unique flows logged) and the ability to achieve network-wide flow measurement goals.

To meet these growing demands, recent work (e.g., [5, 4, 30]) articulates the case for network-wide rather than router-centric approaches for flow monitoring. We take one such proposal, namely Coordinated Sampling (cSamp) [30], as our starting point in this paper. We choose cSamp because compared to current solutions, it provides higher flow coverage, achieves fine-grained network-wide flow coverage goals, efficiently leverages available monitoring capacity on routers and minimizes redundant measurements, and naturally load balances responsibilities to avoid monitoring hotspots.

In order to simplify the underlying algorithmic formulations, cSamp assumes that each router on receiving a packet can immediately ascertain the Origin-Destination (OD) pair for the packet, specified by the ingress and egress routers. However, due to prefix-aggregation and multi-exit peers, interior routers in the network cannot identify the OD-pair given just the source and destination IP addresses. Thus, cSamp imposes two requirements: (i) modifications to packet headers to carry OD-pair identifiers, and (ii) upgrades to border routers to compute the OD-pair identifiers [11] for each packet. Both modifications present significant deployment barriers for many ISPs. Thus, while cSamp is an elegant architecture that has the potential to improve flow monitoring, in its current form it is an impractical solution with no immediate deployment path for ISPs today.

To address this impediment, in this work, we reformulate the problem of implementing a cSamp-like architecture into the scenario where OD-pair identifiers are not available. The goal of such an architecture, to which we refer as cSamp-T[1], is to realize the benefits of cSamp and at the same time be immediately deployable. An immediate consequence of this reformulation is that the known algorithms [30] for efficiently maximizing either the total flow coverage or minimum fractional coverage across all OD-pairs, no longer apply. In fact, we show that these problems are NP-hard. Consequently, a central challenge is to develop algorithms for efficiently computing sampling strategies so as to optimize these measures, either exactly or approximately.

In this paper, we present substantial progress toward meeting this challenge. For the measure of total flow coverage (total number of unique flows logged), we notice that the objective function is *submodular*. This is important because even though it is hard to find an exact optimal solution, we can implement efficient greedy algorithms with good approximation guarantees that leverage this submodularity property. We borrow and extend results from a rich theory of optimizing submodular functions subject to budget constraints (e.g., [12, 25, 19, 21]) to this specific application. We show that on realistic topologies, this approach yields near optimal total flow coverage.

[1]cSamp-T denotes cSamp minus Tags for OD-pairs

The minimum fractional coverage objective (i.e., the minimum across all OD-pairs of the fraction of flows logged per OD-pair) is not submodular, however, and so does not inherit these approximation guarantees with a greedy approach [21]. Moreover, on realistic topologies the greedy approach performs poorly. So, in this case we turn to examining the additional resources needed in order to obtain good performance. We consider two practical scenarios for ISPs to alleviate this concern: (a) augmenting targeted routers with more memory resources and (b) incremental deployment of cSamp by upgrading a small subset of border routers with the functionality to compute OD-pair identifiers and add them into packet headers. Our results in this direction are promising: we show that a few such router upgrades can significantly boost the minimum fractional coverage obtained in realistic topologies.

cSamp-T thus makes cSamp-like solutions more immediately deployable by relaxing the dependence on the OD-pair identifiers. Further, it provides an incremental deployment path for ISPs to transition their flow monitoring infrastructures to cSamp, while in the interim partial deployment phase it provides performance comparable to cSamp. We also believe that many of the specific algorithmic techniques and heuristic extensions we develop here (e.g., applying results from the theory of submodular set maximization, intelligent resource provisioning, hybrid cSamp/cSamp-T deployment) can be more broadly applied to other aspects of network management and measurement.

2 Background and Motivation

Why cSamp: Applications of flow monitoring continue to grow and already include several anomaly detection and security applications (e.g., [23, 6, 41, 39, 38, 28]). Motivated by this trend, Sekar et al. [30] identify five main goals for flow monitoring solutions: (i) provide high flow coverage (i.e., log as many flows as possible) to support the security applications that need a fine-grained understanding of “who-talked-to-whom”, (ii) minimize redundant reports (i.e., use router resources efficiently and reduce the overhead in processing duplicate measurements), (iii) satisfy network-wide flow monitoring objectives (e.g., specify some subsets of traffic as more important than others or ensure fairness across different subsets), (iv) work within (possibly heterogeneous) router resource constraints, and (v) be general enough to support a wide spectrum of flow monitoring applications.

Synthesizing arguments from several previous papers [16, 22, 5, 34, 4, 31, 7], Sekar et al [30] argue that these goals necessitate three design choices: using flow sampling instead of packet sampling to avoid the well-known biases of packet sampling against small flows [16], coordinating the routers to leverage available monitoring resources efficiently and to avoid redundant sampling, and a network-wide framework for assigning flow monitoring responsibilities to routers to optimally achieve ISP objectives.

Description of cSamp: For completeness, we provide a brief overview of the cSamp approach. We refer the reader to [30] for further details. The inputs to cSamp are the flow-level traffic matrix (number of flows per OD-pair), router-level path(s) for each OD-pair, the resource constraints of routers, and a ISP objective function (specified in terms of the fractional flow coverages per OD-pair). The output is a set of *sampling manifests* specifying the monitoring responsibility of each router in the network. The sampling manifest in cSamp is a set of tuples of the form $\langle OD, [start, end]\rangle$, where $[start, end] \subseteq [0,1]$ denotes a hash range.

Each router's sampling algorithm is as follows. For each packet, the router first identifies the OD-pair from the packet header. Next, it computes a hash on the flow 5-tuple (*srcIP, dstIP, srcport, dstport, protocol*) and checks if the hash value lies in the hash range assigned to it for the OD-pair (the function HASH returns a value in the range $[0,1]$). Each router maintains a $Flowtable$ of the flows it is currently logging. If the packet has been selected, the router either creates a new entry (if none exists) or updates counters for the corresponding entry in the $Flowtable$.

The key idea is that all routers are bootstrapped with the same hash function but are assigned disjoint hash ranges per OD-pair. This coordinates the sampling actions of routers in the network. Coordination makes it easy to achieve network-wide flow coverage goals in terms of the per OD-pair coverages and also ensures that the sets of flows sampled by different routers do not overlap.

cSamp formulation: Each OD-Pair OD_i ($i = 1, \ldots, M$) is characterized by its router-level path P_i and T_i, the number of distinct IP-level flows in a measurement interval (e.g., five minutes).[2] Each router R_j ($j = 1, \ldots, N$) is primarily constrained by the available *memory* for maintaining per-flow counters in SRAM [10]; L_j denotes the number of flows R_j can record and report in a given measurement interval.

d_{ij} denotes the fraction of flows of OD_i that router R_j logs. (If R_j does not lie on path P_i, then the variable d_{ij} will not appear in the formulation.) For $i = 1, \ldots, M$, let C_i denote the fraction of flows on OD_i that is logged.

The specific goal in [30] is a two-step objective. First, the largest possible minimum fractional coverage per OD-pair $\min_i\{C_i\}$ subject to the resource constraints is found. Next, this value is used as the parameter α to the linear program shown below in (4) and the total flow coverage $\sum_i (T_i \times C_i)$ is maximized.

$$
\begin{aligned}
&\text{Maximize} \textstyle\sum_i (T_i \times C_i), \quad \text{subject to} \\
&\forall j, \quad \textstyle\sum_{i:R_j \in P_i} (d_{ij} \times T_i) \leq L_j && (1) \\
&\forall i, \quad C_i = \textstyle\sum_{j:R_j \in P_i} d_{ij} && (2) \\
&\forall i, \forall j, \quad d_{ij} \geq 0 && (3) \\
&\forall i, \quad \alpha \leq C_i \leq 1 && (4)
\end{aligned}
$$

The solution $d^* = \{d^*_{ij}\}$ to this two-step procedure yields the optimal sampling strategy. Next, this solution is mapped via a simple algorithm into the *sampling manifests* specifying the flow monitoring responsibility for each router.

Assumptions in cSamp: There are three main assumptions: (i) a centralized module for assigning router responsibilities that has access to routing and traffic matrices, (ii) routers implement hash-based flow sampling, and (iii) routers obtain OD-pair information from packet headers.

[2]For simplicity, we assume that each OD-pair has a single routing-level path. It is easy to extend the framework to accommodate multi-path routing [30].

The first two assumptions are feasible within current technological and operational realities. First, centralization is viable if the router configurations are generated in a reasonable amount of time (say at most 1-2 minutes). Further, recent trends show that ISPs increasingly favor centralization of the network management functions [2, 14] and that routing and traffic matrices are typically already available [11, 40]. The second assumption that routers support hash-based flow sampling is also feasible within capabilities available today. The requirements on such hash functions are quite simple [32, 7] (e.g., no strong cryptographic guarantees) and thus they are amenable to fast hardware implementations [29]. Further, routers already implement hardware hash functions for other tasks. Flow sampling requires flow table lookups for each packet; the flow table, therefore, needs to be implemented in fast SRAM. Prior work has shown that maintaining such counters is feasible [10, 18]. For simplicity, cSamp assumes that the flow counters are maintained in SRAM and the amount of SRAM is the resource constraint that determines the number of flows a router can log.

The assumption that routers can obtain OD-pair identifiers simplifies cSamp's design and makes the optimization problem theoretically tractable. Specifically, (2) implicitly assumes that the hash-ranges assigned to different routers for the same OD-pair are non-overlapping. Thus, the coverage of each OD-pair is simply the sum of the fractional coverages of the routers on the path. If OD-pair identifiers were not available, this would no longer hold. As we argue next, for many ISPs this assumption is not practical.

Challenges in OD-pair identification: Obtaining OD-pair information is quite challenging for many ISPs today. First, it requires routers to be aware of OD-pair identifiers. This may require ISPs to migrate to MPLS-style routing. Second, routers cannot determine the OD-pair based on the IP header and local routing information alone. For example, in the case of traffic destined to a multi-exit peer (i.e., a neighboring AS with which an ISP peers at multiple peering points), prefix information alone is not sufficient to determine the exact egress. To complicate matters further, interior routers only see aggregated prefix information; ingress routers are in a better position to identify the egress when a packet first enters the network. Thus, cSamp assumes that ingress routers explicitly add OD-pair identifiers to packet headers. This leads to another limitation – it imposes additional computational effort on border routers (e.g., replicating some of the routing logic to resolve the egress router) and requires modifications to packet headers.

3 cSamp-T: Problem Statement

Motivating question: The above challenges in OD-pair identification bring us to the motivating question for our work: Can we implement a cSamp-like approach without requiring OD-pair identifiers? Intuitively, we want to specify each router's sampling manifest at a *much coarser granularity* relying only on *local information* rather than the global OD-pair identifiers, while still achieving the coverage guarantees of cSamp. We call this new approach cSamp-T.

cSamp-T eliminates the need for ISPs to (a) upgrade border routers with additional intelligence for OD-pair identification, (b) modify packet headers to accommodate these identifiers, and (c) overhaul their routing infrastructures. Thus, cSamp-T makes the benefits of cSamp-like solutions available to network operators without incurring the overhead for OD-pair identification that

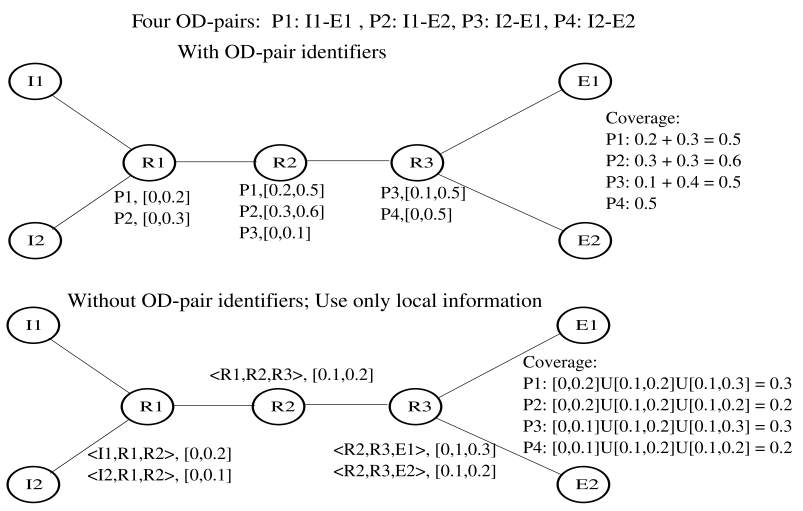


Figure 1: Example topology showing the intuition behind the cSamp-T approach

cSamp imposes.

High-level approach: The key insight behind the cSamp-T approach is to only use *local* information at each router to specify the router's sampling responsibilities. The coverage of each OD-pair is obtained by "stitching" together the coverages provided by each router on the path.

Consider the example shown in Figure 1 with 2 ingresses, 2 egresses, and 4 OD-pairs P1–P4. The top-half shows a cSamp configuration; OD-pair identifiers are available and each router's responsibilities are in terms of hash-ranges per OD-pair and for each OD-pair the ranges on the routers on its path are non-overlapping.

The bottom-half of Figure 1 shows a scenario where routers cannot obtain OD-pairs. The sampling manifests are specified based on just local information; each router is assigned a *hash-range per router 3-tuple* consisting of the previous hop, current router, and the next hop. Note that for each packet, a router can ascertain the previous hop and next hop just based on local information (e.g., the interface the packet arrives on and the next hop router determined by the routing table). The coverage for each OD-pair will then be the *union* of the ranges assigned to its constituent path-segments (the 3-tuples on each path in this example).

This example demonstrates two key differences between cSamp and cSamp-T. First, the sampling responsibilities are specified using locally available information rather than global OD-pair identifiers. Second, the coverage for each OD-pair is no longer simply the sum of the coverage of each router on the path; it is the union of the ranges assigned to the routers on the path.

Now, how do we assign sampling responsibilities in cSamp-T to maximize specific flow coverage objectives while operating within each router's resource constraints? The following sections present a formal framework to address this.

Problem Formulation for cSamp-T: We borrow two assumptions from cSamp: (a) sampling responsibilities are generated at a centralized module with access to routing and traffic matrices and (b) routers implement hash-based flow sampling using SRAM counters and the amount of SRAM is the primary resource constraint on the number of flows a router can log. As discussed earlier, both are reasonable assumptions. Next, we discuss the design of the centralized logic for

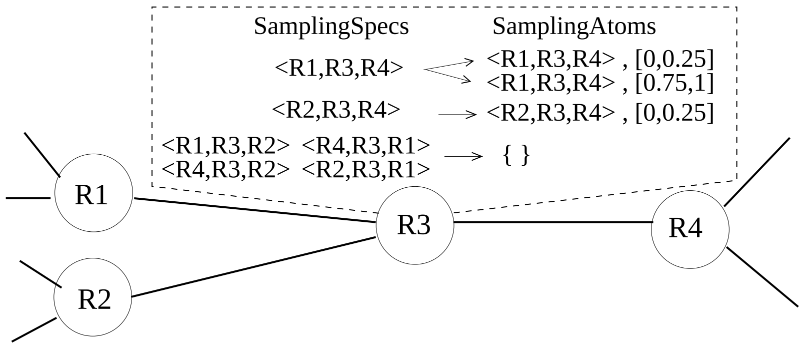


Figure 2: Example to illustrate the definitions showing the SamplingSpecs and assigned SamplingAtoms at router R3.

assigning sampling responsibilities in the absence of OD-pair identifiers.

We first define the notion of a *SamplingSpec* to capture the granularity at which each router's sampling decisions are made. For the current discussion, the SamplingSpecs are three-tuples of router identifiers $\langle R_{j_1}, R_{j_2}, R_{j_3} \rangle$ that appear contiguously on some path in the network, and so in particular R_{j_1} and R_{j_3} are neighbors of R_{j_2}. Let a_k denote a generic SamplingSpec in our system.

The notation $a_k \in P_i$ captures the idea of a SamplingSpec being on the path P_i for OD_i.[3] For example, if the path P_i uses routers $\cdots, R_{j_1}, R_{j_2}, R_{j_3}, \cdots$ in that order, then the SamplingSpec $a = \langle R_{j_1}, R_{j_2}, R_{j_3} \rangle \in P_i$. This is a natural extension similar to the notion of a router R_j being on path P_i. We use $t_k = \sum_{i: a_k \in P_i} T_i$ to denote the total traffic that traverses a_k. Our framework maps SamplingSpecs to routers in a many-to-one fashion; we denote the set of SamplingSpecs assigned to R_j by R_j.specs. In this way, R_j is assigned sampling responsibilities corresponding to all $a_k \in R_j$.specs. In this paper, if $a_k = \langle R_{j_1}, R_{j_2}, R_{j_3} \rangle$, then $a_k \in R_{j_2}$.specs.

From the above discussion, it is clear that if $R_j.\text{specs} \ni a_k$, then R_j is in a position to log (some or all) of the traffic on paths $P_i \ni a_k$. But which fraction should it log? To this end, if the entire traffic corresponding to a_k is mapped to points in the unit interval $[0, 1]$ (say, by hashing) then the router will be responsible for some subset of $[0, 1]$. In particular, we discretize $[0, 1]$ into $\frac{1}{\delta}$ equal-sized intervals of length δ $h_l = [(l-1)\delta, l\delta]$, and assign to a_k some of these δ-intervals.

We formalize this by creating a set of *SamplingAtom*s. A SamplingAtom is a pair $\langle a_k, h \rangle$, where a_k is a SamplingSpec and $h \subseteq [0, 1]$ is a "hash-range"—a subset of the unit interval of length δ. For any SamplingAtom, $g_{kl} = \langle a_k, h_l \rangle$, if $a_k \in R_j$.specs, then router R_j will log the flows that traverse a_k such that the hash of the flow falls in h_l. We use $h(g_{kl})$ as a shortcut for the hash-range associated with g_{kl}.

EXAMPLE: Figure 2 illustrates the above definitions with an example. $R3$ has three SamplingSpecs in the forward direction (and three similar SamplingSpecs in the reverse direction): $\langle R1, R3, R4 \rangle$, $\langle R1, R3, R2 \rangle$ and $\langle R2, R3, R4 \rangle$. $R3$ is assigned three SamplingAtoms, two for $\langle R1, R3, R4 \rangle$, one for $\langle R2, R3, R4 \rangle$, and none for $\langle R1, R3, R2 \rangle$. Say $\delta = 0.25$. Consider paths of the form $\{.., R1, R3, R4, ..\}$ (there may be many such paths). $R3$ will log all flows along these paths whose hashes fall either in the range $[0, 0.25]$ or $[0.75, 1]$, and flows along paths of the form $\{.., R2, R3, R4, ..\}$ such that the hash of the flow falls in the range $[0, 0.25]$.

[3]Since this notion of "on-path"-ness is quite general, our approach works even in the case of multi-path routing.

Notation	Explanation
M	Number of OD-pairs
N	Number of routers
OD_i	OD-pair i
C_i	Fraction of flows on OD-pair i covered
R_j	Router j
L_j	Available resources on R_j
$Load_j$	Total monitoring load on R_j
a_k	SamplingSpec k
R_j.specs	set of SamplingSpecs on R_j
t_k	Total traffic traversing SamplingSpec a_k
g_{kl}	SamplingAtom l on a_k
$\widehat{g_{kl}}$	an assigned or selected SamplingAtom
$h(g_{kl})$	hash-range $\subseteq [0,1]$ in SamplingAtom g_{kl}

Table 1: Notation in the problem statement

Measures of Goodness: Given a set of assigned SamplingAtoms, $\{\widehat{g_{kl}}\}$, the *fractional coverage* for OD_i is as follows. The coverage due to one particular SamplingSpec $a_k \in P_i$ is $\cup_l \; h(\widehat{g_{kl}}) \subseteq [0,1]$, and hence the total coverage is

$$\text{coverage } C_i = \left|\bigcup_{a_k \in P_i} \bigcup_l \; h(\widehat{g_{kl}})\right| \tag{5}$$

Here, given an interval $S \subseteq [0,1]$, we use $|S|$ to denote the fraction of the unit interval covered by this subset. Note that the coverage for a path is the *union* of the assigned hash-ranges across all the constituent SamplingSpecs – if the *same* hash-range is assigned to several SamplingSpecs along a path, then the same set of flows gets sampled and we do not get any extra coverage.

The *monitoring load* on a router is given by summing, over all SamplingSpecs $a_k \in R_j$.specs, the portion of the traffic through a_k that R_j logs:

$$Load_j = \sum_{a_k \in R_j.\text{specs}} t_k \times |\bigcup_l \; h(\widehat{g_{kl}})| \tag{6}$$

Given the C_is for the various OD-pairs, the specific functions we consider are the *total traffic coverage* $f_{tot} = \sum_i T_i C_i$, and the *minimum fractional coverage* $f_{min} = \min_i C_i$. Formally, the goal of our algorithms is to obtain the set of assigned SamplingAtoms $\{\widehat{g_{kl}}\}$ such that we maximize either f_{tot} or f_{min}, while operating within the router resource constraints (i.e., $Load_j \leq L_j$ for all j). We choose these specific objective functions because of their use in cSamp [30]; our framework can accommodate a wider range of objective functions specified as convex combinations of the C_i values.

The maximization problem: We can rewrite the above maximization problems as follows. Consider a "ground set" $\mathcal{V}$ which contains as its elements all possible SamplingAtoms: i.e., $\mathcal{V} = \{\langle a_k, h_l \rangle$ *for all possible SamplingSpecs* a_k *and all* $\frac{1}{\delta}$ *hash-ranges* $h_l\}$. Suppose a subset $S \subseteq \mathcal{V}$ of these SamplingAtoms are chosen and assigned to their corresponding routers. These give us the fractional coverages defined by (5) and router loads given by (6). Now, f_{tot} or f_{min} can be viewed as functions from subsets of $\mathcal{V}$ to the reals. The problem is to select the *optimal* $S^* \subseteq \mathcal{V}$, satisfying $Load_j \leq L_j$, that maximizes f_{tot} or f_{min}.

Exact Solutions are Hard: Finding the optimal S^* to maximize f_{tot} or f_{min} subject to the load constraints on routers is NP-hard. Appendix A demonstrates the hardness via a reduction from the 3-SAT problem. Moreover, it is infeasible for practical system sizes. For example, we cast the problem into an integer linear programming (ILP) formulation by assigning 0-1 indicator variables for each g_{kl} to denote whether it is "assigned" or not. Even on the Internet2 topology with just 11 routers, the commercial solver `CPLEX` did not converge to a solution after a day. It is because of this intractability of solving the problem exactly that we resort to greedy approximations. However, as we will see, our algorithms will yield results that compare favorably to the original cSamp performance.

4 Submodularity and Algorithms

Overview and Intuition: In the previous section, we saw that obtaining exact solutions for maximizing the total coverage or the minimum fractional coverage in the cSamp-T framework is hard. Fortunately, as we will see in the next sections, there are efficient practical algorithms to obtain the sampling strategies in cSamp-T. The key insight is that the coverage functions have a natural "submodularity" property (defined next) which allows us to apply powerful results from the theory of maximizing submodular set functions to our context. This is particularly promising, since submodularity implies that the greedy algorithm yields a constant-factor approximation [12].

More specifically, the coverage functions are "submodular" and the memory constraints at each router are "knapsack" constraints; our problem is then equivalent to the problem of maximizing submodular functions subject to knapsack constraints. We give theoretical bounds (Appendix B) and also show that the greedy algorithms work very well in practice. We also give results for maximizing f_{min} using algorithms for max-min submodular maximization [21].

4.1 Submodularity

Definition: A function $F : 2^{\mathcal{V}} \rightarrow \Re$ mapping subsets of a ground set $\mathcal{V}$ to the reals is *submodular* if for all sets $S \subseteq S' \subseteq \mathcal{V}$, all elements $s \in \mathcal{V}$,

$$F(S \cup \{s\}) - F(S) \geq F(S' \cup \{s\}) - F(S')$$

i.e., *the marginal benefit obtained from adding s to a larger set is smaller* [12]. This captures the intuitive property of diminishing returns. The function F is *monotone (nondecreasing)* if $\forall S \subseteq S', F(S) \leq F(S')$.

Submodular set maximization: The goal is to pick a subset $S \subset \mathcal{V}$ maximizing $F(S)$; what makes this problem hard is that we also have a "budget" constraint of the form $c(S) \leq B$; i.e., given "costs" $c(s)$ for all $s \in \mathcal{V}$, the total cost $c(S) := \sum_{s \in S} c(s)$ of elements picked in set S cannot exceed the "budget" B. This submodular maximization problem is NP-hard [12], but good approximation guarantees are known. In particular, the algorithm specified in Figure 3 either greedily picks elements that give the greatest marginal benefit and do not violate the budget constraints,

```
SubmodularGreedy($F, \mathcal{V}, cbflag, B$)
    // $F : 2^{\mathcal{V}} \rightarrow \Re$ submodular, $B$ is total budget
    // if $cbflag$ is true use benefit/cost instead of benefit
1   $S \leftarrow \emptyset$
2   while ($\exists s \in \mathcal{V} \setminus S : c(S \cup \{s\}) \leq B$) do
3       for $s \in \mathcal{V} \setminus S$ do
4           $norm \leftarrow ((cbflag = \mathbf{true})\ ?\ c(s)\ :\ 1)$
5           $\delta_s \leftarrow \frac{F(S \cup \{s\}) - F(S)}{norm}$
6       $s^* \leftarrow \mathbf{argmax}_{s \in \mathcal{V} \setminus S} \delta_s$
7       $S \leftarrow S \cup \{s^*\}$
8   return $\langle S, F(S) \rangle$
```
Figure 3: Basic greedy algorithm

or greedily picks the elements that give the maximum marginal benefit *per unit element-cost* (depending on whether $cbflag$ is true or false), as long as the budget is not violated. It is well-known that the better of these two algorithms is a constant factor approximation algorithm [37].

4.2 Application to cSamp-T

It is easy to check the coverages C_i viewed as a functions from $\mathcal{V} = SamplingAtoms \rightarrow \Re$ are monotone submodular functions, and hence so is their weighted sum $f_{tot} = \sum T_i C_i$.

Budget constraints in cSamp-T: The budget constraints in cSamp-T come from the bounds on router load. To model router load, we need a knapsack constraint $Load_j \leq L_j$ for each router R_j. A naive approach is to consider the cSamp-T problem as a submodular set maximization problem with multiple knapsack constraints. This naive approach yields a $O(N)$ approximation, where N is the number of routers. This is clearly undesirable, especially for large networks. Specifically, since each $SamplingAtom$ contributes to the load on exactly one router, this results in a collection of *non-overlapping* knapsack constraints. We call the resulting problem *submodular function maximization subject to partition-knapsack constraints*. (Each "partition" corresponds to a different router, and the "knapsack" comes from the load constraint for that router). In Appendix B we show that a modified greedy algorithm—an extension of one from Figure 3—gives a constant-factor approximation.

Maximizing f_{tot}: To match the theoretical guarantees [37] (see Appendix B), we run two separate invocations of the greedy algorithm—with and without the benefit-cost flag set to true, and return the solution with better performance. In practice, both have similar performance (Section 6.1).

Maximizing f_{min}: To maximize f_{min}, we need to go from one submodular function F to many submodular functions $F_1, F_2, \ldots, F_M$—in our case, these are the fractional coverages $C_1, \ldots, C_M$. The problem is now to pick $S \subseteq \mathcal{V}$ to (approximately) maximize $F^{\min}(S) = \min_i F_i(S)$, the *minimum* value across these different functions. This new function $F^{\min}$ is no longer submodular; indeed, obtaining any non-trivial approximation guarantee for this max-min optimization problem is NP-hard [21]. However, we can give an algorithm to maximize $F^{\min}$ when we are allowed

```
GREEDYMAXMIN($F_1, \ldots, F_M, \epsilon, \mathcal{V}, B, \gamma$)
      // Maximize $\mathbf{min}_i\{F_i\}$
      // $\forall i,\ F_i : 2^{\mathcal{V}} \rightarrow [0,1]$ is submodular
 1    $\tau_{lower} \leftarrow 0$; $\tau_{upper} \leftarrow 1$
 2    while ($\tau_{upper} - \tau_{lower} > \epsilon$) do
 3        $\tau_{current} \leftarrow \frac{\tau_{upper}+\tau_{lower}}{2}$
          // Define the modified objective function
 4        $\forall i, \hat{F}_i \equiv \mathbf{min}(F_i, \tau_{current})$; $\hat{F} \equiv \sum_i \hat{F}_i$
          // Run greedy without budget constraints
 5        $B_{used} \leftarrow$ SUBMODULARGREEDY($\hat{F}, \mathcal{V}, \mathbf{true}, \infty$)
          // Compare resource usage
 6        if MAXUSAGE($B_{used}, B$) $> \gamma$ then
            // $\tau_{current}$ is infeasible, reduce upper bound
 7          $\tau_{upper} \leftarrow \tau_{current}$
 8        else
            // $\tau_{current}$ is feasible, increase lower bound
 9          $\tau_{lower} \leftarrow \tau_{current}$
10    Return $\tau_{lower}$
```

Figure 4: Maximizing the minimum of a set of submodular functions with resource augmentation

to exceed the budget constraint by some factor [21]. Formally, if S^* is an optimal set satisfying budget constraints, the algorithm in Figure 4 finds a set S with $F^{\min}(S) \geq F^{\min}(S^*) - \epsilon$ but which exceeds the budget constraints by a factor of γ, where $\gamma = O\big(\log(\frac{1}{\epsilon}\sum_{v \in \mathcal{V}} F_i(v))\big)$.

The key idea is this: the modified objective function $\hat{F}_\tau = \sum_i^M \min(F_i, \tau)$ is submodular. For any τ, $\hat{F}_\tau$ has the property that its maximum value is $M \times \tau$ and at this maximum value $\forall i, F_i \geq \tau$. Running the greedy algorithm assuming no resource constraints always gives a set such that the actual resource usage at router R_j is at most $\gamma \times Load_j$. Notice that this holds for all τ, and in particular, for the optimal value $\tau^* = F^{\min}(S^*)$. Since the optimal τ^* is not known, the algorithm in Figure 4 uses binary search over τ.

Router algorithm: Given a solution to the problem of maximizing f_{tot} or f_{min}, Figure 5 shows each router's sampling algorithm. Note that the router no longer requires the OD-pair information for a packet; it only requires the coarser SamplingSpec information which can be immediately discerned using only the packet headers and other local information (e.g., what interface the packet arrives/leaves on). We allow for the *Ranges* for each SamplingSpec to be a set of non-contiguous hash ranges; thus, the router samples the packet if the hash value falls in *any* of the ranges.

4.3 Practical Issues

Reducing computation time: The computation time of the algorithm of Figure 3 can be reduced by using the insight that for each element $s \in \mathcal{V}$, the marginal benefit obtained by picking s

```
cSamp-T_Router($pkt$, $Manifest$)
    // $Manifest = \{\widehat{g_{kl}} = \langle a, h \rangle\}$
1   $a \leftarrow$ GetSamplingSpec($pkt$)
    // $Ranges$ is a set of hash-range blocks
2   $Ranges \leftarrow$ GetRanges($a$)
    // Hash returns a value in $[0, 1]$
3   $h_{pkt} \leftarrow$ Hash(FlowHeader($pkt$))
    // Log if the hash value falls in one of the ranges
4   if $h_{pkt} \in Ranges$ then
5       Create an entry in $Flowtable$ if none exists
6       Update byte and packet counters for the entry
```

Figure 5: Implementing cSamp-T on router R_j

decreases monotonically across iterations of the greedy algorithm [25, 13]. Thus, we can use a *lazy evaluation* algorithm [25, 13]. The main intuition behind lazy evaluation is that not all δ_s values need to be recomputed in Figure 3 (Step 5); only a smaller subset of that are likely to affect the choice of s^* in Step 6 need to be computed. We omit further details of this algorithm for brevity and refer the reader to the references [25, 13]. We can replace all instances of the procedure call SubmodularGreedy with the lazy evaluation version. Section 6.2 shows that this reduces the computation time by more than an order of magnitude.

Generalizing SamplingSpecs: We assumed that the SamplingSpecs are defined at the granularity of router three-tuples. Note, however, that the greedy algorithms and the per-router sampling algorithm are quite generic; they do not depend on SamplingSpecs being router three-tuples. We can generalize the algorithms and results to different notions of a SamplingSpec. For example, the SamplingSpecs can be router identifiers (in which case the router applies the same sampling decisions to every path passing through it), or router two-tuples (previous hop and current router), or incorporate IP-prefix information as well.

Practical issues in discretization: Section 3 defined discretization intervals δ such that $g_{kl} = \langle a_k, [(l-1)\delta, l\delta] \rangle$, for values $l \in \{1, \ldots, \frac{1}{\delta}\}$. There are two practical issues to note here. First, we can make the width δ arbitrarily small; there is a tradeoff between (potentially) better coverage vs. the time to compute the solution. In our evaluations, we fix $\delta = 0.02$ since we find that it works well in practice. Secondly, instead of considering $\frac{1}{\delta}$ disjoint intervals, we can also consider the $\frac{1}{\delta}^2$ hash-ranges of the form $[m\delta, (m+n)\delta]$ to make assignments as contiguous as possible. This increases the computation time quadratically without providing any additional coverage benefits. In practice, we avoid this and instead run a simple merge procedure (Section 6.3) to compress the sampling manifests.

5 Heuristic Extensions

While the theoretical guarantees for f_{tot} are encouraging, achieving good performance for f_{min} is less promising. The theoretical results suggests that the resource augmentation γ required to obtain any non-trivial guarantee is quite high.

In this section, we consider three practical extensions to improve the performance for f_{min}. The first extension uses a targeted provisioning heuristic to use fewer resources in aggregate. The second extension evaluates an incremental deployment scenario where a small subset of ingress routers can be upgraded to add OD-pair identifiers. We present these in the specific context of the f_{min} objective. However, these two techniques we develop for targeted provisioning and partial marking can be more generally applied to other network-wide objectives where the greedy algorithm performs poorly. We also consider an alternative submodular objective function for getting better performance for f_{min}

5.1 Intelligent Provisioning

The theoretical bounds from the previous section assume that each router in the network is uniformly given γ times more resources. In practice, this may be quite excessive since it might be very expensive to add γ times more SRAM capacity to each router. An interesting question is whether it is possible get better performance if we can add more memory on routers intelligently – instead of upgrading all routers, we seek to augment a smaller subset of routers and still get similar performance. The rationale behind the approach is that it may suffice to upgrade a small number of heavily loaded routers.

Problem $provisioning$**:**

$$\text{Maximize } \mathbf{min}_i\ C_i, \quad \text{subject to}$$
$$\forall j, \textstyle\sum_{k: a_k \in R_j.\mathsf{specs}} u_k \times t_k \leq L_j \tag{7}$$
$$\textstyle\sum_j L_j \leq Budget \tag{8}$$
$$\forall j, LB_j \leq L_j \leq UB_j \tag{9}$$
$$\forall i, C_i = \textstyle\sum_{k: a_k \in P_i} u_k \tag{10}$$
$$\forall k, u_k \geq 0 \tag{11}$$
$$\forall i, C_i \leq 1 \tag{12}$$

To address this question, we consider the above provisioning problem. The network operator specifies a total budget of memory resources to be distributed across different routers (e.g., defined by a total monetary budget and the cost of SRAM). Each router R_j has a lower bound (LB_j) for the default memory configuration and a technology upper bound (UB_j) on the maximum amount of memory that can be provisioned. (There are natural technological limits on the amount of fast SRAM that can be added to linecards [36].) The inputs to the problem are the total memory budget $Budget$, LB_j, and UB_j. The output is the specific allocation of resources to routers to optimize f_{min}.

However, it is difficult to model the coverage C_i of each OD-pair provided by the greedy algorithm under a given set of resources. Thus, we make a simplifying assumption that the hash ranges (represented by the variables u) allocated across the different SamplingSpecs on a given path are mutually non-overlapping. This allows us to model C_i as simply the sum of the ranges u_k in (10). Under this assumption, the resource provisioning problem can be solved as a linear program $provisioning$. While this is not optimal compared to faithfully modeling the C_i as the union of the ranges, this is a reasonable assumption since our goal is to obtain general guidelines for resource provisioning. As we will see in Section 6.4, this heuristic works well in practice.

There are two steps to the intelligent provisioning heuristic. The first step solves the LP $provisioning$. Next, given the resource allocation output by $provisioning$, we run the greedy algorithm in Figure 4 with $\gamma = 1$ to ensure that we are strictly within the resource constraints.

Adding a variance term to the objective: In practice, we find that it is useful to add a variance term to the objective function. We modify the above objective function $\min_i C_i$ to be $\{\min_i C_i\} - g(\{L_j^2\})$, where g is a function of the second-moments of the L values. The negative term denotes that our intent is to *minimize* the variance across the L values (with appropriate normalization to ensure that the variance term and the coverage term do not have wildly different magnitudes). Among the different configurations that maximize $\min_i C_i$, the goal is to pick the configuration that distributes the resources most uniformly across the routers. This offsets two potential undesirable effects. First, the LP solver may not necessarily use all the available resources to achieve the optimal minimum fractional coverage. Second, the LP solution may result in a skewed resource allocation which may be undesirable for the greedy algorithm and less robust to changes in traffic or routing inputs. The variance term forces the optimization solver (now a quadratic program instead of a LP) to use up the available resources efficiently and also reduces the skew. While this works well for most common cases, it may not prevent skewed allocations when $\beta >> \gamma$.

5.2 Partial OD-pair identification

Next, we consider a scenario in which a network operator can choose to upgrade some border routers. For example, this can be achieved using a software update to the router or by adding a simple two-port middlebox (using a software switch running on commodity hardware [27] or using FPGA [17]) that processes each packet, modifies the header, and forwards them to the router. These few upgraded nodes (routers or router plus middlebox) then have the capabilities to identify the OD-pairs and add the identifiers to packet headers. We assume that all routers run both cSamp and cSamp-T sampling algorithms – i.e., a router logs a flow if the hash of the flow falls in a hash-range corresponding *either* to the OD-pair or the SamplingSpec for the packet.

Let $\mathcal{P}_e$ denote the set of “enabled” OD-pairs whose packets carry OD-pair identifiers and let $\mathcal{P}$ denote the set of all OD-pairs. We compute the maximum minimum fractional coverage using a binary search over τ. The key difference between the new algorithm and Figure 4 is that each iteration of the binary search has two logical steps. In the first step, we solve a cSamp-style linear program over the enabled OD-pairs. In the second step, we define the capped functions $\hat{C}_i(\tau) = \min_i(C_i, \tau)$ for the non-enabled OD-pairs and use the greedy algorithm to maximize

$\hat{F} = \sum_i \hat{C}_i$.

Problem $enabledODs(\alpha, \mathcal{P}_e)$**:**

$$\text{Minimize } \sum_j L_j, \text{ subject to}$$

$$\forall j, \textstyle\sum_{i \in \mathcal{P}_e : R_j \in P_i} (d_{ij} \times T_i) \leq L_j \tag{13}$$

$$\forall i \in \mathcal{P}_e, C_i = \textstyle\sum_{j : R_j \in P_i} d_{ij} \tag{14}$$

$$\forall i \in \mathcal{P}_e, \ \forall j, \ d_{ij} \geq 0 \tag{15}$$

$$\forall i \in \mathcal{P}_e, \alpha \leq C_i \leq 1 \tag{16}$$

In each iteration, for the current value $\tau_{current}$, the first step involves solving the LP $enabledODs$. The input to the LP is the set of enabled OD-pairs $\mathcal{P}_e$ and the target fractional coverage $\alpha = \tau_{current}$. The objective of the LP is to minimize the total amount of resources used across the different routers to ensure that each $OD_i \in \mathcal{P}_e$ gets coverage at least $\alpha = \tau_{current}$. Solving the LP returns the resources allotted to each router or returns an infeasible status if there is no feasible solution.

If the LP is infeasible, then we directly proceed to the next iteration of the binary search. If the LP is feasible, then we obtain the new budget per router by subtracting the resources used in the LP stage from the original budget per router. Next, we run the greedy algorithm with the reduced budget and modified objective specified over the non-enabled OD-pairs. By construction, the maximum value $\hat{F}$ can take is $(M - |\mathcal{P}_e|) \times \tau_{current}$ where M is the total number of OD-pairs and $|\mathcal{P}_e|$ is the number of enabled OD-pairs. This maximum value is achieved if and only if each of the non-enabled OD-pairs (i.e., in the set $\mathcal{P} \setminus \mathcal{P}_e$) achieves a fractional coverage equal to $\tau_{current}$. If the greedy algorithm achieves this objective value, then $\tau_{current}$ is feasible and we try a higher value in the next iteration; else we try a lower value in the next iteration.

5.3 Using the α-fair objective function

The α-fairness notion has been traditionally used in the congestion control literature (e.g., [26]) to generalize the max-min notion of fair allocation. Given items x_i, and a total resource C we want to allocate the total resource to the items in a "fair" manner. The α-fairness function is defined as $\sum_i U(x_i)$, where $U(x) = \frac{x^{1-\alpha}}{1-\alpha}$. The parameter α can take values in $[0, \infty)$, and the values $\alpha = 0$, $\alpha = 1$,[4] and $\alpha \rightarrow \infty$ correspond to achieving maximum throughput, proportional fairness, and max-min fairness respectively.

In our problem setting, each x_i corresponds to the submodular function $F_i = C_i$. It is easy to verify that the function $\sum_i U(C_i)$ is submodular; thus we can use the SUBMODULARGREEDY with α set to some large value. To avoid numerical instabilities, we use $\alpha = 100$ and also add a small additive constant to each C_i at the beginning since the function $U(x)$ is undefined when $x = 0$. Note that unlike the above heuristics, using the α-fair function is tightly coupled to maximizing the minimum fractional coverage.

[4]At $\alpha = 1$, the function is defined as $U(x) = \log(x)$.

Topology (AS#)	**PoPs**	**OD-pairs**	**Flows**	**Packets**
			$\times 10^6$	$\times 10^6$
NTT (2914)	70	4900	51	204
Level3 (3356)	63	3969	46	196
Sprint (1239)	52	2704	37	148
Telstra (1221)	44	1936	32	128
Tiscali (3257)	41	1681	32	218
GÉANT	22	484	16	64
Internet2	11	121	8	32

Table 2: Parameters for the experiments

6 Evaluation

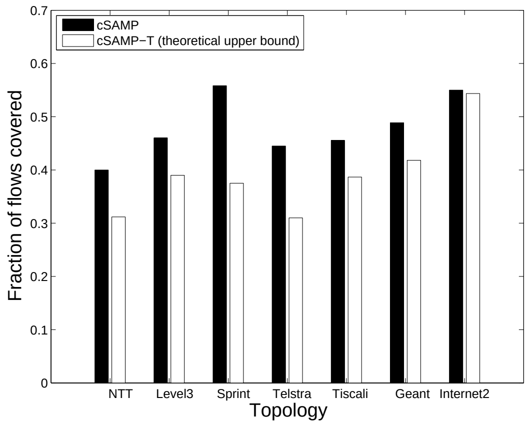
Evaluation Setup: We compare the performance of cSamp and cSamp-T at a PoP-level granularity, i.e., treating each PoP as a "router" in the network model. Our evaluation setup (Table 2) consists of several PoP-level network topologies from educational backbones and tier-1 ISP backbones inferred by Rocketfuel [33]. We use shortest-path routing to construct paths between every OD-pair. The traffic matrix is modeled using a gravity model based on city populations [31]. We assume that each PoP is provisioned to log up to $L = 400,000$ flow records.[5] For cSamp-T, we discretize the hash-range in increments of $\delta = 0.02$.

6.1 Coverage and Overlap

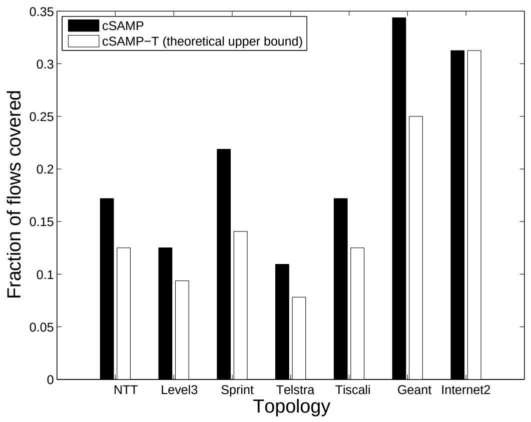
Performance gap between cSamp and cSamp-T: The approximation guarantees compare the performance of the greedy algorithms with the optimal solution for the cSamp-T problem. A related question is the gap between the optimal solutions for cSamp-T and cSamp. It is hard to reason about the optimal cSamp-T solution. Instead, we compare the theoretical upper bound for the cSamp-T problem by considering a relaxed LP-version of the problem (similar to $provisioning$ in Section 5). Figures 6(a) and 6(b) show that this performance gap for the total flow coverage and the minimum coverage respectively using a router 3-tuple granularity for cSamp-T. The figure shows that the upper bound on cSamp-T performance can be up to 30% lower than cSamp.

Total flow coverage: We are interested in two aspects: (a) the granularity of SamplingSpecs and (b) is there a significant difference in performance between the benefit or benefit-cost tradeoff versions of the greedy algorithm. Figure 7 shows that using the tuple granularity provides a significant improvement (25-30%) over the coarser router-level formulation. The figure also shows the performance of maximal flow sampling [30]. In maximal flow sampling, the flow sampling rate for a router is $\min(1, \frac{l}{t})$, where l is the number of flow records it is provisioned to hold and t is the total number of flows it observes; each node maximally utilizes the available resources. cSamp-T with the tuple formulation is closest to cSamp. Comparing this with Figure 6(a), we also see that the

[5]Assuming 12 bytes per flow record [30], this translates into $400,000 \times 12 = 4.8$ MB of SRAM per PoP, which is well within the 8 MB technology limit per linecard suggested by Varghese [36].



(a) Total coverage



(b) Minimum coverage

Figure 6: Performance gap between cSamp and theoretical upper-bound for cSamp-T

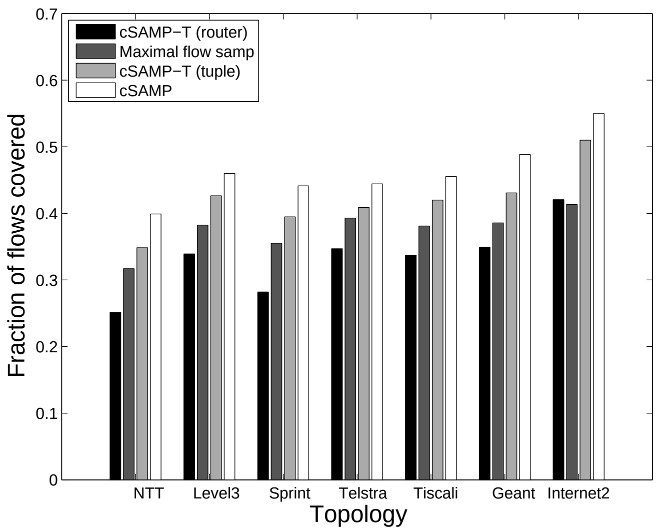


Figure 7: Total flow coverage

greedy algorithm is very close to the theoretical upper bound for cSamp-T.

The theoretical guarantee for total flow coverage depends on running the two greedy algorithms: with and without the cost-benefit flag. We want to understand if there is a clear difference in performance between the two configurations. Figure 8 shows that both configurations have very similar performance and that the algorithm with the cost-benefit flag $cbflag = \textbf{false}$ is slightly better.

Minimum fractional coverage: We saw in Section 4 that it is impossible to maximize f_{min} using a greedy algorithm without resource augmentation. Thus, we evaluate the performance as a function of the resource augmentation factor γ where each router's budget is $\gamma \times 400,000$. As in the previous scenario, we consider both router and tuple granularities. In Figure 9, we normalize the minimum fractional coverage by the optimal value achieved by cSamp at the baseline provisioning (i.e., cSamp at $\gamma = 1$). For example, if the greedy algorithm returned a value of 0.2 at $\gamma = 3$ and the

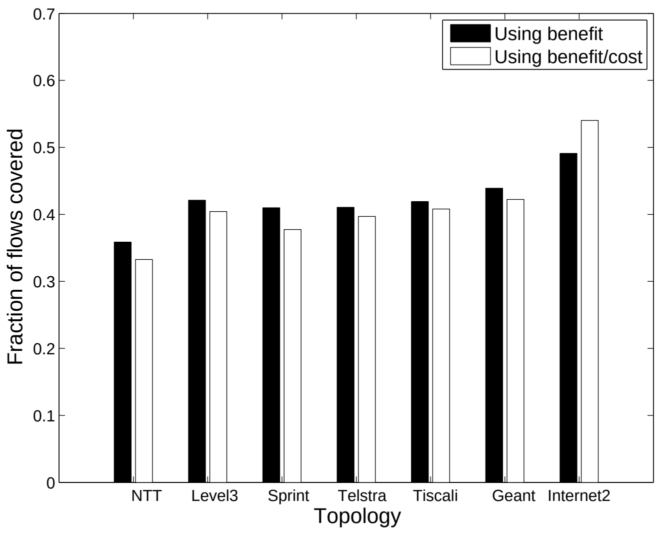


Figure 8: Benefit vs. benefit-cost versions

solution for cSamp has value 0.4 at $\gamma = 1$, the normalized y-axis value corresponding to $\gamma = 3$ is $\frac{0.2}{0.4} = 0.5$.

First, with $\gamma \geq 4$, cSamp-T has performance comparable ($\geq$ 50%) to cSamp for all topologies. Second, the difference between the router and tuple formulations becomes even more pronounced with the minimum fractional coverage result – there is a significant advantage to be gained in using more fine-grained SamplingSpecs. With router-level SamplingSpecs, even at $\gamma = 5$, four out of the seven topologies only reach 40% of cSamp's performance. For the same $\gamma = 5$, with tuple-level SamplingSpecs, five out seven topologies achieve at least 90% of cSamp's performance.

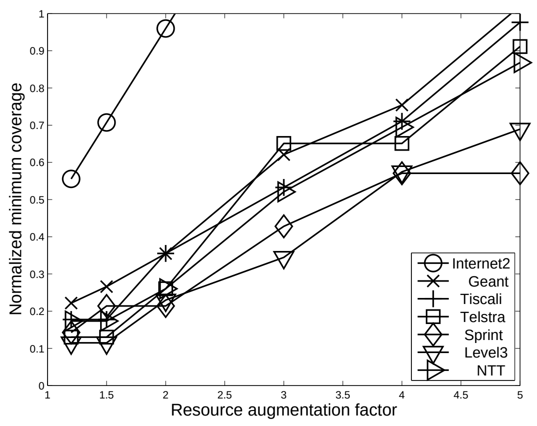
Figure 10 shows the corresponding result when we use the α-fairness objective function with the tuple formulation. We see that this function gives slightly better performance compared to the capped-minfrac technique used above.

The γ at which cSamp has good performance is much better than the theoretical bound in Section 4. In Section 6.4, we show that targeted provisioning reduces this even further.

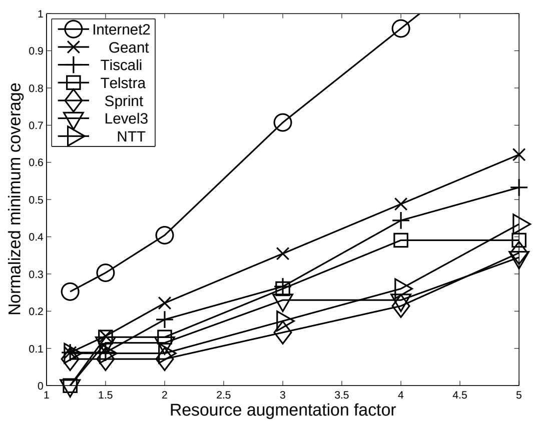
Duplicated flow reports: A secondary objective of cSamp is to minimize the total amount of duplicated flow reports. This reduces the data management overhead in processing and eliminating duplicated flow measurements. Figure 11 shows the ratio of duplicated flow reports to the number of unique flow reports comparing cSamp-T (at the tuple granularity) and maximal flow sampling. Compared to maximal flow sampling, cSamp-T has 2-3$\times$ fewer duplicated flow reports. Compared to cSamp (zero duplicated reports) this is not ideal; however, this performance penalty is unavoidable since cSamp-T operates at a much coarser granularity.

6.2 Algorithm running time

In order for cSamp-T to be reasonably responsive to network dynamics, we want the time to compute sampling manifests to be within few tens of seconds. (A typical measurement epoch spans a few minutes; we expect that manifests are recomputed across epochs, not within epochs.) Ta-



(a) Tuple



(b) Router

Figure 9: Normalized minimum fractional coverage achieved by cSamp-T as a function of the resource augmentation factor

Topology	Total coverage (sec)		Min. Fractional (sec)	
	Naive	**Lazy**	**Naive**	**Lazy**
NTT	207.12	4.15	39632	154.1
Level3	205.36	3.30	48269	84.3
Sprint	75.30	2.21	14211	71.6
Telstra	50.53	1.65	6997	45.0
Tiscali	35.18	1.16	8518	33.7
GÉANT	3.06	0.28	542	7.6
Internet2	0.22	0.05	38.4	1.9

Table 3: Time to compute sampling strategy comparing the vanilla greedy algorithm with the lazy evaluation optimization

ble 3 shows the computation times using the “vanilla” greedy and lazy evaluation algorithms. Lazy evaluation provides more than an order of magnitude reduction in the total computation time. The reduction is even more significant for the minimum fractional coverage since it involves multiple invocations of the greedy subroutine during the binary search. With this reduction, cSamp-T scales to larger topologies.

6.3 Size of sampling manifests

Compared to cSamp, cSamp-T increases the size of the sampling manifests. This is because, unlike cSamp, the hash-ranges assigned for each SamplingSpec are no longer contiguous blocks. As discussed earlier in Section 4.3, to reduce the size of the manifests, we implement a simple compression heuristic to merge hash-ranges after the greedy algorithm computes the manifests. This looks for maximally contiguous hash ranges in the original sampling manifest and merges them into a single hash range.

We evaluate the overhead of disseminating manifests in Table 4. First, the merge algorithm

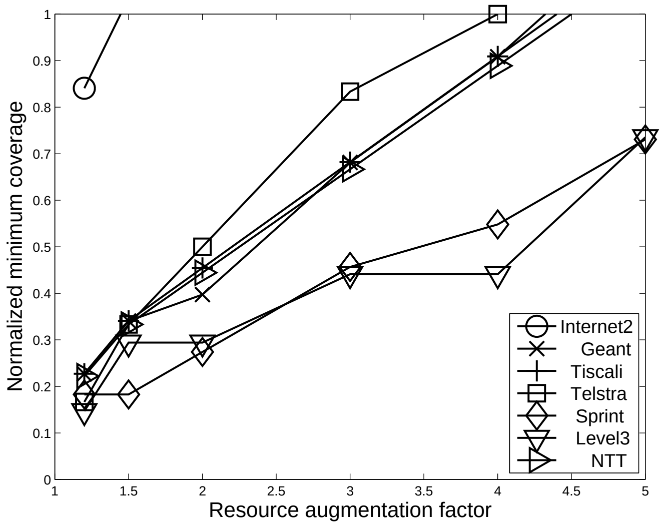


Figure 10: Normalized minimum fractional coverage using the α-fair function with the tuple formulation

Topology	Total (KB)		Max. per PoP (KB)	
	Naive	**Merged**	**Naive**	**Merged**
NTT	178.5	16.3	5.6	1.0
Level3	341.9	25.2	34.1	3.3
Sprint	140.9	13.0	10.3	0.6
Telstra	112.3	7.2	3.3	0.5
Tiscali	110.9	12.6	9.8	0.6
GÉANT	45.5	6.5	5.6	0.6
Internet2	14.5	5.0	4.5	0.7

Table 4: Size of the sampling manifests (in kilobytes of text configuration files) with cSamp-T

reduces the manifest sizes roughly 10$\times$. Second, we notice that the total bandwidth overhead of disseminating the manifests is not large – 25KB in the worst case after the merge routine. Finally, on a per-router basis, the worst case size of the manifest is around 3KB which is quite low.

6.4 Intelligent Resource Provisioning

As a specific scenario, we set $LB_j = L = 400,000$ for all j. We model the total budget as $Budget = \gamma \times N \times L$ (N is the number of PoPs) and the technology limit as $\beta \times L$. We vary γ and β and for each pair of values. Figures 12(a) and 12(b) show the result for two of the topologies, Level3 (AS3356) and Telstra (AS1221) respectively. We chose these topologies because the greedy algorithm performed poorly with respect to cSamp in Figure 9. An interesting result is that the curve levels off as a function of γ; i.e., there is not much to be gained with increasing the total budget. However, there is significant improvement by increasing β, the technology upper bound. In fact, even with a moderate increase $\gamma = 1.2$, we see that the performance gets within 80% of

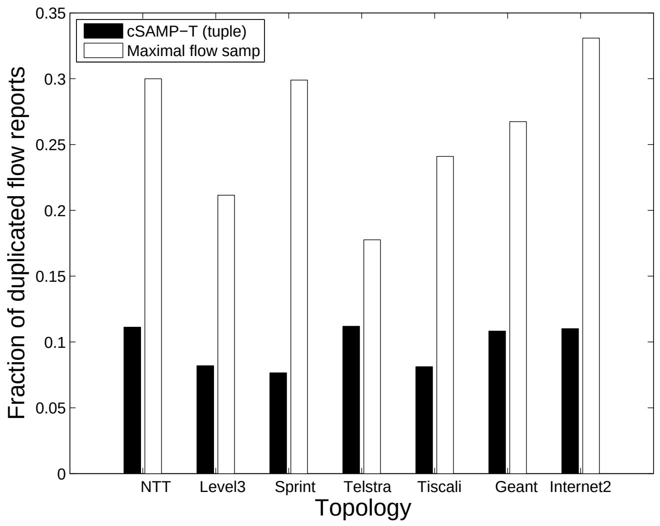


Figure 11: Ratio of duplicated flow reports to the number of unique flow reports

the cSamp performance.

Since β is more crucial to the overall performance than γ, for the remaining topologies we fix $\gamma = 1.5$ and analyze the normalized minimum fractional coverage as a function of β in Figures 13 and 14. With $\beta = 5$, all the topologies achieve at least 60% of the ideal cSamp performance. Similar to the previous results, the α-fair shows slightly better performance. Contrasting this result with Figures 9 and 10, the main difference is that we do not require all PoPs to be augmented with five times as many resources – the total resource budget is less than $1.5\times$.

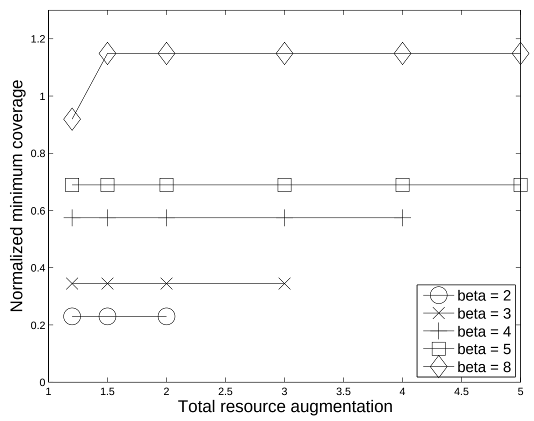
6.5 Partial OD-pair identification

We try three strategies for selecting the enabled OD-pairs $\mathcal{P}_e$: upgrading the top-k PoPs that (a) observe the maximum amount of traffic, (b) lie on most number of routing paths, or (c) originate the most traffic. Here, upgrading implies that we enable OD-pair identifiers on all OD-pairs having one of these top-k PoPs as origins. For each k, we run the two-step procedure from Section 5.2 for all values in $1, \ldots, k$ and pick the configuration with the highest f_{min}.

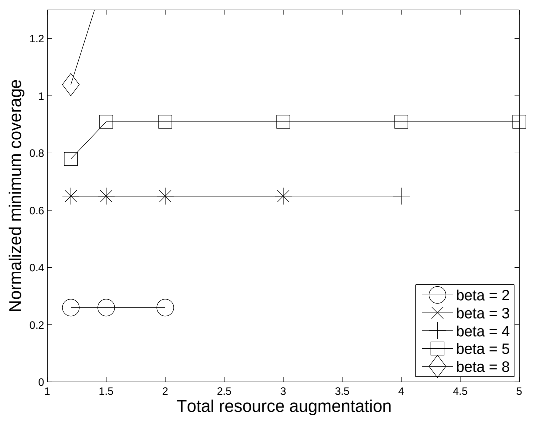
Figures 15(a) and 15(b) show the normalized minimum fractional coverage for the Level3 and Telstra topologies as a function of k (number of top-k PoPs). First, we observe that enabling even on a small number (around 8%) significantly improves the performance. Second, enabling identifiers on routers that observe the most traffic performs much better than the other two strategies.

6.6 Hybrid Coverage objective

cSamp maximizes the total flow coverage subject to achieving the highest possible minimum fractional coverage across OD pairs. So far, in cSamp-T we considered these two objectives separately. A natural question is if there is an effective algorithm for maximizing the hybrid objective, i.e., maximize total coverage subject to achieving the maximum minimum fractional coverage. It is relatively simple to extend the algorithm in Figure 4 to achieve this – first run the greedy algorithm



(a) Level3



(b) Telstra

Figure 12: Understanding the impact of total resource augmentation (γ) and technology upper bound (β) in the resource allocation formulation.

AS	**Greedy-Minfrac**		**Greedy-Total**
	NoHybrid	**Hybrid**	
NTT	0.13	0.58	0.58
Level3	0.10	0.60	0.60
Sprint	0.22	0.61	0.64
Telstra	0.13	0.59	0.62
Tiscali	0.23	0.60	0.63
GÉANT	0.35	0.63	0.68
Internet2	0.60	0.71	0.78

Table 5: Comparing the performance of the hybrid maximization to the greedy algorithm for maximizing the total flow coverage alone

to optimize the capped minimum fractional objective ($\hat{F}$) and then modify the objective function to optimize the total coverage if $\tau_{current}$ is feasible.

To evaluate this hybrid approach, we consider the resource configuration obtained using the targeted provisioning approach with $\alpha = 1.5$ and $\beta = 5$. Table 5 compares the total coverage obtained with three strategies: maximizing the minimum fractional coverage, maximizing the total flow coverage, and the above two-step heuristic. Not surprisingly, we find that maximizing the minimum fractional coverage alone does not work well for the total coverage. This is because the greedy algorithm terminates when it has achieved the targeted coverage for all OD pairs even if it has additional resources that can be used to boost the total coverage. The table also shows that total coverage obtained by the hybrid approach is very close to that of the greedy algorithm for maximizing the total coverage alone. While it is hard to provide theoretical guarantees for the hybrid objective, Table 5 shows that the our approach works very well in practice.

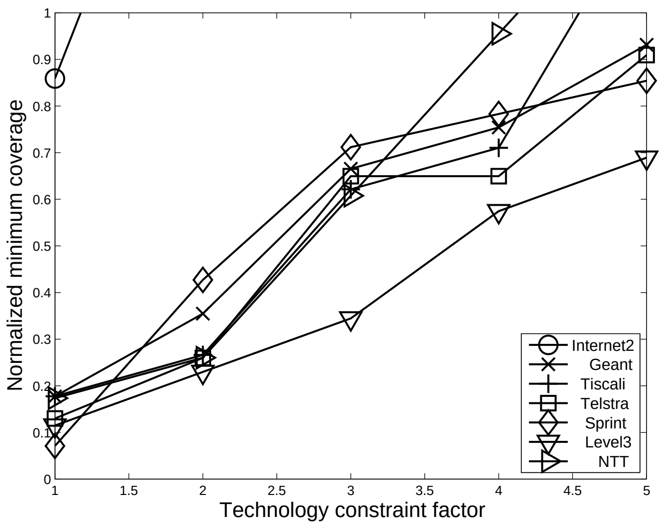


Figure 13: Intelligent resource allocation with $\gamma = 1.5$ and varying β

7 Discussion

More fine-grained local information: Our current choice of SamplingSpecs is topology-driven; we model the granularity of sampling manifests in terms of path-segments (e.g., router or router 3-tuple). One direction of future work is to expand the scope to include prefix and routing table information. For example, it might be possible to approximately estimate the OD-pair information given the source and destination address of a packet and the available routing table information or alternatively providing additional information (e.g., distributing IP-prefix to ingress-egress maps to routers [1]). This creates the possibility of a cSamp-T formulation with more fine-grained information to bring the performance closer to cSamp.

Sensitivity of router upgrades: Section 5 suggests two heuristics for upgrading routers either with additional memory or the ability to insert OD-pair identifiers in packet headers. The provisioning and partial marking formulations, as presented, assume static routing and a static traffic matrix. Real-world routing and traffic matrices typically have some dominant structural patterns that are invariant to localized dynamics. Thus, we can apply these formulations and perform upgrades after extracting these dominant patterns. Evaluating the sensitivity of the performance improvements to traffic or routing dynamics and designing upgrade strategies robust to dynamics are topics of future work.

8 Related Work

Theory of submodularity: Submodular set-functions have long been studied as discrete analogs of convex functions: in particular, maximizing a submodular function subject to side constraints has a rich history; see, e.g., [3, 37, 35] and the references therein.

Sampling solutions: Most of the related work focuses on the single-router case and on providing incremental solutions to work around the limitations of uniform packet sampling. This includes

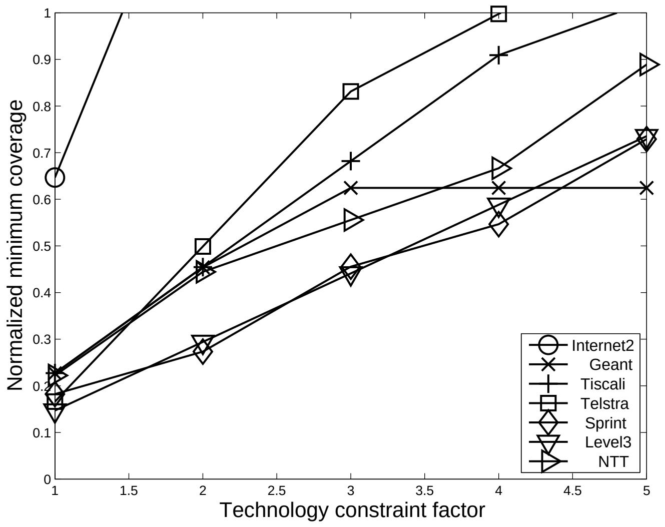
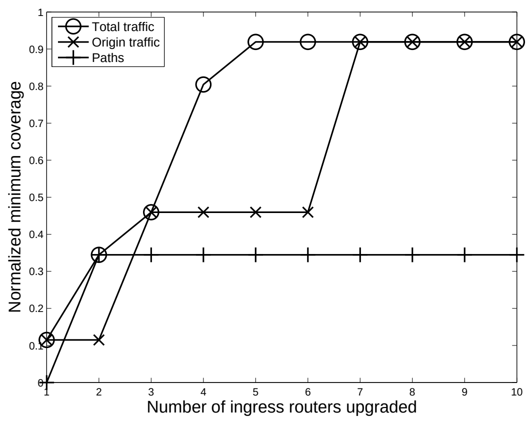
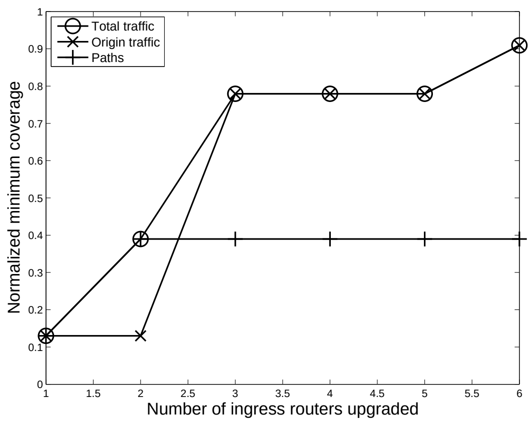


Figure 14: Intelligent resource allocation with $\gamma = 1.5$ with the α-fair function



(a) Level3



(b) Telstra

Figure 15: Performance of cSamp-T with partial OD-pair identification. Alternatively, this can be viewed as incremental deployment of cSamp via cSamp-T.

work on adaptive sampling [9, 18], heavy-hitter detection [10], inverting sampled measurements [8, 16], and data streaming algorithms [22, 24]. The closest related work is cSamp [30], which we discussed in Section 2.

Greedy algorithms for monitor placement: Prior work has applied greedy algorithms for monitor placement to cover all routing paths using as few monitors as possible [5, 34]. The authors show that such a formulation is NP-hard and propose greedy approximation algorithms. There are also extensions to these problems to incorporate packet sampling [34, 4]. However, these do not satisfy flow coverage objectives, and in fact by relying on packet sampling, they can result in a large amount of redundant flow measurements. cSamp-T provides more fine-grained flow coverage objectives and reduces duplicated flow reports.

Sensor network monitoring: There has been recent work applying the theory of maximizing submodular set cover functions in the context of maximizing information obtained from multiple sensors [15, 20]. The objective of selecting observations against a set of adversarial objectives [21] is similar to the notion of maximizing the minimum fractional coverage objective. Krause and Guestrin [19] provide a good survey of known results and applications of these ideas.

References

[1] K. Argyraki, P. Maniatis, O. Irzak, S. Ashish, and S. Shenker. Loss and Delay Accountability for the Internet. In *Proc. of ICNP*, 2007.

[2] H. Ballani and P. Francis. CONMan: A Step Towards Network Manageability. In *Proc. of ACM SIGCOMM*, 2007.

[3] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a submodular set function subject to a matroid constraint (extended abstract). In *12th Integer Programming and Combinatorial Optimization Conference*, pages 182–196, 2007.

[4] G. R. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran. Reformulating the Monitor Placement problem: Optimal Network-Wide Sampling. In *Proc. of CoNeXT*, 2006.

[5] C. Chadet, E. Fleury, I. Lassous, H. Rivano, and M.-E. Voge. Optimal Positioning of Active and Passive Monitoring Devices . In *Proc. of CoNeXT*, 2005.

[6] M. P. Collins and M. K. Reiter. Finding Peer-to-Peer File-sharing using Coarse Network Behaviors. In *Proc. of ESORICS*, 2006.

[7] N. Duffield and M. Grossglauser. Trajectory Sampling for Direct Traffic Observation. In *Proc. of ACM SIGCOMM*, 2001.

[8] N. Duffield, C. Lund, and M. Thorup. Charging from sampled network usage. In *Proc. of IMW*, 2001.

[9] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a Better NetFlow. In *Proc. of ACM SIGCOMM*, 2004.

[10] C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting. In *Proc. of ACM SIGCOMM*, 2002.

[11] A. Feldmann, A. G. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving Traffic Demands for Operational IP Networks: Methodology and Experience. In *Proc. of ACM SIGCOMM*, 2000.

[12] G. Nemhauser, L. Wosley, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.

[13] G. Robertazzi and S. C. Schwartz. An accelerated sequential algorithm for producing D-optimal designs. *SIAM Journal of Scientific and Statistical Computing*, 10(2):341–358, Mar. 1989.

[14] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Meyers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A Clean Slate 4D Approach to Network Control and Management. *ACM SIGCOMM CCR*, 35(5), Oct. 2005.

[15] C. Guestrin, A. Krause, and A. Singh. Near-optimal Sensor Placements in Gaussian Processes. In *Proc. of ICML*, 2005.

[16] N. Hohn and D. Veitch. Inverting Sampled Traffic. In *Proc. of IMC*, 2003.

[17] J. W. Lockwood et al. NetFPGA - An Open Platform for Gigabit-rate Network Switching and Routing . In *Proc. IEEE MSE*, 2007.

[18] R. Kompella and C. Estan. The Power of Slicing in Internet Flow Measurement. In *Proc. of IMC*, 2005.

[19] A. Krause and C. Guestrin. Near-optimal Observation Selection Using Submodular Functions. In *Proc. of AAAI*, 2007.

[20] A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg. Near-optimal Sensor Placements: Maximizing Information while Minimizing Communication Cost . In *Proc. of IPSN*, 2006.

[21] A. Krause, B. McMahan, C. Guestrin, and A. Gupta. Selecting Observations Against Adversarial Objectives. In *Proc. of NIPS*, 2007.

[22] A. Kumar, M. Sung, J. Xu, and J. Wang. Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Distribution. In *Proc. of ACM SIGMETRICS*, 2004.

[23] A. Lakhina, M. Crovella, and C. Diot. Diagnosing Network-Wide Traffic Anomalies. In *Proc. of ACM SIGCOMM*, 2004.

[24] A. Lall, V. Sekar, J. Xu, M. Ogihara, and H. Zhang. Data Streaming Algorithms for Estimating Entropy of Network Traffic. In *Proc. of ACM SIGMETRICS*, 2006.

[25] M. Minoux. Accelerated Greedy Algorithms for Maximizing Submodular Set Functions. In *Proc. of 8th IFIP Conference, Springer-Verlag*, 1977.

[26] J. Mo and J. Walrand. Fair end-to-end window-based congestion control. *IEEE Transactions on Networking*, 8(5), Oct 2000.

[27] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The click modular router. *SIGOPS Operating Systems Review*, 33(5):217–231, 1999.

[28] A. Ramachandran, S. Seetharaman, and N. Feamster. Fast Monitoring of Traffic Subpopulations . In *Proc. of IMC*, 2008.

[29] M. Ramakrishna, E. Fu, and E. Bahcekapili. Efficient Hardware Hashing Functions for High Performance Computers. *IEEE Transactions on Computers*, 46(12):1378–1381, 1997.

[30] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. Kompella, and D. G. Andersen. cSamp: A System for Network-Wide Flow Monitoring. In *Proc. of NSDI*, 2008.

[31] M. R. Sharma and J. W. Byers. Scalable Coordination Techniques for Distributed Network Monitoring. In *Proc. of PAM*, 2005.

[32] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-Based IP Traceback . In *Proc. of ACM SIGCOMM*, 2001.

[33] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proc. of ACM SIGCOMM*, 2002.

[34] K. Suh, Y. Guo, J. Kurose, and D. Towsley. Locating Network Monitors: Complexity, heuristics and coverage. In *Proc. of IEEE INFOCOM*, 2005.

[35] M. Sviridenko. A note on maximizing a submodular set function subject to knapsack constraint. *Operations Research Letters*, pages 41–43, 2004.

[36] G. Varghese. *Network Algorithmics*. Morgan Kaufman, 2005.

[37] L. A. Wolsey. Maximising real-valued submodular functions: Primal and dual heuristics for location problems. *Mathematics of Operations Research*, 7(3):410–425, 1982.

[38] Y. Xie, V. Sekar, D. A. Maltz, M. K. Reiter, and H. Zhang. Worm Origin Identification Using Random Moonwalks. In *Proc. of IEEE Symposium on Security and Privacy*, 2005.

[39] K. Xu, Z.-L. Zhang, and S. Bhattacharya. Profiling Internet Backbone Traffic: Behavior Models and Applications. In *Proc. of ACM SIGCOMM*, 2005.

[40] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast Accurate Computation of Large-scale IP Traffic Matrices from Link Loads. In *Proc. of ACM SIGMETRICS*, 2003.

[41] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund. Online Detection of Hierarchical Heavy-hitters. In *Proc. of IMC*, 2004.

A NP-hardness

First, we show that the decision version of the f_{tot} cSamp-T problem with $\delta = 1$ is NP-hard via a reduction from 3-SAT. Then, we extend the result and show the $\delta < 1$ case is at least as hard as the $\delta = 1$ case.

Hardness for $\delta = 1$: Let the variables in the 3-SAT problem be denoted by $x_1, \ldots, x_N$ and the clauses denoted by $C_1, \ldots, C_M$. Given an instance of a 3-SAT problem, we construct a cSamp-T problem as follows.

The set of "routers" in cSamp-T is $X \cup T \cup F \cup D$, where $X = \{X_1, \ldots, X_N\}$, $T = \{T_1, \ldots, T_N\}$, $F = \{F_1, \ldots, F_N\}$, and $D = \{D_1, \ldots, D_N\}$. Edges in the graph are $\{\langle T_j, X_j \rangle\} \cup \{\langle F_j, X_j \rangle\} \cup \{\langle X_j, D_j \rangle\} \cup \{\langle D_j, T_{j'} \rangle | j' > j\} \cup \{\langle D_j, F_{j'} \rangle | j' > j\}$.

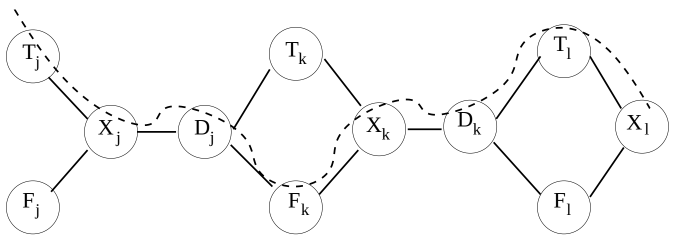


Figure 16: Example showing the path corresponding to the clause $C_i = (x_j \vee \overline{x_k} \vee x_l)$

Each SamplingSpec a_k can be one of the following: $\langle T_j, X_j, D_j \rangle$, $\langle F_j, X_j, D_j \rangle$, $\langle X_j, D_j, T_{j'} \rangle$, and $\langle X_j, D_j, F_{j'} \rangle$. There is exactly one SamplingAtom g_{k1} for each a_k and is equal to $\langle a_k, [0,1] \rangle$. The budget constraints for D, F, and T nodes is zero. The only non-zero budgets are on the X nodes and $Budget(X_j)$ is equal to $\max($*#clauses with* x_j, *#clauses with* $\overline{x_j})$.

For each clause, we construct a OD-pair/path P_i as follows. Without loss of generality, let us assume that the clauses appear in sorted order of the variable indices. If the literal x_j appears in the clause, there is a sequence of vertices of the form T_j, X_j, D_j in the path. If the literal $\overline{x_j}$ appears in the clause, there is a sequence of vertices of the form F_j, X_j, D_j in the path. P_i has edges from D_j to the adjacent (in sorted order of indices) variable's $T_{j'}$ or $F_{j'}$ depending on whether $x_{j'}$ appears in positive or negative form in the clause. Each path has unit traffic, i.e. $\forall i,\ T_i = 1$.

Example: If $C_i = (x_j \vee \overline{x_k} \vee x_l)$, we create a path $P_i = (T_j, X_j, D_j, F_k, D_k, T_l, X_l)$ as shown in Fig. 16.

Claim: The decision problem of checking if $f_{tot} = M$ on the above cSamp-T problem is equivalent to solving the 3-SAT instance.

By construction, the only non-trivial SamplingAtoms are of the form $\langle \langle T_j, X_j, D_j \rangle, [0,1] \rangle$ or $\langle \langle F_j, X_j, D_j \rangle, [0,1] \rangle$. Note that they specify all-or-nothing responsibilities. Due to the way the budgets are defined, for each X_j exactly one of $\langle T_j, X_j, D_j, [0,1] \rangle$ or $\langle F_j, X_j, D_j, [0,1] \rangle$ is "active"—in effect this corresponds to setting the variable x_j to be true or false. Hence, P_i has unit coverage in the solution of the cSamp-T instance if and only if there is at least one satisfied literal in clause C_i. Thus, checking if there is a satisfying assignment or not for the 3-SAT formula is equivalent to checking if the coverage $f_{tot} = M$ or $f_{tot} < M$. (In fact, it is also equivalent to checking if $f_{min} = 1$ or $f_{min} = 0$.) This proves the hardness for both cSamp-T problems of maximizing f_{tot} and f_{min} with $\delta = 1$.

Hardness with finer discretization: Given integer $d \geq 1$, the hardness for the $\delta = 1/d < 1$ case follows from a reduction from the $\delta = 1$ problem. Indeed, given an instance of the cSamp-T decision problem of deciding if $f_{tot} = M$ with $\delta = 1$, we construct the following instance with $\delta = 1/d$: we create $d - 1$ "dummy" vertices $V_1, \ldots, V_{d-1}$, and prepend these vertices to all paths P_i. We set the budgets on the dummy vertices to be $(1/d) \times M$. For every non-dummy vertex in the $\delta = 1$ problem, we scale the budgets by a factor $1/d$. By construction, $f_{tot} = M$ on the $\delta = 1/d$ problem if and only if $f_{tot} = M$ on the $\delta = 1$ problem; an analogous result holds for f_{min}. Thus, the $\delta = 1/d$ problems are at least as hard as the $\delta = 1$ problems.

B Algorithmic Guarantees

Suppose we are given a monotone submodular function $F : U \to \Re$ with a partition $U = U_1 \uplus U_2 \uplus \ldots \uplus U_k$. The goal is to pick a set $S \subseteq U$ such that $|S \cap U_i| \leq 1$ and the value $F(S)$ is maximized. (In other words, we have a partition matroid on U and want to maximize F subject to S being independent in this matroid.) If we greedily pick elements $e_i \in U_i$ such that e_i is an element that α-approximately maximizes ($\alpha \leq 1$) the marginal benefit $F(\{e_1, e_2, \ldots, e_{i-1}, e_i\}) - F(\{e_1, e_2, \ldots, e_{i-1}\})$, then the benefit $F(\{e_1, \ldots, e_k\})$ is at least $\frac{\alpha}{2+\alpha}$ of the optimal benefit possible [3].

A different setting is when $F : U \to \Re$ is monotone submodular, we have a "budget" B, and each $e \in U$ has "size" c_e: the goal is to pick $S \subseteq U$ with $c(S) := \sum_{e \in S} c_e \leq B$. Consider two greedy algorithms: (a) the "cost/benefit" algorithm greedily keeps picking an element e which maximizes $\frac{\text{increase in } F}{c_e}$ and does not violate the budget, and (b) the "benefit" algorithm greedily keeps picking element e which maximizes the increase in F and does not violate the budget. One can show that the better of these two algorithms gets benefit at least 0.35 times the best possible [37]. In fact, an algorithm based on partial enumeration [35] gets an optimal $(1 - e^{-1})$-approximation.

We can combine these ideas to solve the problem of "submodular maximization subject to partition-knapsack constraints". Formally, we are given a monotone submodular function $F : \mathcal{V} \to \Re$, where there is a partition $\mathcal{V} = \mathcal{V}_1 \uplus \mathcal{V}_2 \uplus \ldots \uplus \mathcal{V}_k$. Each element $e \in \mathcal{V}$ has a size c_e, and each part $\mathcal{V}_i$ has a budget B_i: we want to pick a set $S \subseteq \mathcal{V}$ such that if $S_i = S \cap \mathcal{V}_i$, then the knapsack constraint $\sum_{e \in S_i} c_e \leq B_i$ is satisfied. For this problem, we can combine the two ideas above: imagine each valid knapsack of the elements in $\mathcal{V}_i$ to be a distinct element of the abstract set U_i, and $U = \uplus U_i$. Then considering the parts $\mathcal{V}_i$ one-by-one, and running the better of the benefit or cost-benefit algorithms on each part, results in the following result:

Theorem B.1. *The simple greedy algorithm described above is a* $\frac{0.35}{2+0.35} \geq 0.148$*-approximation for the problem of submodular maximization subject to partition-knapsack constraints. Using a knapsack algorithm based on partial enumeration, we can get a* $\frac{e-1}{3e-1} \approx 0.406$*-approximation.*

As always, note that the results are *worst-case guarantees*: often these greedy algorithms for submodular maximization perform much better in practice.

The idea can be extended to the max-min problem. The algorithm for the max-min problem (subject to a cardinality constraint) from Krause et al. [21] uses an $(1 - e^{-1}) \approx 0.632$-approximation algorithm for submodular maximization only in a black-box fashion. Hence we can replace that algorithm by the above algorithm for submodular maximization subject to partition-knapsack constraints to get a bicriteria algorithm for the max-min problem that achieves optimal benefit, but exceeds each budget by a factor $O\big(\log(\sum_{e \in \mathcal{V}} F_i(v))\big)$—the fact that we are using an approximation guarantee of 0.148 instead of 0.632 only changes the constants in the big-oh.