Coordinated Sampling sans Origin-Destination Identifiers: Algorithms, Analysis, and Evaluation PaperId: 19

Vyas Sekar*, Anupam Gupta*, Michael K. Reiter†, Hui Zhang*
*Carnegie Mellon University, †UNC-Chapel Hill

Abstract

Flow monitoring is used for a wide range of network management applications. Many such applications require that the flow monitoring infrastructure 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 sees. We present a new framework called cSamp-T, in which each router works with *only local information*. Using results from the theory of maximizing submodular set functions, we show that cSamp-T provides near-ideal performance in maximizing the total flow coverage in the network. Further, with a small amount of targeted provisioning or upgrades to a small number of ingress routers, cSamp-T nearly optimally maximizes the minimum fractional coverage across all OD-pairs. We demonstrate these results on a range of real topologies. cSamp-T thus makes the benefits of coordinated network-wide monitoring immediately available to ISPs and also provides an incremental deployment path for such solutions.

1. INTRODUCTION

Flow monitoring supports critical network management tasks such as anomaly detection [23], identifying unwanted application traffic [9], understanding traffic structure [36], botnet analysis [26], and forensic analysis [37], in addition to traditional traffic engineering and accounting [11]. These applications require that monitoring infrastructures provide high flow coverage (number of flows logged) and the ability to achieve network-wide flow measurement goals.

To meet these demands, recent proposals articulate the case for network-wide rather than router-centric approaches for flow monitoring [7, 15, 33]. We use one such proposal, Coordinated Sampling (cSamp) [33], 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.

The key to these benefits is that cSamp coordinates the sampling actions of routers in the network. To achieve this coordination, cSamp assumes that each router can immediately determine the Origin-Destination (OD) pair (i.e., the ingress and egress router) for each packet it sees. However, due to practical issues such as multi-exit peers and prefix-aggregation, interior routers in ISPs cannot identify the OD-pair given just the source and destination IP addresses. Thus, cSamp requires: (i) upgrades to border routers to compute the OD-pair identifiers and (ii) modifications to packet headers to carry OD-pair identifiers. These present significant deployment barriers for ISPs today. Thus, while cSamp has the potential to substantially improve flow monitoring, its current form lacks a practical deployment path.

In this paper, we address the challenge of providing flow monitoring capabilities comparable to cSamp, without relying on OD-pair identifiers. We present an architecture called cSamp-T[1], in which each router makes sampling decisions using only local information from (unmodified) packet headers and routing tables, but still provides performance comparable to cSamp.

An immediate concern in cSamp-T is that the efficient algorithms used in cSamp to compute sampling strategies for maximizing the total flow coverage and minimum fractional coverage across all OD-pairs no longer apply. In fact, we show that these problems are NP-hard in the cSamp-T case. Consequently, a central challenge we address in this paper is to develop efficient algorithms for computing sampling strategies to (approximately) optimize these measures.

To maximize the total number of unique flows logged, we use the insight that the objective function is *submodular* and obtain near-optimal performance by extending results from the theory of optimizing submodular functions subject to budget constraints [14, 24, 21]. In particular, submodularity implies that we can implement efficient greedy algorithms with good approximation guarantees.

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, and the greedy algorithm performs poorly in theory [21] and practice. In this case, we present two practical strategies to improve the performance: (a) augmenting targeted routers with more resources and (b) incrementally upgrading border routers with the functionality to compute OD-pair identifiers and add them to packet

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

headers. We show that a few such router upgrades significantly boosts the minimum fractional coverage.

cSamp-T thus makes the benefits of network-wide monitoring solutions such as cSamp immediately available to ISPs by relaxing the dependence on OD-pair identifiers. It also provides an incremental deployment path for ISPs to transition to cSamp-like solutions while providing performance comparable to the ideal case under partial deployment. We also believe that the algorithms and heuristics we develop (e.g., applying results from the theory of submodular set maximization, intelligent resource provisioning, hybrid cSamp and cSamp-T deployment) can be more broadly applied to other network management problems (e.g., [4]).

2. BACKGROUND AND MOTIVATION

In this section, we provide a brief overview of cSamp and also explain a key challenge that makes it impractical for ISPs to deploy cSamp-like solutions today.

2.1 Why cSamp?

Flow monitoring is crucial for several network management functions including several anomaly detection and security applications (e.g., [23, 9, 39, 36, 37, 26]), and this set of applications continues to grow. Synthesizing arguments from several previous papers [17, 22, 7, 31, 15, 29, 10], we identify some key requirements to address these demands:

- Provide high flow coverage, i.e., log as many flows as possible, to support security applications which need a fine-grained understanding of "who-talked-to-whom".
- Work efficiently within (possibly heterogeneous) router resource constraints and minimize redundant reports.
- Satisfy network-wide flow monitoring objectives; e.g., specify some subsets of traffic as more important than others or ensure fairness across different subsets.
- Support a broad spectrum of monitoring applications.

Extending observations from the above body of work, these lead to three natural design choices in cSamp [33]: (1) using flow sampling instead of packet sampling to avoid the bias of packet sampling against small flows [17]; (2) coordinating routers to avoid redundant sampling and to use router resources efficiently [29]; and (3) a network-wide optimization framework for assigning monitoring responsibilities to routers to meet an ISP's objectives [15].

2.2 Overview of cSamp

The goal of cSamp is to assign sampling responsibilities to routers in a coordinated manner to optimize network-wide flow monitoring objectives. Network operators typically specify network-wide goals in terms of *Origin-Destination (OD) pairs*, identified by the ingress and egress routers. Thus, the network-wide flow monitoring objective is expressed as a function of the fractional flow coverages (i.e., fraction of flows logged) of each OD-pair.

cSamp assigns sampling responsibilities as *hash-ranges per OD-pair per router*. These configurations are called *sampling manifests*; the manifest for each router is a set of entries of the form $\langle OD, [start, end]\rangle$, where $[start, end] \subseteq [0, 1]$ denotes a hash range. The key idea is that all routers are bootstrapped with the same hash function but are assigned *non-overlapping* hash ranges per OD-pair. This coordinates the routers to ensure efficient non-redundant monitoring. Since the sets of flows sampled by different routers do not overlap, the flow coverage for an OD-pair is simply the sum of the per OD-pair per-router flow coverages across routers on that OD-pair's path(s).

Each router's sampling algorithm is as follows. For each packet, the router determines the OD-pair from the packet header. Next, it computes the HASH of the flow 5-tuple *srcIP, dstIP, srcport, dstport, proto*, which returns a value in $[0, 1]$, and checks if the hash value lies in the range assigned to it for the OD-pair. Each router maintains a $Flowtable$ of the flows it logs. If the packet is selected, the router either creates a new flow entry or updates counters for the corresponding flow in the $Flowtable$.

Next, we discuss how the sampling manifests are generated by the network-wide optimization framework.

Optimization Framework: The inputs to the optimization module 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 expressed in terms of the fractional flow coverages per OD-pair. Each OD-Pair OD_i ($i = 1, \ldots, M$) is characterized by its router-level path P_i and the approximate number T_i of distinct IP-level flows on that path in a measurement interval (e.g., five minutes).[2] Each router R_j ($j = 1, \ldots, N$) is constrained by the available SRAM for keeping per-flow counters [13]; L_j denotes the number of flows R_j can record 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 [33] has two steps. First, we find the largest possible minimum fractional coverage per OD-pair $\min_i\{C_i\}$ subject to the resource constraints. Next, we use this as θ in Eq 3 in the linear program shown below and maximize the total flow coverage $\sum_i(T_i \times C_i)$.

$$\text{Maximize} \sum_i(T_i \times C_i), \text{ subject to}$$

$$\forall j, \quad \sum_{i:R_j \in P_i}(d_{ij} \times T_i) \leq L_j \tag{1}$$

$$\forall i, \quad C_i = \sum_{j:R_j \in P_i} d_{ij} \tag{2}$$

$$\forall i, \quad \theta \leq C_i \leq 1 \tag{3}$$

$$\forall i, \forall j, \quad d_{ij} \geq 0$$

The solution $d^* = \{d^*_{ij}\}$ to this two-step procedure yields

[2]We assume that each OD-pair has a single routing path. It is easy to extend the framework to accommodate multi-path routing or route changes [33].

the optimal sampling strategy. This solution is then translated into sampling manifests specifying the flow monitoring responsibility for each router.

2.3 Assumptions in cSamp

There are three main assumptions: (i) a centralized optimization module which has routing and traffic matrices, (ii) routers implement hash-based flow sampling, and (iii) routers obtain OD-pair information from packet headers.

The first two assumptions are feasible within current operational realities. First, centralization is viable if the configurations are generated reasonably quickly, say 1-2 minutes. Recent trends show that ISPs favor centralized network management [6, 2] and that routing and traffic matrices are readily available [1, 38]. Second, the requirements of hash functions for flow sampling are simple [10] and they are amenable to fast hardware implementations [27]. Flow sampling requires lookups for each packet and is feasible if the $Flowtable$ is in fast SRAM. Prior work shows that maintaining such counters is feasible [13, 19].

The assumption that routers can obtain OD-pair identifiers is crucial to cSamp's design. Specifically, Eq 2 implicitly assumes that the hash-ranges assigned to different routers for the same OD-pair are non-overlapping. This requires that the hash ranges be specified per OD-pair per router, which is not possible if routers cannot determine the OD-pair. In fact, this non-overlapping property makes the optimization problem a tractable linear program (since the coverage per OD-pair is the *sum* across the routers), which can be solved efficiently. If OD-pair identifiers were not available, this would no longer hold. As we argue next, this assumption is not practical for ISPs today.

2.4 Challenges in OD-pair identification

Given no additional information, a router needs to determine the ingress and egress routers (i.e., the OD-pair) for a given packet using only the packet's source and destination IP addresses and its local routing table. The feasibility of doing this depends on whether the ISP uses IP-based or MPLS-based forwarding. While IP forwarding is purely destination-based, MPLS can also take into account source information. However, we are unaware of deployments configured in this way, and we have confirmed that a large tier-one ISP's deployment of MPLS, for example, does not [3]. As such, here we restrict our attention to destination-based MPLS forwarding, which we believe to be the norm. Table 1 summarizes the feasibility of resolving the ingress and egress in these two scenarios.

Information to resolve	Routing/Forwarding	
	IP (dest-based)	MPLS (dest-based)
Ingress router	Difficult	Difficult
Egress router	Maybe, with some ambiguity	Possible

Table 1: Feasibility of resolving ingress and egress routers using packet headers and local routing tables.

In both cases, resolving the ingress is nearly impossible.

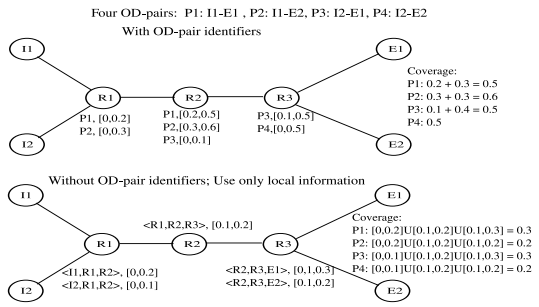


Figure 1: Example to show the intuition behind cSamp-T

For example, in the case of traffic entering from a multi-exit peer (i.e., a neighboring AS with which an ISP peers at multiple peering points), source IP address and routing table information cannot determine the ingress from which the packet arrived. Resolving the egress is relatively easier because forwarding is destination-based. With MPLS, the egress can be resolved exactly; with IP the egress can be resolved within some ambiguity. In IP forwarding, ingress/egress resolution may also be additionally complicated if interior routers only see aggregated prefix information.

Due to the above challenges, cSamp assumes that ingress routers explicitly add OD-pair identifiers to packet headers. However, this leads to a key deployment hurdle–it imposes additional effort on border routers (e.g., replicating the routing logic to resolve the egress router) and requires modifications to packet headers to carry OD-pair identifiers.

2.5 Motivating question

The above challenges in OD-pair identification bring us to the motivating question for our work: *Can we provide the flow coverage benefits of cSamp without requiring OD-pair identifiers?* In the next sections, we describe our approach, cSamp-T, to addresses this challenge.

3. PROBLEM STATEMENT

The main idea in cSamp-T is that each router makes sampling decisions using only *local information* rather than the global OD-pair identifiers. Since each router operates only with local information, the coverage of an OD-pair is obtained by "stitching" together the coverage provided by each router on the path.

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

The bottom half shows a scenario where routers cannot obtain OD-pairs and 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 determine the 3-tuple using only local information: the interface the packet arrived on, the destination IP, and the forwarding table, and then decide whether or not to sample the flow/packet. The coverage for each OD-pair is the *union*

of the ranges assigned for each 3-tuple on the path.

The example highlights two differences between cSamp-T and cSamp. First, the sampling responsibilities are specified using local information rather than global OD-pair identifiers. Second, the coverage for each OD-pair is no longer the sum across the routers 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 network-wide flow coverage objectives while respecting each router's resource constraints? The following sections present a formal framework to address this.

3.1 Problem Formulation

We borrow two assumptions from cSamp: (a) a centralized module with access to routing and traffic matrices assigns sampling responsibilities and (b) routers implement hash-based flow sampling using SRAM counters and SRAM size constrains the number of flows a router can log. As discussed earlier, both are reasonable assumptions. Next, we discuss how a centralized module can assign sampling responsibilities without OD-pair identifiers.

We introduce the notion of a *SamplingSpec* to capture the granularity at which a router makes sampling decisions. 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; i.e., R_{j_1} and R_{j_3} are neighbors of R_{j_2}. Let a_k denote a generic SamplingSpec.

$a_k \in P_i$ denotes that the SamplingSpec a_k lies 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 to the notion that a router R_j lies on a 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. SamplingSpecs are mapped to routers in a many-to-one fashion; we denote the set of SamplingSpecs mapped to R_j by R_j.specs. That is, R_j can be assigned sampling responsibilities corresponding to $a_k \in R_j$.specs. In the 3-tuple case, if $a_k = \langle R_{j_1}, R_{j_2}, R_{j_3}\rangle$, then $a_k \in R_{j_2}$.specs.

If R_j.specs $\ni a_k$, then R_j can log some of the traffic on paths $P_i \ni a_k$. But what fraction should it log? We formalize this by creating a set of *SamplingAtom*s. Suppose the traffic traversing a_k is mapped to the unit interval $[0,1]$ by hashing and that the interval $[0,1]$ is divided into $\frac{1}{\delta}$ equal-sized intervals $h_l = [(l-1)\delta, l\delta]$, of length δ. A SamplingAtom is a pair $\langle a_k, h_l\rangle$. If a SamplingAtom, $g_{kl} = \langle a_k, h_l\rangle, a_k \in R_j$.specs, is *assigned*, 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 synonym for h_l.

EXAMPLE: Figure 2 illustrates the above definitions with an example, where $\delta = 0.25$. $R3$ 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

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

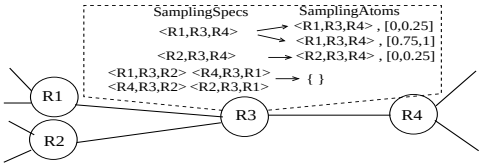


Figure 2: Example showing the SamplingSpecs and assigned SamplingAtoms at router R3.

for $\langle R1, R3, R4\rangle$, one for $\langle R2, R3, R4\rangle$, and none for $\langle R1, R3, R2\rangle$. 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 on paths of the form $\{.., R2, R3, R4, ..\}$ such that the hash of the flow falls in the range $[0, 0.25]$.

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

$$\text{coverage } C_i = \left|\bigcup_{a_k \in P_i} \bigcup_l\, h(\widehat{g_{kl}})\right| \tag{4}$$

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 adding, over all SamplingSpecs $a_k \in R_j$.specs, the portion of the traffic through a_k that R_j logs:

$$Load_j = \sum_{a_k \in R_j.\text{specs}} t_k \times \left|\bigcup_l\, h(\widehat{g_{kl}})\right| \tag{5}$$

Given the C_is, the specific functions we are interested in optimizing 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 f_{tot} or f_{min}, while operating within the router resource constraints (i.e., $Load_j \le L_j$ for all j). We choose these specific objective functions because of their use in cSamp [33]; our framework can accommodate a wider range of objective functions expressed as 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 (4) and router loads given by (5). Now, f_{tot} or f_{min} can be viewed as functions from subsets of $\mathcal{V}$ to the reals. The problem is to select an *optimal* $S^* \subseteq \mathcal{V}$, i.e., that maximizes f_{tot} or f_{min}, subject to $Load_j \le L_j$.

3.2 Exact Solutions are Hard

Finding an 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 3-SAT. Moreover, it is infeasible for practical system sizes. For example, consider the problem as an integer linear programming formulation using $\{0,1\}$ 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 after a day. Because of the intractability of solving the problem exactly, we use approximation algorithms. However, as we will see, the performance of our algorithms is comparable to the ideal performance of cSamp.

4. SUBMODULARITY AND ALGORITHMS

In the previous section, we saw that obtaining exact solutions for maximizing the total coverage or the minimum fractional coverage in cSamp-T is hard. In this section, we show that there are practical approximation algorithms to obtain the sampling strategies. The key insight is that the coverage functions have a natural "submodularity" property which allows us to extend results from the theory of maximizing submodular set functions.

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* [14]. 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 [14], 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, or greedily picks the elements that give the maximum marginal benefit *per unit element-cost* (depending on whether $cbflag$ is set to true or false), as long as the budget is not violated. The better of these two settings is a constant factor approximation algorithm [35].

4.2 Application to cSamp-T

It is easy to check the coverages C_i viewed as functions from $2^{\mathcal{V}} \rightarrow \Re$ where $\mathcal{V} = SamplingAtoms$ are monotone submodular, and hence so is their weighted sum $f_{tot} = \sum_i T_i C_i$.

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 $\psi_s \leftarrow \frac{F(S \cup \{s\}) - F(S)}{norm}$
6 $s^* \leftarrow \mathbf{argmax}_{s \in \mathcal{V} \setminus S} \psi_s$
7 $S \leftarrow S \cup \{s^*\}$
8 return $\langle S, F(S) \rangle$

Figure 3: Basic greedy algorithm

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$(\hat{F}, \mathcal{V}, \mathbf{true}, \infty)$
6 **if** MAXUSAGE$(B_{used}, B) > \gamma$
7 **then** $\tau_{upper} \leftarrow \tau_{current}$
8 **else** $\tau_{lower} \leftarrow \tau_{current}$
9 Return τ_{lower}

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

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. However, these budget constraints have a special structure. 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*, where each partition corresponds to a different router, and the knapsack constraint is the load constraint for that router. In Appendix B, we prove that a modified greedy algorithm—an extension of Figure 3—gives a constant-factor approximation.

Maximizing f_{tot}: To match the theoretical guarantees [35] (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 (§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 maximize $F^{\min}(S) = \min_i F_i(S)$, the *minimum* across these different functions. This new function $F^{\min}$ is not 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 to exceed the budget constraint by some factor [21]. Formally, if S^* is an optimal set satisfying the 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}, each router's sampling algorithm is as follows. The router no longer requires the OD-pair information for a packet; it gets the SamplingSpec using only the packet headers and other local information (e.g., what interface the packet arrives and leaves on). The router is assigned a set of non-contiguous hash ranges for each SamplingSpec and samples the packet if the hash of the flow 5-tuple falls in *any* of the ranges for this SamplingSpec.

4.3 Practical Issues

Reducing computation time: The computation time of the algorithm in Figure 3 can be reduced using the insight that for each element $s \in \mathcal{V}$, the marginal benefit ψ_s obtained by picking s decreases monotonically across iterations of the greedy algorithm. Thus, we can use *lazy evaluation* [24]. The main intuition behind lazy evaluation is that not all ψ_s values need to be recomputed in Figure 3 (Step 5); only a smaller subset that is likely to affect the choice of s^* in Step 6 needs to be computed. We omit details for brevity and refer the reader to [24]. We can replace all instances of the procedure call SUBMODULARGREEDY with the lazy evaluation version. §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 in terms of router three-tuples. Note, however, that our algorithms are generic and do not depend on SamplingSpecs being router three-tuples. Thus, we can generalize our results to other notions of SamplingSpecs. For example, the SamplingSpecs can be made coarser (e.g., ignore previous/next hop information and operate at a router granularity), or more fine-grained (e.g., add egress information to the 3-tuple if available).

Effects of Discretization: §3 defined a discretization interval δ such that $g_{kl} = \langle a_k, [(l-1)\delta, l\delta]\rangle$, $l \in \{1, \ldots, \frac{1}{\delta}\}$. There are two practical issues 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. Second, instead of $\frac{1}{\delta}$ 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. We avoid this overhead and instead run a simple merge procedure (§6.3) to compress the sampling manifests.

5. HEURISTIC EXTENSIONS

While the theoretical guarantee for f_{tot} is encouraging, the result for f_{min} requires fairly high resource augmentation values (γ) to get non-trivial guarantees.

In this section, we describe three practical extensions to improve the performance for f_{min}.

1. Targeted provisioning to use fewer additional resources.
2. Incremental deployment where some ingress routers are upgraded to add OD-pair identifiers.
3. Using alternative objective functions to approximate the max-min property.

We present these in the context of the f_{min} objective. However, the targeted provisioning and partial deployment techniques can be more generally applied to other network-wide objectives where the greedy algorithm performs poorly.

5.1 Intelligent Provisioning

The theoretical bound in §4 assumes that each router is given $\gamma\times$ more resources. However, it is expensive to add $\gamma\times$ more SRAM to *all* routers. Here, we want to add more SRAM intelligently–selectively augment some routers and still get good performance. The insight here is that it may suffice to upgrade a small number of heavily loaded routers.

$\mathbf{Prob}$. *provisioning* : Maximize $\mathbf{min}_i\ C_i$, subject to

$$\forall j, \textstyle\sum_{k:a_k\in R_j.\mathsf{specs}} u_k \times t_k \leq L_j \quad (6)$$

$$\textstyle\sum_j L_j \leq Budget \quad (7)$$

$$\forall j, LB_j \leq L_j \leq UB_j \quad (8)$$

$$\forall i, C_i = \textstyle\sum_{k:a_k\in P_i} u_k \quad (9)$$

$$\forall k, u_k \geq 0; \forall i, C_i \leq 1 \quad (10)$$

To address this, we consider the following provisioning problem. The operator gives a memory budget $Budget$ to be distributed across routers defined by her monetary budget and SRAM cost. Each router R_j has a lower bound LB_j for the default memory configuration and an upper bound UB_j

on the maximum amount that can be provisioned.[4] The output of the provisioning problem is the allocation of resources to routers to optimize f_{min}.

However, it is difficult to model the coverage C_i per OD-pair achieved by the greedy algorithm. Thus, we make a simplifying assumption that the hash ranges (the u_k values) across the different SamplingSpecs on a given path are non-overlapping. This allows us to model C_i as the sum of the u_ks in Eq 9. Under this assumption, the provisioning problem can be solved as a linear program. While this is less optimal compared to faithfully modeling the C_is, this is a reasonable assumption since our goal is to obtain general guidelines for resource provisioning. As we will see in §6.4, this heuristic works well in practice.

Given the memory allocations given by solving the LP, we run the greedy algorithm in Figure 4 with $\gamma = 1$ to ensure that we are strictly within the resource constraints.

5.2 Partial OD-pair identification

Next, we consider a scenario in which the network operator can upgrade some border routers. This can be achieved using a software update to the router or by adding a simple two-port middlebox [18] that processes each packet, modifies the header, and forwards them to the router. These few upgraded nodes (routers or router plus middlebox) have the ability to determine and add OD-pair 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 flow's hash falls in a hash-range corresponding *either* to the OD-pair or the SamplingSpec for the packet.

Prob. $enabledODs(\theta, \mathcal{P}_e)$: Minimize $\sum_j L_j$, subject to

$$\forall j, \sum_{i \in \mathcal{P}_e : R_j \in P_i} (d_{ij} \times T_i) \leq L_j \quad (11)$$

$$\forall i \in \mathcal{P}_e, C_i = \sum_{j : R_j \in P_i} d_{ij} \quad (12)$$

$$\forall i \in \mathcal{P}_e, \theta \leq C_i \leq 1; \ \forall i \in \mathcal{P}_e, \ \forall j, \ d_{ij} \geq 0 \quad (13)$$

Let $\mathcal{P}_e$ be the set of "enabled" OD-pairs whose packets carry OD-pair identifiers and let $\mathcal{P}$ be the set of all OD-pairs. As in Figure 4, we compute the maximum achievable minimum fractional coverage using binary search over the parameter τ. The key difference in the new algorithm 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 $\hat{F} = \sum_i \hat{C}_i$.

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 coverage $\theta = \tau_{current}$. The objective of the LP is to minimize the total resource used across the routers to ensure that each $OD_i \in \mathcal{P}_e$ has $C_i \geq \theta = \tau_{current}$. 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 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 non-enabled OD-pairs in the set $\mathcal{P} \setminus \mathcal{P}_e$ achieves a fractional coverage equal to $\tau_{current}$. If the greedy algorithm achieves this 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 function

f_{min} is difficult to optimize because the max-min function is not submodular. A natural heuristic is to design a function other than max-min that is submodular but (approximately) provides the desired max-min property, and then use SUBMODULARGREEDY to optimize this new function.

The notion of α-fairness has been used in congestion control [25] to generalize and approximate max-min fair allocation. Given items x_i, we want to allocate a given resource to the items "fairly". The α-fairness function is defined as $\sum_i U(x_i)$, where $U(x) = \frac{x^{1-\alpha}}{1-\alpha}$.[5] α can take values in $[0, \infty)$, and the values $\alpha = 0$ and $\alpha \to \infty$ correspond to maximum throughput and max-min fairness respectively.

In our context, each x_i corresponds to C_i. It is easy to verify that the function $\sum_i U(C_i)$ is submodular and we can use the greedy algorithm (Figure 3) to optimize this function. We use $\alpha = 100$ and also add a small constant to each C_i since $U(x)$ is undefined when $x = 0$. Note that while the two previous heuristics can be extended to other objectives, the α-fair heuristic is specific to maximizing f_{min}.

6. EVALUATION

Setup: We evaluate the performance of cSamp-T at a PoP-level granularity with each PoP as a node in the network. We use PoP-level topologies of educational backbones and tier-1 ISPs [30] (Table 2). We use shortest-path routing to construct paths between every OD-pair and model the traffic matrix using a gravity model based on city populations [29]. We assume that each PoP can log $L = 400,000$ flow records.[6] For cSamp-T, we discretize the hash-range with $\delta = 0.02$.

[4]There are natural technological limits on the amount of fast SRAM that can be added to linecards [34].

[5]At $\alpha = 1$, the function is $U(x) = \log(x)$

[6]Assuming 12 bytes per flow record [33], 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 [34].

Topology (AS#)	PoPs	OD-pairs	Flows $\times 10^6$	Packets $\times 10^6$
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

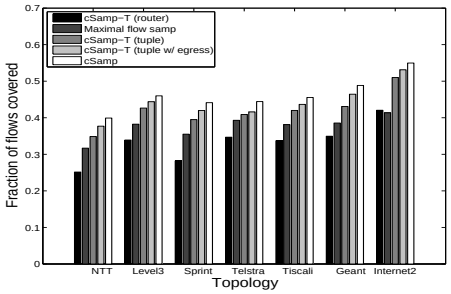


Figure 5: Total flow coverage

6.1 Coverage and Overlap

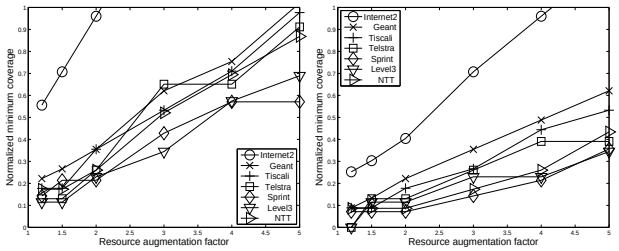
Total flow coverage: We consider three granularities of SamplingSpecs: router, router 3-tuple, and router 3-tuple augmented with egress information. Note that the first two SamplingSpecs can always be inferred from just local information but there may be some residual ambiguity in resolving the egress (Table 1). We consider the tuple+egress SamplingSpec as a hypothetical point in the solution space to see the gap between it and the other purely local solutions. We also compare these solutions to cSamp and a maximal uncoordinated flow sampling solution [33].[7]

Figure 5 shows that using 3-tuple SamplingSpecs provides significant improvement (25-30%) over the router-level case. cSamp-T (3-tuple+egress) is closest to cSamp, but the gap between the 3-tuple and egress-added cases is small. Also, the performance of the greedy algorithm is close to the theoretical upper bound for cSamp-T. (Not shown due to space constraints; please see our extended technical report [28] for additional results).

The theoretical guarantee for total flow coverage depends on running the two greedy algorithms: with and without the cost-benefit flag. We found that both configurations have similar performance and that the algorithm with the cost-benefit flag $cbflag =$ **false** is slightly better [28].

Minimum fractional coverage: §4 showed that it is impossible to maximize f_{min} without resource augmentation. Thus, we evaluate the performance as a function of the resource augmentation factor γ, where each router's budget is $\gamma \times 400,000$. We normalize the minimum fractional coverage by the optimal value achieved by cSamp at $\gamma = 1$; i.e., if cSamp-T has value 0.2 at $\gamma = 3$ and cSamp has value 0.4 at $\gamma = 1$, the y-value corresponding to $\gamma = 3$ is $\frac{0.2}{0.4} = 0.5$.

[7] In maximal flow sampling, each router maximally utilizes its memory. A router's flow sampling rate is $\min(1, \frac{l}{t})$, where l is the number of flows it can log and t is the number of flows it observes.



(a) Tuple (b) Router

Figure 6: Normalized minimum fractional coverage of cSamp-T with resource augmentation

Figure 6 shows the result for the router and tuple granularities. The tuple+egress was almost identical to the tuple case; we do not show this for brevity. First, with $\gamma \geq 4$, cSamp-T achieves $\geq 50\%$ of cSamp for all topologies. Second, the difference between the router and tuple formulations is more pronounced in the minimum fractional coverage result. With router-level SamplingSpecs, even at $\gamma = 5$, four out of the seven topologies only reach 40% of cSamp's performance. For the same $\gamma = 5$, using the 3-tuple SamplingSpecs, five out seven topologies achieve $\geq 90\%$ of cSamp's performance. Also, the γ at which cSamp-T has good performance is much better than the theoretical bound in §4. §6.4 shows that targeted provisioning reduces this even further.

Since 3-tuple SamplingSpecs perform much better than router SamplingSpecs, and are also very close to the tuple+egress case, we focus on 3-tuples in the rest of the paper.

Duplicated flow reports: A secondary objective in cSamp is avoiding duplicated flow reports to reduce the overhead in processing duplicated measurements. Uncoordinated sampling can have $\geq 30\%$ duplicated reports expressed as a fraction of unique flows logged. Compared to the uncoordinated case, cSamp-T with 3-tuples has $3\times$ fewer duplicated flow reports (not shown). Relative to cSamp which has no duplicate reports, this is not ideal; however, this is unavoidable since cSamp-T operates at a much coarser granularity.

6.2 Algorithm Running Time

In order for cSamp-T to be responsive to network dynamics, we want the time to compute sampling manifests to be within 1-2 minutes. (Manifests are recomputed across measurement epochs that span several minutes.) Table 3 shows that lazy evaluation provides more than an order of magnitude reduction in computation time compared to the vanilla greedy algorithm. The reduction is 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 because the hash-ranges assigned for each SamplingSpec need not be contiguous. As discussed in §4.3, we use a simple compression procedure to merge hash ranges after the greedy algorithm. This looks for maximally con-

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 lazy evaluation

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

tiguous hash ranges in the original sampling manifest and merges them into a single hash range.

Table 4 shows the overhead of disseminating manifests. First, we see that the compression procedure reduces the manifests roughly $10\times$. Second, we notice that the total bandwidth overhead after compression is small–25KB in the worst case. Finally, the worst case per-node manifest size is also small $\approx 3KB$.

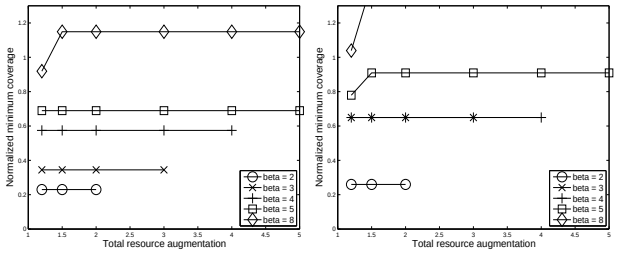
6.4 Intelligent Provisioning

As a specific scenario, we set $LB_j = L = 400,000$ for all j in the formulation from §5.1. We specify the total budget as $Budget = \gamma \times N \times L$, where N is the number of PoPs, and the technology limit as $\beta \times L$. We vary the parameters γ and β. Figure 7 shows the minimum fractional coverage normalized w.r.t cSamp for two topologies, Level3 and Telstra. We choose these because the greedy algorithm performs poorly compared to cSamp in Figure 6. An interesting result is that the curve levels off as a function of γ; i.e., increasing the total budget does not add much benefit. However, increasing the technology upper bound β provides significant improvement. In fact, even with a moderate total increase $\gamma = 1.2$, we see that the performance is within 80% of cSamp.

Since β is more crucial to than γ, for the remaining topologies we fix $\gamma = 1.5$ and analyze the normalized minimum fractional coverage as a function of β in Figure 8. With $\beta = 5$, all topologies achieve $\geq 60\%$ of cSamp's performance. Contrasting this with Figure 6, the main difference is that we do not require all PoPs to be augmented with $5\times$ more resources – the total resource budget is $\leq 1.5\times$.

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



(a) Level3 (b) Telstra

Figure 7: Understanding the impact of total resource augmentation (γ) and technology upper bound (β) in the resource allocation formulation.

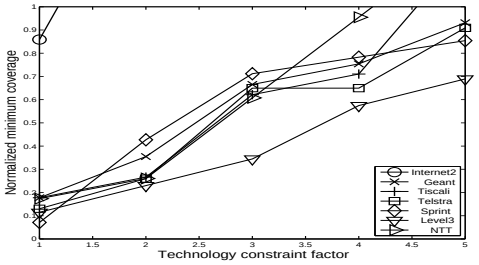


Figure 8: Intelligent allocation with varying β at γ=1.5

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 §5.2 for all values in $1, \ldots, k$ and pick the configuration with the highest f_{min}.

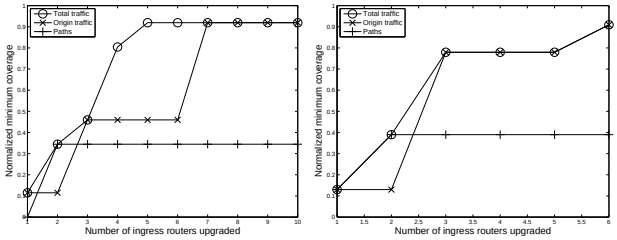
Figures 9(a) and 9(b) show the normalized minimum fractional coverage for the Level3 and Telstra topologies as a function of k (number of top-k PoPs). First, we observe that upgrading just a small number of PoPs ($< 8\%$) significantly improves the performance. Second, enabling identifiers on nodes that observe the most traffic performs much better than the other two strategies.

6.6 Using α-fairness

Figure 10 shows the normalized minimum fractional coverage with the α-fairness function with both naive (as in Figure 6) and intelligent augmentation (as in Figure 8). For each resource configuration, we run the greedy algorithm to optimize the α-fairness function from §5.3, find the minimum fractional coverage in the output of the greedy algorithm, and then normalize it w.r.t cSamp at baseline provisioning. Compared to Figures 6 and 8, the overall performance using the α-fairness optimization is slightly better.

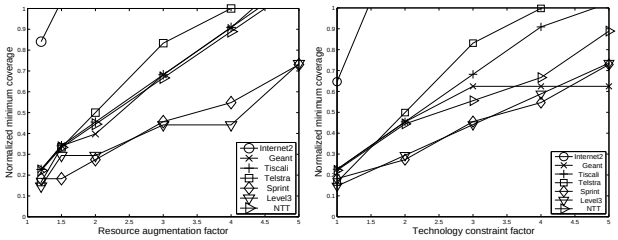
6.7 Hybrid Coverage Objective

cSamp maximizes a hybrid objective: maximizing the total flow coverage subject to achieving the highest minimum fractional coverage across OD pairs. So far, in cSamp-T we considered these two objectives separately. A natural question is if we can maximize this hybrid objective also in cSamp-T. It is relatively simple to extend the algorithm in Figure 4 to achieve this. First, run the greedy algorithm to optimize the capped minimum fractional objective ($\hat{F}$) and then modify the objective function to optimize the total cov-



(a) Level3 (b) Telstra

Figure 9: Performance of cSamp-T with partial OD-pair identification. Alternatively, this can be viewed as incremental deployment of cSamp via cSamp-T.



(a) Naive augmentation (b) Intelligent augmentation

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

erage if $\tau_{current}$ is feasible.

To evaluate this hybrid approach, we consider the resource configuration using targeted provisioning 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 procedure. Maximizing the minimum fractional coverage alone does not work well for the total coverage. This is because the greedy algorithm terminates when it achieves the target coverage for all OD-pairs even if there more resources that can 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 this case, Table 5 shows that the two-step optimization works well in practice.

6.8 Evaluation Summary

- The greedy algorithm for maximizing the total flow coverage is close to cSamp's performance and also close to the theoretical upper bound for cSamp-T.
- The two versions of the greedy algorithm for total flow coverage perform similarly.
- cSamp-T has $3\times$ fewer duplicated reports than an uncoordinated approach.
- With intelligent resource augmentation, cSamp-T needs $\leq 1.5\times$ as much total SRAM as cSamp to get $\geq 80\%$ of cSamp's optimum minimum fractional coverage.
- cSamp-T provides an incremental deployment path for cSamp; upgrading a small number ($\leq 8\%$) of ingresses to add OD-pair identifiers gets close to 90% of the ideal cSamp performance.

Topology	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

- The α-fairness heuristic only marginally improves the minimum fractional coverage.
- Lazy evaluation reduces the computation time by 1-2 orders of magnitude and enables cSamp-T to scale to large tier-1 ISP topologies.
- The total size of sampling manifests in cSamp-T is small – $\leq$25KB for the entire network after a simple compression step to merge hash ranges.
- cSamp-T also achieves near-ideal performance for the two-step objective in cSamp.

7. DISCUSSION

More fine-grained local information: One possibility is to provide additional hints to routers, e.g., distributing IP-prefix to ingress-egress maps to routers [5], to enable more fine-grained sampling decisions. This can bring the performance of cSamp-T even closer to cSamp.

Sensitivity of router upgrades: §5 provides two heuristics for upgrading routers with more resources or the ability to add OD-pair identifiers. These 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 dynamics. Thus, we can apply these formulations using 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. OTHER RELATED WORK

The closest related work is cSamp [33], which we discussed in §2. Here, we discuss other related work.

Sampling solutions: Most of the related work focuses on the single-router case to work around the limitations of packet sampling. This includes work on adaptive sampling [12], inverting sampled measurements [11, 17], and data streaming algorithms (e.g., [13, 22]).

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 [7, 31]. The authors show that this is NP-hard and propose greedy algorithms. These formulations can be extended to incorporate packet sampling [31, 15]. 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 functions in sensor networks [16, 20]. The problem of placing sensors robust to adversarial objectives [21] is conceptually similar to maximizing the minimum fractional coverage.

9. CONCLUSIONS

cSamp is a recent proposal to meet the increasing demand for fine-grained flow monitoring capabilities in network management. However, ISPs cannot realize the benefits of cSamp in practice today because of its reliance on OD-pair identifiers; it requires modifications to packet headers and imposes additional overhead at ingress routers, and may require ISPs to overhaul their routing infrastructure.

This paper was motivated by the challenge of providing the benefits of cSamp without relying on OD-pair identifiers. To address this, we presented cSamp-T, in which the sampling decisions at routers are based only on local information, and do not rely on global OD-pair identifiers.

We show that obtaining exact solutions to the maximize the total flow coverage (f_{tot}) and minimum fractional coverage (f_{min}) is NP-hard. We achieve near-optimal performance for f_{tot} by leveraging its submodularity. For f_{min}, getting good performance without resource augmentation is provably hard. However, targeted provisioning achieves near ideal performance with low overhead. Alternatively, upgrading a small number of border routers to provide OD-pair information also yields good results.

cSamp-T thus makes the benefits of coordinated network-wide monitoring solutions like cSamp practically and more immediately available to ISPs and also provides an incremental deployment path for ISPs to transition to cSamp.

10. REFERENCES

[1] A. Feldmann et al. Deriving Traffic Demands for Operational IP Networks: Methodology and Experience. In *Proc. of ACM SIGCOMM*, 2000.

[2] A. Greenberg et al. A Clean Slate 4D Approach to Network Control and Management. *ACM SIGCOMM CCR*, 35(5), Oct. 2005.

[3] Personal Communication with Aman Shaikh, AT&T Research.

[4] A. Anand, V. Sekar, and A. Akella. SmartRE: An Architecture for Coordinated Network-Wide Redundancy Elimination. In *to appear in Proc. SIGCOMM*.

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

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

[7] C. Chadet et al. Optimal Positioning of Active and Passive Monitoring Devices . In *Proc. of CoNeXT*, 2005.

[8] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a submodular set function subject to a matroid constraint. In *Proc. of IPCO*, 2007.

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

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

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

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

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

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

[15] G. R. Cantieni et al.,. Reformulating the Monitor Placement problem: Optimal Network-Wide Sampling. In *Proc. of CoNeXT*, 2006.

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

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

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

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

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

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

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

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

[28] V. Sekar, A. Gupta, M. K. Reiter, and H. Zhang. Coordinated Sampling sans Origin-Destination Identifiers: Algorithms, Analysis, and Evaluation . Technical Report, CMU-CS-09-104, Carnegie Mellon University, 2009.

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

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

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

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

[33] V. Sekar et al. cSamp: A System for Network-Wide Flow Monitoring. In *Proc. of NSDI*, 2008.

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

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

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

[37] Y. Xie et al. Worm Origin Identification Using Random Moonwalks. In *Proc. of IEEE Symposium on Security and Privacy*, 2005.

[38] 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.

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

APPENDIX

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 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 $\{\langle T_j, X_j\rangle\} \cup \{\langle F_j, X_j\rangle\} \cup \{\langle X_j, D_j\rangle\} \cup \{\langle D_j, T_{j'}\rangle | j' > j\} \cup \{\langle D_j, F_{j'}\rangle | j' > j\}$.

A SamplingSpec a_k can be one of $\langle T_j, X_j, D_j\rangle$, $\langle F_j, X_j, D_j\rangle$, $\langle X_j, D_j, T_{j'}\rangle$, 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

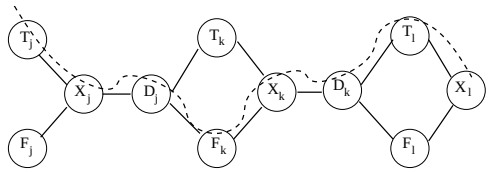


Figure 11: Example showing the path corresponding to the clause $C_i = (x_j \vee \overline{x_k} \vee x_l)$

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 $\max(\#\text{clauses with } x_j, \#\text{clauses with } \overline{x_j})$.

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. 11.

Claim: 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. APPROXIMATION 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 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 [8].

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 [35]. In fact, an algorithm based on partial enumeration [32] gets an optimal $(1 - e^{-1})$-approximation.

We can combine these ideas to solve the problem of "submodular maximization with 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. We can 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, gives 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.