Net2Text: Query-Guided Summarization of Network Forwarding Behaviors

Rüdiger Birkner, Dana Drachsler-Cohen, Laurent Vanbever, and Martin Vechev, *ETH Zürich*

https://www.usenix.org/conference/nsdi18/presentation/birkner

This paper is included in the Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18).

April 9–11, 2018 • Renton, WA, USA

ISBN 978-1-931971-43-0

Open access to the Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation is sponsored by USENIX.

Net2Text: Query-Guided Summarization of Network Forwarding Behaviors

Rüdiger Birkner, Dana Drachsler-Cohen, Laurent Vanbever, Martin Vechev
ETH Zürich

net2text.ethz.ch

Abstract

Today network operators spend a significant amount of time struggling to understand how their network forwards traffic. Even simple questions such as *"How is my network handling Google traffic?"* often require operators to manually bridge large semantic gaps between low-level forwarding rules distributed across many routers and the corresponding high-level insights.

We introduce *Net2Text*, a system which assists network operators in reasoning about network-wide forwarding behaviors. Out of the raw forwarding state and a query expressed in natural language, *Net2Text* automatically produces succinct summaries, also in natural language, which efficiently capture network-wide semantics. Our key insight is to pose the problem of summarizing ("captioning") the network forwarding state as an optimization problem that aims to balance *coverage*, by describing as many paths as possible, and *explainability*, by maximizing the information provided. As this problem is NP-hard, we also propose an approximation algorithm which generates summaries based on a sample of the forwarding state, with marginal loss of quality.

We implemented *Net2Text* and demonstrated its practicality and scalability. We show that *Net2Text* generates high-quality interpretable summaries of the entire forwarding state of hundreds of routers with full routing tables, in few seconds only.

1 Introduction

Put yourself in the shoes of a network operator working for a large transit provider: you just received a call from one of the largest Content Delivery Networks (CDN) informing you that they observed bad performance for flows crossing your network. As a cautious operator, you run the latest control- and data-plane verification technologies and are confident that your network state is correct; you suspect a runtime problem. You start by collecting the CDN routing advertisements and identify a dozen of possible egress points used to reach them together with hundreds of ingresses. Analyzing some of the internal paths, you do not observe any signs of loss. Looking at traffic volumes, you realize that most of the CDN traffic leaves via one egress connected to an Internet Exchange Point (IXP). You suspect congestion inside the fabric (invisible to you). Indeed, lowering the preference for the CDN prefixes at the IXP solves the problem.

This example is loosely based on a real troubleshooting scenario observed at a Tier 1 and illustrates the challenges in understanding and reasoning about network-wide forwarding behavior. The main issue lies in the large semantic gaps that separate low-level forwarding rules distributed across the entire network and actionable high-level insights by network operators. Bridging this gap manually (the default nowadays) is slow. Reasoning about network behavior often takes hours (e.g., for the case above)—even for the most skilled network operator. As networks grow more complex (e.g. as the number of peering increases), so does the corresponding reasoning time. This tension is becoming even more palpable as networks carry more and more critical services.

The example also illustrates that human insights and domain-specific knowledge are *fundamental* for understanding non-trivial unwanted network behavior. Even if the network control- [1–4] and data-plane [5–10] are formally verified, subtle problems will arise at runtime. Here, no observable signs were available to the operator. The goal is therefore not to remove the human out of the loop, but instead to assist her.

Net2Text In this paper, we introduce *Net2Text*, an interactive system which assists the network operator in reasoning about network-wide forwarding state. Out of the low-level forwarding state and a query expressed in *natural language*, *Net2Text* automatically produces succinct, *natural language* descriptions, which efficiently capture network-wide semantics.

Relying on natural language ensures seamless human interactions. We think of *Net2Text* as a "chatbot" for networks. We confirmed the usefulness of such interfaces with the network operators themselves: all of them liked the idea of having natural language descriptions of their network. Of course, *Net2Text*'s reasoning capabilities could also be integrated with other high-level interfaces such as graphs [11].

Coming back to the example above, the operator could simply ask *Net2Text*: *"What happens to the traffic destined to CDN X?"* to which *Net2Text* would answer: *"Traffic enters via n ingresses and mostly (85%) leaves via IXP 1. Traffic is load-balanced between A and B."*. Using this high-level insight, the operator could then ask for more targeted information: *"Tell me more about the CDN traffic leaving via IXP 1"*.

The main challenge behind *Net2Text* is to generate concise summaries which "explain"' as much as possible out of the network forwarding state. We formulate this as an optimization problem that aims to balance *coverage* and *explainability* and show that it is *NP*-hard.

Fortunately, we show that the skewness of the network forwarding state (i.e. its inherent redundancy) makes it well-amenable to summarization in practice. This motivates us to focus on a subspace of solutions which we can prove contains good solutions. An important property of this subspace is that every search path is of polynomial size. This enables us to design an approximation algorithm that traverses the space efficiently.

We designed and implemented *Net2Text*. *Net2Text* takes a query in natural language, parses it to a database query, runs the query on a network database, summarizes the results, and translates the summarization back to natural language. The operator can then pose follow-up queries, and thereby interactively guide *Net2Text* towards producing summaries focusing on particular aspects of the network.

We evaluated *Net2Text* on a variety of realistic networks. Our results demonstrate that *Net2Text* is practical: it generates high-quality interpretable summaries of the entire forwarding state of hundreds of routers and full routing tables, in few seconds only.

Contributions Our main contributions are:

- A precise formulation of the network-wide summarization problem as an optimization problem (§4).
- An approximation algorithm for generating high-quality summaries (§5,§6), which scales to large data sets, and a translation of the abstract summaries to a description in natural language (§7).
- An implementation of *Net2Text*, along with a comprehensive evaluation. Experiments show that *Net2Text* can derive summaries for backbone networks with full routing tables within seconds (§9).

2 Overview

Consider an operator wondering how her network is forwarding traffic towards Google:

> *"How is Google traffic being handled?"*

Net2Text automatically parses the question expressed in natural language and produces a concise description (also in natural language) of the current forwarding behavior observed for Google:

> *"Google traffic experiences hot-potato routing. It exits in New York (60%) and Washington (40%). 66% of the traffic exiting in New York follows the shortest path and crosses Atlanta."*

Producing such a summary is challenging: the system has to understand what the operator is interested in, extract the relevant information, summarize it, and then translate it to natural language. Extracting this information goes beyond simply querying a database: it requires processing the data to identify common path features (e.g., the New York and Washington egresses) as well as high-level features pertaining to different paths (e.g., hot-potato routing, shared waypoints). In addition, the entire process should be quick (even if the network is large) to guarantee interactivity and deal with traffic dynamics.

In the following, we give a high-level overview of how *Net2Text* manages to solve these challenges and go from the above query to the final summary using a three-staged process (see Fig. 1).

Parsing operator queries in natural language (§8) *Net2Text* starts by parsing the operator query in natural language using a context-free grammar. This grammar defines a natural language fragment consisting of multiple network features (e.g., ingress, egress, organization, load-balancing) and possible feature values (e.g., New York, Google) allowing a network operator to express a wide range of queries. Our grammar consists of ~ 150 derivation rules which are extended with semantic inference rules to infer implicit information. In the above example, our grammar infers that the operator refers to traffic destined to the *organization* Google. *Net2Text* also understands other kinds of queries: *(i)* yes/no queries, "*Does* all traffic to New York go through Atlanta?"; *(ii)* counting queries, "*How many* egresses does traffic to Facebook have?"; and *(iii)* data retrieval queries, "*Where* does traffic to New York enter?". Our grammar is extendable with new features, keywords, and names.

Net2Text maps the parsed query to an internal query language, similar to SQL. Here, the query is mapped to: `SELECT * FROM paths WHERE org=GOOGLE`. This query is then run over a network database that stores the entire forwarding state of the network. Afterwards, the results are passed to the core part of *Net2Text*: the summarization module.

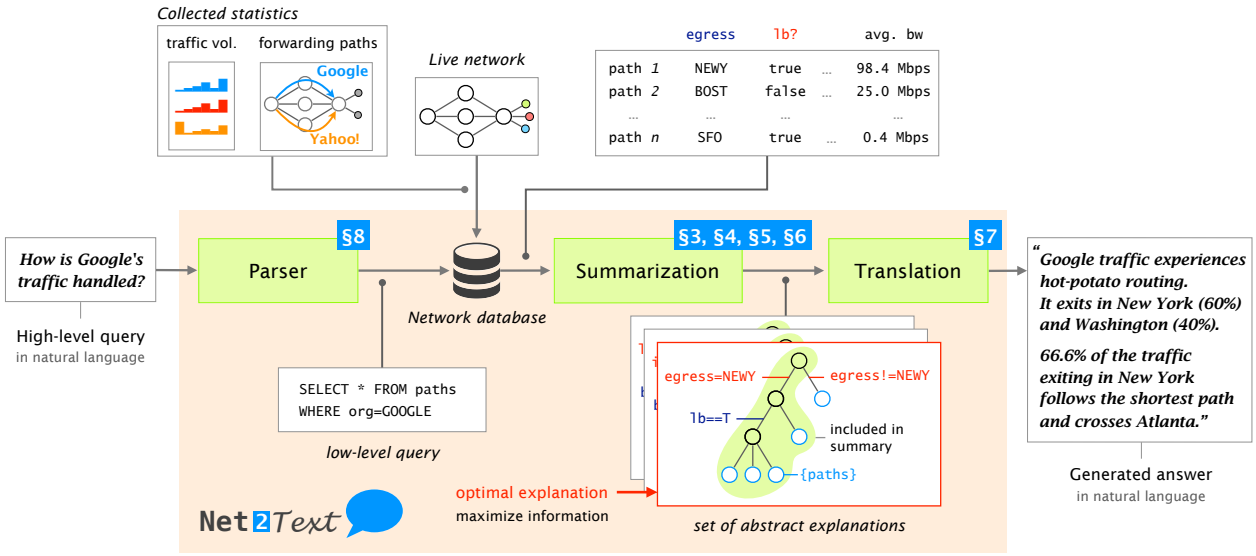


Figure 1: *Net2Text*: Workflow and key components.

Summarizing forwarding states (§4, §5, and §6) Most queries (including the one above) can and will return a plethora of low-level forwarding entries. *Net2Text* assists the operator in reasoning about forwarding state by automatically generating high-quality interpretable summaries out of low-level forwarding entries.

Summarizing network-wide forwarding states requires overcoming a fundamental tradeoff between *explainability* (how much detail a summary provides) and *coverage* (how many paths a summary describes). By defining a score function capturing both concepts analytically, we show that we can formally phrase this problem as an *NP*-hard optimization problem (§4). This renders both exhaustive techniques along with techniques based on Integer Linear Programming impractical.

To scale, we leverage the insight that traffic is skewed (heavy-tailed) across multiple levels: in the traffic distribution itself (few prefixes are typically responsible for most of the traffic [12]) or at the routing level (network topologies are usually built following guidelines, leading to repetitive forwarding patterns, e.g., edge/aggregation/core). This insight enables us to design an approximate summarization algorithm, called ComPass (§6), which explores a reduced search space that we can prove contains good summaries (§5). In addition, we show that ComPass can only summarize a sample of the forwarding entries instead of all of them with only a marginal loss in summarization quality.

Taken together, the reduced space and sampling optimization enable *Net2Text* to generate high-quality interpretable summaries for large networks (with hundreds of routers) running with full routing tables *in less than 2 seconds* (§9).

Converting path specifications to concrete text (§7) Given a set of path specifications, *Net2Text* finally produces a summary expressed in natural language in two steps. It first extends the set with additional properties inferred by examining the specifications as a whole. For example, if the specifications imply that there are multiple paths between the egress and ingress, *Net2Text* infers that the traffic is load balanced. *Net2Text* then maps the extended specifications to sentences in natural language.

3 Preliminaries

We now introduce the key terms we use in the paper.

Routing paths We model the network as a graph and define a network path P as a finite sequence of links. A routing path (d,P) is a pair of an IP prefix and a path, which describes that traffic to prefix d can be routed on P (a prefix can be routed on multiple paths). We denote by $\mathcal{R}$ the set of all routing paths in the network.

Feature functions Feature functions describe path features. Formally, a feature function $q\colon \mathcal{R} \to U_q$ maps routing paths $\mathcal{R}$ to *feature values* from U_q. We denote by v_q a value in U_q. We focus on the following feature functions. Organization $O\colon \mathcal{R} \to U_O$ maps every (d,P) to the organization owning d, e.g., Google. Egress $E\colon \mathcal{R} \to U_E$ maps every (d,P) to the egress of P, and ingress $I\colon \mathcal{R} \to U_I$ maps to P's ingress. Shortest path $SP\colon \mathcal{R} \to \{0,1\}$ maps to 1 if P is a shortest path between its ingress and egress, and 0 otherwise. We use the subscripts e, i, o, and sp to denote feature values of the egress, ingress, organization, and shortest path feature functions, e.g., New York$_e \in U_E$ and $1_{sp} \in U_{SP}$.

Path specifications To explain the behavior of the network and its routing paths, we define the concept of sets of feature values called *path specifications*. Given a set of l feature functions with disjoint ranges $U_1, ..., U_l$ [1] and a bound t (for $t \leq l$), a path specification is a (nonempty) set of feature values where the size of the set is at most t and each feature value describes a different feature function. Formally, a path specification is an element in:

$$\mathcal{S}^t_{U_1,...,U_l} = \bigcup_{1 \leq m \leq t} \bigcup_{1 \leq j_1 < ... < j_m \leq l} U_{j_1} \times ... \times U_{j_m}$$

Since the order of the feature values is not important for our needs, we treat path specifications as sets, e.g., $S_{G,NY} = \{\text{Google}_o, \text{New York}_e\}$.

We say a routing path (d,P) meets a path specification S, denoted $(d,P) \models S$, if for *every* feature value $v \in S$, if $v \in U_q$ for a feature function q, then $q(d,P) = v$. We define a specification set $\mathcal{S}$ as a set of path specifications, i.e., $\mathcal{S} \subseteq \mathcal{S}^t_{U_1,...,U_l}$. A routing path (d,P) meets a specification set $\mathcal{S}$, if there exists $S \in \mathcal{S}$ such that $(d,P) \models S$.

4 Problem Definition

Here, we formally phrase the problem of explaining network behaviors as an optimization problem.

Our goal is to find a summary of a (large) set of routing paths in the form of path specifications. The main challenge then is to infer a specification set that describes as many routing paths as possible while providing maximal amount of information about them. To evaluate the quality of a specification set, we define a score function. Intuitively, the score of a specification set is the sum of the "amount of explanation" of its routing paths. Given a score function, we formulate the problem of network summarization as constraint optimization.

We phrase our optimization problem as a MAP inference task [13], in which the goal is to find an assignment that maximizes a score while satisfying a set of constraints. In our context, an assignment consists of (up to) k path specifications each with at most t feature values and over feature functions $q_1, ..., q_l$. The score of an assignment is the weighted sum of the features the assignment describes for every routing path in $\mathcal{R}$. We define the score in two steps: *(i)* the score of a feature function $q \in \{q_1, ..., q_l\}$ and *(ii)* the score of all feature functions.

Feature score A score function of a feature function q maps sets of up to k specifications to a real number score:

$$\Phi_q \colon \left(\mathcal{S}^t_{U_1,...,U_l} \cup \{\emptyset\}\right)^k \to \mathbb{R}$$

The domain consists of k-ary tuples, whose elements are specification sets or the empty set. The empty set $\emptyset$ denotes "no specification", and it enables us to cleanly capture specification sets with less than k specifications. To simplify definitions, we assume: $(d,P) \not\models \emptyset$ for all (d,P). For a set $\mathcal{S}$, the score $\Phi_q(\mathcal{S})$ is the weighted sum of routing paths in $\mathcal{R}$ for which q is described by a specification in $\mathcal{S}$. A path (d,P) is part of the sum if there is a specification $S \in \mathcal{S}$ containing a feature value of q that (d,P) satisfies. The weight of a path $w_{d,P}$ is a positive number (e.g., the traffic size). Formally:

$$\Phi_q(\mathcal{S}) = \sum_{(d,P) \in \mathcal{R}} w_{d,P} \cdot [\bigvee_{S \in \mathcal{S}:\ q(d,P) \in S} (d,P) \models S] \quad (1)$$

In this definition, $[\cdot]$ denotes the Iverson bracket that returns 1 if the formula is satisfied or 0 otherwise.

Specification set	Φ_E	Φ_{SP}	$\Phi_{E,SP}$
$\{\{\text{NY}_e\}\}$	1	0	1
$\{\{\text{LA}_e\}\}$	2	0	2
$\{\{1_{sp}\}\}$	0	3	3
$\{\{\text{NY}_e\}, \{\text{LA}_e, 1_{sp}\}\}$	3	2	5

Table 1: Score functions for $\mathcal{R} = \{(d_1,P_1), (d_2,P_2)\}$, where $w_{d_1,P_1} = 1$ and $w_{d_2,P_2} = 2$, $E(d_1,P_1) = \text{NY}_e$ and $E(d_2,P_2) = \text{LA}_e$, and $SP(d_1,P_1) = SP(d_2,P_2) = 1_{sp}$.

Example 1 Table 1 shows an example for $\mathcal{R} = \{(d_1,P_1), (d_2,P_2)\}$ with $w_{d_1,P_1} = 1$ and $w_{d_2,P_2} = 2$. Assume $E(d_1,P_1) = \text{NY}_e$, $E(d_2,P_2) = \text{LA}_e$, and that P_1 and P_2 are shortest paths: $SP(d_1,P_1) = SP(d_2,P_2) = 1_{sp}$. Then, $\Phi_E(\{\{\text{NY}_e\}\}) = 1 \cdot 1 + 2 \cdot 0 = 1$ and similarly $\Phi_E(\{\{\text{LA}_e\}\}) = 1 \cdot 0 + 2 \cdot 1 = 2$. Since both P_1 and P_2 are shortest paths, $\Phi_{SP}(\{\{1_{sp}\}\}) = 1 \cdot 1 + 2 \cdot 1 = 3$. However, $\Phi_{SP}(\{\{\text{NY}_e\}, \{\text{LA}_e, 1_{sp}\}\}) = 1 \cdot 0 + 2 \cdot 1 = 2$

Feature set score A score function of feature functions $q_1, ..., q_l$ maps k specifications of size at most t to a score:

$$\Phi_{q_1,...,q_l} \colon \left(\mathcal{S}^t_{U_1,...,U_l} \cup \{\emptyset\}\right)^k \to \mathbb{R}$$

The score is the sum of all the features' scores:

$$\Phi_{q_1,...,q_l}(\mathcal{S}) = \sum_{j \colon [1,l]} \Phi_{q_j}(\mathcal{S})$$

The last column of Table 1 shows the feature set score of the previous example. We can now define the optimization problem.

Definition 1 (Optimization Problem) Given a set of routing paths $\mathcal{R}$, weights $w_{d,P}$ for each $(d,P) \in \mathcal{R}$, a set of feature functions $q_1, ..., q_l$, a constant k limiting the number of path specifications, and a constant t limiting the size of path specifications, we formulate the network summarization problem as:

$$\underset{\mathcal{S} \in (\mathcal{S}^t_{U_1,...,U_l} \cup \{\emptyset\})^k}{\arg\max} \quad \Phi(\mathcal{S})$$

[1] This is not a limitation, because values can be uniquely annotated.

Example 2 Let $\mathcal{R} = \{(\text{Google}, P_i)\}_{i=1}^{100}$, each with weight 1, and $k = t = 3$. We assume that *(i)* if $i \leq 60$, $E(\text{Google}, P_i) = \text{NY}_e$, and $E(\text{Google}, P_i) = \text{LA}_e$ otherwise, *(ii)* for $i \leq 40$, $SP(\text{Google}, P_i) = 1_{sp}$, and *(iii)* all other feature values are unique for every path. An optimal solution is: $\{\{\text{NY}_e\}, \{\text{Washington}_e\}, \{\text{NY}_e, 1_{sp}\}\}$, and its score is $\Phi_E + \Phi_{SP} = 100 + 40 = 140$. Another optimal solution is $\{\{\text{NY}_e\}, \{\text{Washington}_e\}, \{1_{sp}\}\}$. Though scores are identical, the operator is likely to prefer the former specification set as it provides additional information (e.g., all traffic following the shortest path exits in New York). We leverage this insight in §5.

Definition 1 can be refined by extending the objective function or adding constraints, as demonstrated in the next section. Also, while this problem can be considered as a general summarization problem suitable for other contexts, the skewed nature of traffic makes our context a better instantiation to this problem: the heavy traffic is likely to share many feature values which can lead to solutions that are clearly better than others. At the same time, these properties are exactly the kind of information that an operator needs in order to understand the behavior of the main part of the traffic.

One approach to solving this optimization problem is to phrase it as an integer linear program and use an off-the-shelf solver. We show such a formulation and a performance evaluation in Appendix A. Computing an exact solution to this NP-hard optimization problem is (expectedly) too expensive for practical use when summarizing a large number of paths. Instead, we introduce an approximate and scalable optimization algorithm, which we describe in the next sections.

5 Approximate Optimization

A key challenge when designing an inference algorithm for an NP-hard problem is dealing with the size of the search space that is at least exponential. In our setting, we show that the search space is exponential in both t and k, making the search very challenging (§5.1). Intuitively, this stems from the fact that we need to explore two dimensions: path coverage and path explainability. To address the issue with the large search space, we leverage the fact that traffic is skewed and focus on parts of it, enabling us to trade-off expressivity of the specification set with the size of the search space. We show that the optimal solution for this part of the search space: *(i)* has at least $min\{1/k, 1/t\}$ of the score of the optimal solution for the full search space, *(ii)* the length of every search path is polynomial in t, and *(iii)* the number of children of every node is polynomial in the number of feature values (§5.2). We further identify an equivalence relation over the path specifications and leverage it to define a search space with solutions of higher quality (§5.3).

5.1 An Exponential Search Space

In this section, we analyze the size of the search space, organize the solutions in a graph, and discuss the challenges of traversing it.

Size of search space We begin with showing that the size of the search space is exponential in t and k. The search space is the set of all specifications, that is $(\mathcal{S}^t_{U_1,\ldots,U_l} \cup \{\emptyset\})^k$. Thus, it immediately follows that its size is exponential in k. To conclude that the size is exponential in k and t, we show that the size of $\mathcal{S}^t_{U_1,\ldots,U_l}$ is exponential in t. To prove this, we reduce this computation to the combinatorial problem of choosing without replacement up to t feature functions from l feature functions (we assume $l \geq t$) and then for each, picking a feature value (we assume $|U_i| \geq 2$ for all i). Then, using a combinatorial identity [14, Vol. 2, (1.37)] we get:

$$\sum_{m=0}^{t} \binom{l}{m} \cdot 2^m \geq \sum_{m=0}^{t} \binom{t}{m} \cdot \frac{2^m}{m+1} = \frac{3^{t+1}-1}{2(t+1)}$$

Search space as a graph We organize the solutions in a directed graph $\mathcal{G}$. The nodes of $\mathcal{G}$ are the solutions: $(\mathcal{S}^t_{U_1,\ldots,U_l} \cup \{\emptyset\})^k$. There is an edge (u, v) if v extends one of u's specifications with one feature value (Fig. 2). We distinguish between two kinds of edges: edges that extend an empty specification (colored blue) and those that extend an existing specification (colored red). Intuitively, the blue edges try to increase *coverage* by including more path specifications. This increases the number of routing paths for which the overall specification set holds. The red edges aim to increase the amount of detail captured in a path specification, resulting in better *explainability*. However, they can reduce the number of routing paths that satisfy the specification set (and thus, have the opposite effect of blue edges). Two extreme points in this coverage versus explainability exploration are: *(i)* specification sets that maximize explainability (specifications are of size t) and *(ii)* specification sets that maximize coverage (all specifications are of size 1). Depending on the weights and number of routing paths, the optimal solution sits in-between these two extremes.

Example 3 $\{\{\text{New York}_e\}\}$ maximizes coverage, while $\{\{\text{New York}_e, \text{Dallas}_i, \text{Google}_o, 1_{sp}\}\}$ explainability.

Search challenge An important ingredient in any search strategy is the solution scoring function, which guides the search towards the optimal result, while effectively pruning subspaces. In our setting, such a score function is even more critical as the size of the search space is exponential in k and t. An immediate candidate for a score function is Φ, as in Definition 1. However, Φ can guide us towards a good solution only if we restrict our traversal to nodes reachable through the blue edges. This is due

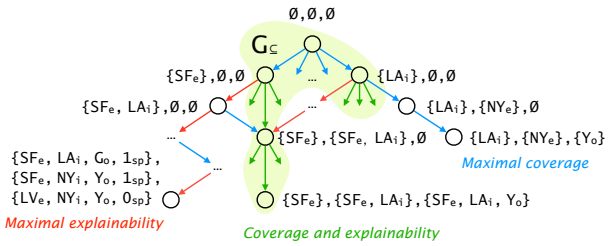


Figure 2: Part of the search space for k=3 specifications, t=4 feature values per specification and features egress *(e)*, ingress *(i)*, organization *(o)* and shortest path *(sp)*.

to a monotonicity property guaranteeing that if v is reachable from u only through blue edges, then $\Phi(v) > \Phi(u)$ (since v includes all feature values described by u). However, the red edges do not have this property for Φ (as it trades off path coverage with explainability). Even if we consider a different scoring function, pruning is unlikely to be effective and the traversal may end up exploring an exponential number of nodes. Instead, we consider a reduced subspace that has shorter paths and satisfies the monotonicity property for *every type of edge*.

5.2 A Reduced Search Space

In this section, we define a reduced space $\mathcal{G}_{\subseteq}$, which is a subspace of $\mathcal{G}$. Our reduced space leverages the fact that traffic is skewed, and thus the heavy part of the traffic shares many feature values. This means that specifications consisting of these common feature values have higher score than other specifications and that these higher-scored specifications intersect. This motivates us to focus only on solutions whose specifications are contained in one another. Such an approach guarantees that the solutions balance path coverage (provided by the shorter specifications) and explainability (provided by the larger specifications). We show that $\mathcal{G}_{\subseteq}$ contains solutions which are not significantly worse than an optimal solution in $\mathcal{G}$. Specifically, we show that $\mathcal{G}_{\subseteq}$ contains a solution whose score is at least $min\{\frac{1}{k},\frac{1}{t}\}$ of an optimal solution in $\mathcal{G}$, in the worst case. In $\mathcal{G}_{\subseteq}$, the size of the search paths is t (instead of $t \cdot k$ as in $\mathcal{G}$), and every node has at most $\sum_{i=1}^{l} |U_i|$ children (instead of $k \cdot \sum_{i=1}^{l} |U_i|$).

The nodes of $\mathcal{G}_{\subseteq}$ are all specification sets whose path specifications are *extensions of one another*. More formally, a node has the property that its (nonempty) specifications can be ordered to $S_1, ..., S_m$ such that *(i)* $S_1 \subset ... \subset S_m$ and *(ii)* for all $1 \leq i \leq m$, $|S_i| = i$. For example, $\{\{\text{New York}_e\}, \{\text{New York}_e, 1_{sp}\}\}$ is a node in $\mathcal{G}_{\subseteq}$, while $\{\{\text{New York}_e\}, \{\text{Washington}_e\}\}$ is not.

The edges of $\mathcal{G}_{\subseteq}$ combine both kinds of edges of $\mathcal{G}$. Concretely, there is an edge (u,v) if v contains *all* specifications of u and *also* contains a specification that extends the largest specification of u with an additional feature value. More formally, if the (nonempty) specifications of u are ordered as defined before to $S_1, ..., S_m$, then v has the specifications $S_1, ..., S_m, S_{m+1}$ such that $S_m \subset S_{m+1}$ and $|S_{m+1}| = m+1$. Fig. 2 highlights the nodes of $\mathcal{G}_{\subseteq}$ with a green background and shows the edges of $\mathcal{G}_{\subseteq}$ (which are different from the edges of $\mathcal{G}$) in green.

Optimality We now discuss how solution optimality in $\mathcal{G}_{\subseteq}$ relates to that in $\mathcal{G}$. Intuitively, there are two "worst case scenarios". First, if specifications are of size t, a solution of $\mathcal{G}_{\subseteq}$ that contains any such specification contains subsets of this specification as well, which "take the spot" of the other specifications, *without necessarily contributing to the score*. To illustrate this, consider the scenario where $k = 3$, $t = 4$ and there are 3 paths, p_1, p_2, p_3 with weight 1 whose feature values are $\{e_1, i_1, o_1, sp_1\}$, $\{e_2, i_2, o_2, sp_2\}$,$\{e_3, i_3, o_3, sp_3\}$, respectively (where e_n is an egress, i_n is an ingress, o_n is an organization, and sp_n is an indicator for shortest path). An optimal solution is to pick exactly these three feature values resulting in a score of 12. However, in $\mathcal{G}_{\subseteq}$, a solution that includes one of these specifications contains also its subsets, making the score of the optimal solution only 3. The other extreme is if all optimal solutions are of size 1. In this case, sets of size greater than 1 may add little gain to the score. To illustrate this, consider the scenario where $k = 3, t = 4$ and there are 12 paths, $p_1, \ldots, p_{12}$ with weight 1 such that $p_1, \ldots, p_4$ have property e_1, $p_5, \ldots, p_8$ have property e_2 and $p_9, \ldots, p_{12}$ have property e_3 (besides this, there are no common feature values). An optimal solution is $\{e_1\}$, $\{e_2\}$,$\{e_3\}$ whose score is 12. However, because of the structure of our space, the optimal solution has score 6.

The next lemma states that the maximum gap between the scores of the optimal solution in $\mathcal{G}_{\subseteq}$ and $\mathcal{G}$ is at most a factor of $min\{\frac{1}{t},\frac{1}{k}\}$. Proof is provided in Appendix B.

Lemma 1 Let $OPT_{\mathcal{G}}, OPT_{\mathcal{G}_{\subseteq}}$ be the optimal solutions in $\mathcal{G}$ and $\mathcal{G}_{\subseteq}$. Then, $\min\{\frac{1}{t},\frac{1}{k}\} \cdot \Phi(OPT_{\mathcal{G}}) \leq \Phi(OPT_{\mathcal{G}_{\subseteq}})$.

5.3 A Path Equivalent Space

In this section, we define a search space which is similar to $\mathcal{G}_{\subseteq}$ but may contain solutions with higher score. Intuitively, this is obtained by "merging" nodes in $\mathcal{G}_{\subseteq}$ that are equivalent with respect to the satisfying paths. In other words, for every two nodes in this space, there is at least one path satisfying one but not the other. Path equivalence does not imply the same score. For example, if $\{e_1\}, \{i_1\}$ are path equivalent, then $\{e_1, i_1\}$ is also path equivalent to them, but with a score twice as high from each (because each path contributes its weight twice, once per feature). By considering only nodes that are not path equivalent, we can potentially obtain better solutions, without sacrificing the lower bound of Lemma 1.

We use this observation to modify $\mathcal{G}$ to a space $\mathcal{G}_=$ whose solutions consist of specifications that are *(i)* contained in one another (like $\mathcal{G}_\subseteq$) and *(ii)* maximal with respect to path equivalence. In our example, this means that $\{e_1\},\{i_1\}$ are not part of any solution in $\mathcal{G}_=$, but $\{e_1,i_1\}$ might be if its extensions are not equivalent to it. In $\mathcal{G}_=$, there is an edge (u,v) if, for u whose specifications are $S_1 \subseteq ... \subseteq S_m$, we have *(i)* the specifications of v are $S_1,...,S_m,S_{m+1}$, *(ii)* $S_m \subset S_{m+1}$, and *(iii)* for any subset S such that $S_m \subset S \subset S_{m+1}$, S_{m+1} and S are path equivalent. By construction, $\mathcal{G}_=$ has solutions which are at least as good as those in $\mathcal{G}_\subseteq$, which gives us:

Lemma 2 Let $O_{\mathcal{G}_\subseteq}$ and $O_{\mathcal{G}_=}$ be optimal solutions in $\mathcal{G}_\subseteq$ and $\mathcal{G}_=$, respectively. Then, $\Phi(O_{\mathcal{G}_=}) \geq \Phi(O_{\mathcal{G}_\subseteq})$.

By traversing $\mathcal{G}_=$, algorithms can return solutions with larger path specifications than if they traversed $\mathcal{G}_\subseteq$. This follows since the maximal size of a specification in $\mathcal{G}_\subseteq$ is k, while the size of specifications in $\mathcal{G}_=$ is up to t.

6 The ComPass Algorithm

We now introduce ComPass, our algorithm for **com**puting **pa**th **s**pecification**s** by traversing the search space $\mathcal{G}_=$. ComPass (Algorithm 1) lazily computes nodes in $\mathcal{G}_=$ and continues to the node with the highest increase in score. It takes as input a set of routing paths $\mathcal{R}$, a set of feature functions $q_1,\ldots,q_l$, and constants k and t denoting the maximal number of specifications and the maximal size of each path specification. ComPass starts by initializing the set of solutions $\mathcal{S}$ and the current specification L to the empty set and $\mathcal{Q}$ to the set of all candidate feature functions (Lines 1–3). In up to k iterations, the best feature value is selected to extend L according to the score function – namely, the feature value that will maximize the score of $\mathcal{S}$ as defined by the score function (Eq. (1)) when adding it to L . This can be formalized as maximizing the function on Line 5.

Let v be this feature value and q its feature. Then, L is extended with v and q is dropped as L cannot contain another feature value from U_q. The paths in $\mathcal{R}$ not meeting v are dropped as well, as these will not be described by the next specifications (Lines 6–8). Then, if the size of L reaches the bound t, the loop breaks as it is impossible to extend L further (Line 9). Otherwise, ComPass computes the maximal specification that is equivalent to L by checking whether it can be extended with other feature values (Lines 10–13). Finally, L is added to $\mathcal{S}$ (Line 14), and the next iteration begins. To ensure the limit of t is not exceeded, once L has reached this bound, ComPass completes and returns the current specification sets. This means that ComPass may return fewer than k specifications. It can be shown that this solution has a higher score than a solution with k specifications that are not representative of their class. Intuitively, this follows since the paths described by the descendants are subsumed by the paths described by their ancestors.

Algorithm 1: ComPass ($\mathcal{R}$, $q_1,\ldots,q_l$, k, t)

Input : $\mathcal{R}$: a set of routing paths.
$q_1,\ldots,q_l$: a set of feature functions.
k: limit on the number of specifications.
t: limit on the size of specifications.
Output: A set of specifications $\mathcal{S}$.

1 $\mathcal{S} = \emptyset$ // The specification set
2 $L = \emptyset$ // The last computed specification
3 $\mathcal{Q} = \{q_1,...,q_l\}$ // Candidate features
4 **while** $|\mathcal{S}| < k$ **do**
5 $\quad q, v = \arg\max_{q\in\mathcal{Q}, v_q\in\mathcal{U}_q} \sum_{(d,P)\in\mathcal{R}} w_{d,P}\cdot[q(d,P)=v_q]$
6 $\quad L = L\cup\{v\}$
7 $\quad \mathcal{Q} = \mathcal{Q}\setminus\{q\}$
8 $\quad \mathcal{R} = \mathcal{R}\setminus\{(d,P) \mid q(d,P)\neq v\}$
9 $\quad$ **if** $|L| = t$ **then** $\mathcal{S} = \mathcal{S}\cup\{L\}$; **break**
10 $\quad$ **while** $\exists v\in\mathcal{U}_{\mathcal{Q}}.(L\cup\{v\}\equiv L)$ **do**
11 $\quad\quad L = L\cup\{v\}$
12 $\quad\quad$ **if** $|L| = t$ **then** $\mathcal{S} = \mathcal{S}\cup\{L\}$; **break**
13 $\quad\quad \mathcal{Q} = \mathcal{Q}\setminus\{q\}$
14 $\quad \mathcal{S} = \mathcal{S}\cup\{L\}$
15 **return** $\mathcal{S}$

Example 4 Consider $\mathcal{R} = \{(\text{Google}, P_i)\}_{i=1}^{100}$, each with weight 1, and $k = t = 2$. As before, we assume that *(i)* if $i \leq 60$, $E(\text{Google}, P_i) = \text{NY}_e$, and $E(\text{Google}, P_i) = \text{LA}_e$ otherwise, *(ii)* for $i \leq 40$, $SP(\text{Google}, P_i) = 1_{sp}$, and *(iii)* all other feature values are unique for every path. We now show how ComPass computes the optimal solution $\{\text{NY}_e\},\{\text{NY}_e, 1_{sp}\}$. In its first iteration, ComPass discovers that the feature value NY_e maximizes the score. It thus extends L to $\{\text{NY}_e\}$, prunes the egress feature E from $\mathcal{Q}$, and removes from $\mathcal{R}$ all paths whose egress is not NY. Since $\{\text{NY}_e\}$ is the representative of its equivalence class, it is added to $\mathcal{S}$. In the second iteration, the feature value 1_{sp} maximizes the score. Hence, ComPass extends L with 1_{sp}. Since the limit $t = 2$ has been reached, the loop breaks (Line 7), and $\{\text{NY}_e\},\{\text{NY}_e, 1_{sp}\}$ is returned.

Finding the best feature value To avoid iterating every feature value separately in Line 5 (which can incur high overhead), we find the best feature value by iterating over the feature functions in $\mathcal{Q}$ and the routing paths in $\mathcal{R}$ and storing the score of each feature value in a hash table. Then, with a single pass over the hash table, we find the feature value with the highest score.

Guarantees Our next theorem states that ComPass computes a solution whose score is at least $\frac{1-f}{1-f^{\min\{t,k\}}}$ of the optimal solution in $\mathcal{G}_=$, where $f \in (0,1)$ is the maximal portion of paths that a child of a node can have. Note that

since ComPass explores $\mathcal{G}_=$, whose nodes are not path equivalent, f cannot be 1. Proof is in Appendix B.

Theorem 1 Given that there is $f \in (0,1)$ such that for every pair of path specifications A, A' if $A \subset A'$, then $\Phi(\{A\}) \leq f \cdot \Phi(\{A'\})$. Then, if O is the solution returned by ComPass, we have $\frac{1-f}{1-f^{\min\{t,k\}}} \cdot OPT_{\mathcal{G}_=} \leq O$.

Example 5 The factor f is determined by the pair of nodes $A \subset A'$ whose scores are the closest. In the previous example, $A = \{\{\text{NY}_e\}, \emptyset\}$, $A' = \{\{\text{NY}_e\}, \{\text{NY}_e, 1_{sp}\}\}$. Since $\Phi(A) = 60$ and $\Phi(A') = 100$, we get that $f = 0.6$. By the theorem, ComPass returns a solution whose score is at least 62.5% compared to the optimal solution in $\mathcal{G}$.

Speeding up ComPass by sampling To compute the best feature value, ComPass iterates in Line 5 over all routing paths to determine the best feature value. This step is very expensive, especially if the number of routing paths and feature functions is large. To mitigate this problem, we leverage two observations that allow ComPass to uniformly sample the routing paths instead of considering all routing paths. First, Internet traffic is heavily skewed, which means that most traffic is directed towards a few organizations (e.g., CDNs), and egresses see different traffic volumes depending on the peering. This means that sampling is likely to pick representative routing paths. Second, by the score function definition, optimal solutions consist of specifications describing the main part of the traffic. This means that specifications representing little traffic have little effect on the decisions ComPass makes. This implies that sampling will perform well as it is more likely to ignore the specifications with few routing paths than the ones with many.

7 From Specifications to Summaries

In this section, we describe how *Net2Text* produces a natural language summary given a specification set $\mathcal{S}$ (generated by ComPass). It begins by augmenting $\mathcal{S}$ with additional information in three steps. It then transforms the path specifications in $\mathcal{S}$ to natural language sentences using templates. In the first two steps, *Net2Text* augments $\mathcal{S}$ with information computed as a byproduct by ComPass (i.e., additional specification sets and the amount of traffic). In the third step, *Net2Text* extends $\mathcal{S}$ with high-level features, which cannot be directly computed by ComPass. We next describe these steps and exemplify them on our running example, $\mathcal{S} = \{\{\text{NY}_e\}, \{\text{NY}_e, 1_{sp}\}\}$.

Step 1: Adding path specifications *Net2Text* extends every $S \in \mathcal{S}$ with the next m (a parameter) best path specifications that have the same parent in $G_=$ and are values of the same feature function. These path specifications can be extracted from the computation of ComPass (in Line 5). In our example, for $m = 1$, this step results in adding $\{\text{Washington}_e\}$ to $\mathcal{S}$ as NY_e and Washington_e have the same parent and same feature function (egress). This will eventually be translated to a single sentence: *Google traffic exits in New York and Washington*.

Step 2: Adding traffic size Then, *Net2Text* extends every $S \in \mathcal{S}$ with the total weight of the paths it describes to let the operator understand how much traffic the summary covers. In our example, this gives $\{(60\%, \{\text{NY}_e\}), (40\%, \{\text{Washington}_e\}), (39.6\%, \{\text{NY}_e, 1_{sp}\})\}$.

Step 3: Computing high-level features Next, we extend $\mathcal{S}$ with high-level features (e.g., load-balancing, waypointing, or hot-potato routing) that are not properties of single paths but rather of *sets of paths*, i.e., entire specifications. Thus, these features can only be identified after ComPass computed the best specification set. Each high-level feature defines the criteria that a specification has to meet for it to hold. For example, for load-balancing, the ingress and egress of a specification have to be fixed and the paths described by it need to be disjoint. In our example, *Net2Text* inferred that a common waypoint for $(39.6\%, \{\text{New York}_e, 1_{sp}\})$ is Atlanta as all the paths in this specification go through Atlanta, and thus this specification is extended to $(39.6\%, \{\text{New York}_e, 1_{sp}, \text{Atlanta}_w\})$. In addition, *Net2Text* inferred that the traffic to Google experiences hot-potato routing as it has multiple egresses and all the traffic is forwarded to the closest one.

Step 4: Translation to natural language Lastly, $\mathcal{S}$ is translated to natural language sentences using templates. The sentences are a composition of multiple basic templates. To create fluency in the summary, *Net2Text* connects related sentences by building upon the previous sentence. In addition, it does not repeat information. For example, the second sentence in our example summary in Section 2 does not repeat that it refers to Google traffic, and the percentage shown is relative (i.e., $39.6\%/60\% = 66\%$). Namely, $\{(39.6\%, \text{NY}_e, 1_{sp}, \text{Atlanta}_w)\}$ is mapped to: *66% of the traffic exiting in New York follows the shortest path and crosses Atlanta*.

8 Parsing Queries

To leverage *Net2Text*'s summarization capabilities, the operator needs to provide the feature functions Q, t, k, and the routing paths $\mathcal{R}$. Typically, once Q, t and k are specified, the operator queries the network database to obtain $\mathcal{R}$. To simplify this, *Net2Text* allows the operator to submit queries in natural language which it then translates to SQL-like queries for the network database. In the following, we describe how *Net2Text* parses these queries expressed in natural language.

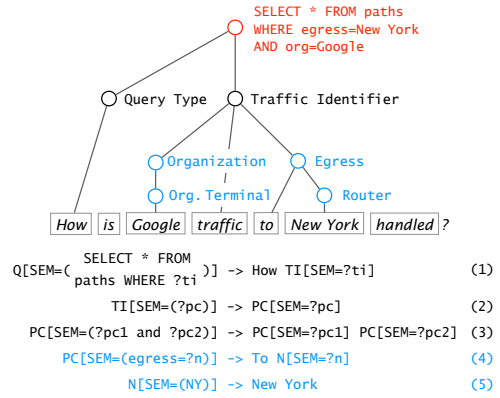


Figure 3: A parse tree and rules in the network grammar.

Network grammar The grammar consists of rules specifying how constituents of the queries (e.g., clauses, words) can be composed. The rules also specify the semantics of the constituents (e.g., the network terms such as "ingress"), which enables the parser to construct the SQL-like query. The grammar consists of two parts: *(i)* a structural part ($\sim$ 70 rules), which defines the allowed constituent compositions; and *(ii)* a domain-specific part consisting of mapping rules ($\sim$ 80 rules), which capture the network specific features (e.g., egress, organization) as well as keywords (e.g., router and location names). This split enables the operator to easily extend the grammar with new features and keywords without having to deal with the structure of the queries.

Structural grammar This grammar defines the query structure and its building blocks. We identify two main building blocks: query type and traffic identifier. Depending on the query type, there may be additional building blocks. There are four query types: yes/no ("Is/Does..."), counting ("How many..."), data retrieval ("What is/are..."), and explanation ("How is/does traffic..."). The query type determines whether the answer is yes/no, a count, a list, or a summary (obtained using ComPass). The traffic identifier defines the WHERE clause of the SQL-like query. The attributes selected by the query are either determined by the query (in data retrieval queries) or are simply a wildcard (i.e., $\star$).

Fig. 3 illustrates the structural parsing. *"How"* defines the desired behavior of *Net2Text* (summarize the data with ComPass), while "Google traffic to New York" is the traffic identifier. The black rules (1-3) are part of the structural grammar, while the blue rules (4 & 5) are part of the domain-specific grammar, which we discuss next.

Domain-specific grammar This grammar defines a mapping between keywords and names to features and their values. For example, the grammar defines the rules *(i)* "to $N \rightarrow$ egress=N" indicating that the natural language phrase "to N" means that N is a name of an egress, where N is a non-terminal and *(ii)* $N \rightarrow \text{NY}, \text{LA}, ...$, lists the possible egress names. Using these rules, "to NY" is parsed to `egress=NY` in the SQL-like query.

9 Evaluation

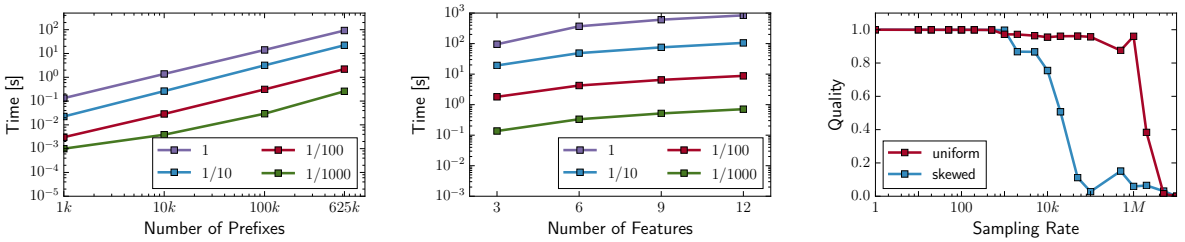
We evaluated *Net2Text*'s scalability and usability. For scalability, we show that *Net2Text* can summarize large forwarding state (§9.2) and generate summaries of high quality, even with sampling (§9.3). Worst-case queries complete within 2 seconds in large networks ($\sim$ 200 nodes). For usability, we show that *Net2Text* is useful for operators by conducting interviews (§9.4) and showcase its end-to-end implementation in a case study (§9.5).

9.1 Methodology

We run our Python-based prototype ($\sim 3k$ lines of code) on a machine with 24 cores at 2.3 GHz and 256 GB of RAM. For the experiments, we implemented an ISP-like forwarding state generator, which we use to produce realistic forwarding state for various Topology Zoo [15] topologies ranging from 25 to 197 nodes (Table 2). The generator enables us to control how "summarizable" a state is by varying how skewed it is.

Forwarding state generation Our generator synthesizes network-wide forwarding states (i.e., the set of routing paths $\mathcal{R}$) for a given number of IP prefixes and a given network topology in five consecutive steps. *First*, it randomly chooses a set of egress nodes (see Table 2). *Second*, it creates a prefix-to-organization mapping using the CAIDA AS-to-organization dataset [16]) and a full IPv4 RIB [17]. *Third*, for each organization, it chooses the number of egresses using an exponential distribution fitted according to real measurements [18, Fig.3.], after which the actual egresses are uniformly chosen from the set of egress nodes. *Fourth*, for each node, it computes its forwarding state by picking for each prefix the closest egress. *Fifth*, each routing path (d, P) is finally associated with an amount of traffic sampled from an exponential distribution. This leads to few organizations owning many prefixes, carrying relatively more traffic than others (as shown in [12]). The generator can also generate extra features whose values are arbitrarily picked.

Generality While we generate the input forwarding state, we stress that our results are representative because: *(i)* the scalability of ComPass does *not* depend on the actual feature values but only on the number of features (see §6); and *(ii)* the quality analysis does not depend on the actual score but rather on the ratio compared to other scores under the same setting.



(a) Time as a function of the number of prefixes and sampling rate.

(b) Time as a function of the number of features and sampling rate.

(c) Summary quality as a function of sampling rate.

Figure 4: *Net2Text* scales—both in the size of the forwarding state (sub-second, even for 625k prefixes) *(a)* and the number of features (sub-second, even for 12 features) *(b)*—while sampling affects the summary quality marginally *(c)*.

Topology	Nodes	Egresses	No sampling	1/1000
ATT NA	25	10	94.07 s	**0.26 s**
Switch	42	15	128.12 s	**0.24 s**
Sinet	74	30	223.91 s	**0.50 s**
GTS CE	149	40	611.81 s	**1.18 s**
Cogent	197	50	766.61 s	**1.84 s**

Table 2: With sampling (§6), *Net2Text* summarizes large network forwarding states (> 600k prefixes), *within 2 seconds*, for networks with close to 200 nodes.

9.2 Scalability Analysis

We evaluate *Net2Text* scalability by measuring the time it takes to summarize all routing paths (worst-case) while varying the number of key dimensions: prefixes, nodes, and feature functions. To evaluate the sampling optimization of ComPass, we run ComPass four times: without path sampling and with sampling rate of $1/10, 1/100, 1/1000$. We repeated each experiment 10 times and report median results (std dev is small).

Fig. 4a shows the results when varying the number of prefixes from 10^3 to 10^5 and the full RIB for the ATT NA topology using 3 feature functions. The results indicate that *Net2Text* scales linearly in the number of prefixes. The running time decreases proportionally to the sampling rate. Without sampling, summarizing forwarding states with 625k prefixes takes about 100 seconds and *less than one second* with a sampling rate of 1/1000. Fig. 4b shows a similar trend when varying the number of features from 3 to 12 and using a full RIB.

Table 2 shows the results when considering different topology sizes, with full routing tables (625k prefixes) and 3 feature functions. The table also reports results with (rate of $1/1000$) and without sampling. We see that the runtime is roughly linear in the number of nodes in the network. More importantly, our results indicate that *Net2Text* scales to large networks with hundreds of nodes thanks to sampling: it takes less than 2 seconds for *Net2Text* to summarize Cogent forwarding state.

9.3 Quality Analysis

We now evaluate the effect of sampling the input data (i.e. the forwarding paths) and show that doing so only marginally impacts the quality of the summary. In addition, we show that ComPass compares well against two baselines both in terms of quality and running time.

We measure the quality of a summary using the score function presented in §4. Intuitively, the score represents the traffic volume of the paths covered by the resulting summary, rewarding more detailed summaries by multiplying the volume of each path by the number of details (feature values) present in the summary. When computing the score, we always account for all entries that match the resulting summary and not just for the sampled entries. As in §9.2, we consider the problem of summarizing every single entry in the network database.

For the experiment, we generate forwarding state for the ATT NA topology with a full routing table and vary the sampling rate from 1 to 1/5,000,000. Note that we have more entries in the network database than the total number of prefixes as there is at least one path from every node to every prefix. Hence, even with sampling rates higher than the number of prefixes, we still have paths to summarize. For this setup, we have more than 15 million entries in the network database.

Fig. 4c shows the score of the summary for different sampling rates normalized to the score without sampling. We ran the experiment for two different scenarios: *(i)* highly skewed traffic distributions among the feature values, where the size difference between the feature values is high; and *(ii)* uniform distributions, where the difference between them is low. Our results show that the sampling rate at which the score of the summary drops significantly is very high. Even with sampling rates of 1/1000, ComPass still creates summaries whose qualities are within 5% of the unsampled summary.

To further illustrate the quality of ComPass summaries, we compare it against two baselines. Both iterate once over the relevant routing paths

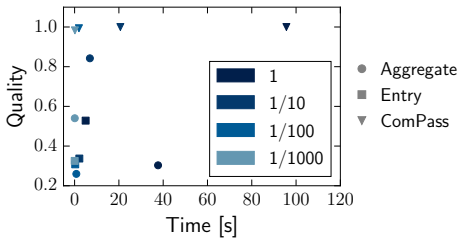


Figure 5: *Net2Text* produces summaries of higher quality than two simple baselines.

and pick the most detailed specification (e.g., $\{\text{New York}_e, \text{Philadelphia}_i, \text{Google}_o\}$). From this specification, we build the full specification set by randomly removing one feature value after the other to obtain for example $\{\{\text{New York}_e\}, \{\text{New York}_e, \text{Google}_o\}, \{\text{New York}_e, \text{Philadelphia}_i, \text{Google}_o\}\}$. The baselines differ in how they choose the most detailed specification: *Entry*, takes the routing path with the highest weight and uses it as the most detailed specification; and *Aggregate*, aggregates all routing paths with the same feature values and uses the largest aggregate. When computing the quality of the summaries, we consider all routing paths matching the resulting summary. Thanks to sampling, ComPass produces *higher* quality summaries in the same amount of time as the two baselines (see Fig. 5). If we also consider the information added to the summary by extending it as described in 7, we see that ComPass outperforms the baselines by almost 5 times.

9.4 Usefulness

To better assess the usability of *Net2Text*, we conducted five interviews with network operators of ISP networks (research, Tier 1 and Tier 2) and one enterprise network. We questioned them about four aspects.

Aspect 1: Need for virtual assistants All operators see opportunities for virtual assistants in tasks requiring to process a lot of data to identify and extract the relevant information. An assistant allows them to focus on remedying, rather than identifying and analyzing the event.

Aspect 2: Relevance of the NL input The possibility to write queries in natural language was well-perceived. Some operators, however, do not mind a fixed query language or writing their own scripts.

Aspect 3: Relevance of the NL output Most operators see value in natural language summaries as they are concise and simple to understand, especially for less technical persons. Depending on the query, some operators mentioned that they would like to see visualizations of the summary (e.g., a graph) in addition to text.

Aspect 4: Usefulness of *Net2Text* queries All operators confirmed that the queries currently supported by *Net2Text* are relevant. In particular, they appreciated the ability to query about incoming traffic. In addition, most operators testified interest in service-oriented queries, instead of purely destination-oriented ones (e.g., traffic to the Gmail-service instead of Google traffic in general).

In the discussions, we saw a clear difference between the queries of ISP and enterprise network operators. While the ISP operators were mostly concerned about where traffic was entering and leaving the network, the enterprise operator was more interested in the status of the different applications running in the network and their policies (e.g., is there always a firewall on the path).

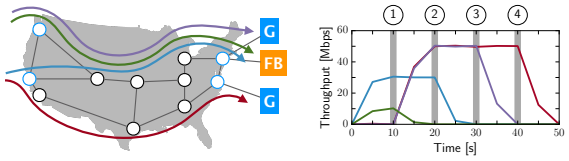
9.5 Case Study

We showcase our end-to-end implementation of *Net2Text* by running it in a Quagga-based network emulating Internet2 (Fig. 6a). Routers in Seattle, Sunnyvale, New York and Washington are connected to external peers. The router in New York receives routes to both Google and Facebook, the router in Washington only to Google. All external routes have the same local-preference. We generate transit traffic entering via Seattle and Sunnyvale towards both destinations. The flows are highlighted in Fig. 6a and the measured throughput is depicted in Fig. 6b. Every ten seconds, *Net2Text* summarizes the entire forwarding state as indicated by the grey bars.

Table 3 shows the 4 summaries produced by *Net2Text*. We see that *Net2Text* is able to explain the current forwarding behavior at different levels of detail and automatically zoom in on the largest part of the traffic. At the time of the second summary, for example, traffic for Google has spiked (purple and red) and is now three times larger than Facebook. We see that *Net2Text* automatically focuses on the traffic to Google and provides more details about it, yet it still mentions traffic to Facebook. In the third summary, we see how *Net2Text* captures higher-level constructs that are not directly present in the database such as “hot-potato routing” (§7).

10 Discussion

Why natural language? We believe that a chat-like interface provides a familiar and intuitive way for operators to interact with their network. That said, our summarization contribution is useful in its own right, independently of the NLP interface. As an illustration, we could easily translate *Net2Text* summaries to a graph-based representation (e.g. using PGA [11]) rather than natural language.



(a) Internet 2 Topology (b) Network Throughput

Figure 6: We ran our *Net2Text* implementation in a *live* network emulating Internet 2 (a) and vary the network throughput according to (b).

1	*"Traffic has a single egress (New York), and goes to a single destination (Facebook). It enters at the following ingresses: Sunnyvale (76%) and Seattle (24%)."*
2	*"Traffic goes to the following destinations: Google and Facebook. Traffic for Google exits through Washington (50%) and New York (50%)."*
3	*"Traffic is destined to Google. It experiences hot-potato routing. It exits through the following egresses: New York (50%) and Washington (50%)."*
4	*"Traffic leaves through Washington, has a single ingress (Sunnyvale), and goes to Google."*

Table 3: Actual summaries produced by our *Net2Text* implementation when run on the network depicted in Fig. 6.

What about new feature functions? While we only deal with a limited set of features in this paper, we stress that ComPass is flexible and can deal with *any* features defined over paths. Additional features (e.g. such as the TCP port number) can be easily added by adding a new field to the database. For the translation, the singular and plural of the feature name also have to be added to the rules. The operator can also add a mapping of feature values to some string, e.g., TCP port 80 to HTTP.

What about the network database? We assume that the network database is fed with high-quality *and* consistent data and focus on the problem of summarizing it. This is a strong assumption. Gathering high-quality state consistently is challenging and the quality of our summaries will inevitably suffer should the data be incomplete, outdated or inconsistent. Fortunately, multiple works have looked at the problem of extracting network data in a fast and consistent manner, which *Net2Text* can directly leverage. In particular, Libra [19] tackles the problem of capturing consistent snapshots of the network forwarding state. Similarly, FlowRadar [20] and Stroboscope [21,22] tackle the problem of quickly gathering fine-grained traffic statistics. Yet, summarizing network-wide behavior in the presence of incorrect or inconsistent data is an interesting problem we plan to address in future work.

11 Related work

Network verification & testing *Net2Text* directly complements previous initiatives on data-plane [5–10] and control-plane verification [1–4] as it does not aim at verifying, but *explaining* network-wide behavior. The reason is that even perfectly correct networks might exhibit unwanted or suboptimal behaviors at runtime, for instance, due to unforeseen traffic shifts or partial failures.

Network provenance *Net2Text*'s high-level objectives of *explaining how networks behave* bear similarities with many works on Network Provenance (e.g., [23–29]). The main difference between these works and *Net2Text* is that *Net2Text* does not aim at explaining *why* a particular state is observed (by following the derivation history), but rather summarizing *what* is the current state being observed to make it understandable to human operators. *Net2Text* can therefore be seen as complementary to these frameworks. Once the network operator understands what is the network behavior, he or she can then ask questions about why. We also believe that *Net2Text*'s summarizing capabilities can be applied to summarize provenance explanations which often tend to be large.

Connecting natural languages and networks A prior work [30] introduced NLP techniques to network management. It proposes to use natural language as interface between operators and an SDN network. Unlike *Net2Text*, it does not provide any abstraction capability and is limited to simple yes/no questions/answers along with simple control tasks such as rate limiting a flow.

12 Conclusions

We presented *Net2Text*, a novel approach to assist network operators in reasoning about network forwarding behaviors. *Net2Text* is based on efficient summarization techniques which generate interpretable summaries (in natural language) out of low-level forwarding rules. We propose an efficient approximation algorithm (with provable bounds) to solve the summarization problem. We fully implemented *Net2Text* and showed that it is highly effective—it only takes 2 seconds to summarize the state of hundreds of routers carrying full routing tables.

Acknowledgements

We are grateful to our shepherd Ranjita Bhagwan, the anonymous reviewers, Roland Meier and Dimitar Dimitrov for the constructive feedback and comments. We are also grateful to all the network operators who provided feedback and insights about *Net2Text*.

References

[1] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd D Millstein. A General Approach to Network Configuration Analysis. In *USENIX NSDI*, Oakland, CA, USA, 2015.

[2] Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. Fast Control Plane Analysis Using an Abstract Representation. In *ACM SIGCOMM*, Florianópolis, Brasil, 2016.

[3] Konstantin Weitz, Doug Woos, Emina Torlak, Michael D. Ernst, Arvind Krishnamurthy, and Zachary Tatlock. Scalable Verification of Border Gateway Protocol Configurations with an SMT Solver. In *ACM OOPSLA*, Amsterdam, Netherlands, 2016.

[4] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. A General Approach to Network Configuration Verification. In *ACM SIGCOMM*, Los Angeles, CA, USA, 2017.

[5] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. Debugging the Data Plane with Anteater. In *ACM SIGCOMM*, Toronto, Canada, 2011.

[6] Peyman Kazemian, George Varghese, and Nick McKeown. Header Space Analysis: Static Checking for Networks. In *USENIX NSDI*, San Jose, CA, USA, 2012.

[7] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. Real Time Network Policy Checking using Header Space Analysis. In *USENIX NSDI*, Lombard, IL, USA, 2013.

[8] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. VeriFlow: Verifying Network-Wide Invariants in Real Time. In *USENIX NSDI*, Lombard, IL, USA, 2013.

[9] Nuno P Lopes, Nikolaj Bjørner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. Checking Beliefs in Dynamic Networks. In *USENIX NSDI*, Oakland, CA, USA, 2015.

[10] Radu Stoenescu, Matei Popovici, Lorina Negreanu, and Costin Raiciu. SymNet: Scalable Symbolic Execution for Modern Networks. In *ACM SIGCOMM*, Florianópolis, Brasil, 2016.

[11] Chaithan Prakash, Jeongkeun Lee, Yoshio Turner, Joon-Myung Kang, Aditya Akella, Sujata Banerjee, Charles Clark, Yadi Ma, Puneet Sharma, and Ying Zhang. PGA: Using Graphs to Express and Automatically Reconcile Network Policies. In *ACM SIGCOMM*, London, United Kingdom, 2015.

[12] Jennifer Rexford, Jia Wang, Zhen Xiao, and Yin Zhang. BGP Routing Stability of Popular Destinations. In *ACM IMC*, Marseille, France, 2002.

[13] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[14] Henry Wadsworth Gould. *Combinatorial Identities: A Standardized Set of Tables Listing 500 Binomial Coefficient Summations*. Morgantown, 1972.

[15] S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The Internet Topology Zoo. *IEEE JSAC*, October 2011.

[16] CAIDA. AS Organizations Dataset, 2017-04-01. `http://www.caida.org/data/as-organizations`.

[17] CAIDA. BGPStream. `https://bgpstream.caida.org/`.

[18] Jaeyoung Choi, Jong Han Park, Pei-chun Cheng, Dorian Kim, and Lixia Zhang. Understanding BGP Next-Hop Diversity. In *IEEE Global Internet Symposium*, Shanghai, China, 2011.

[19] Hongyi Zeng, Shidong Zhang, Fei Ye, Vimalkumar Jeyakumar, Mickey Ju, Junda Liu, Nick McKeown, and Amin Vahdat. Libra: Divide and Conquer to Verify Forwarding Tables in Huge Networks. In *USENIX NSDI*, Seattle, WA, USA, 2014.

[20] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. FlowRadar: A Better NetFlow for Data Centers. In *USENIX NSDI*, Santa Clara, CA, USA, 2016.

[21] Olivier Tilmans, Tobias Bühler, Ingmar Poese, Stefano Vissicchio, and Laurent Vanbever. Stroboscope: Declarative Network Monitoring on a Budget. In *USENIX NSDI*, Renton, WA, USA, 2018.

[22] Olivier Tilmans, Tobias Bühler, Stefano Vissicchio, and Laurent Vanbever. Mille-Feuille: Putting ISP Traffic under the scalpel. In *ACM Hotnets*, Atlanta, GA, USA, 2016.

[23] Ang Chen, Yang Wu, Andreas Haeberlen, Wenchao Zhou, and Boon Thau Loo. The Good, the Bad, and the Differences: Better Network Diagnostics with Differential Provenance. In *ACM SIGCOMM*, Florianópolis, Brasil, 2016.

[24] Yang Wu, Mingchen Zhao, Andreas Haeberlen, Wenchao Zhou, and Boon Thau Loo. Diagnosing Missing Events in Distributed Systems with Negative Provenance. In *ACM SIGCOMMM*, Chicago, IL, USA, 2014.

[25] Wenchao Zhou, Suyog Mapara, Yiqing Ren, Yang Li, Andreas Haeberlen, Zachary Ives, Boon Thau Loo, and Micah Sherr. Distributed Time-aware Provenance. In *VLDB*, Riva del Garda, Trento, Italy, 2013.

[26] Wenchao Zhou, Micah Sherr, Tao Tao, Xiaozhou Li, Boon Thau Loo, and Yun Mao. Efficient Querying and Maintenance of Network Provenance at Internet-Scale. In *ACM SIGMOD*, Indianapolis, IN, USA, 2010.

[27] Andreas Wundsam, Dan Levin, Srini Seetharaman, and Anja Feldmann. OFRewind: Enabling Record and Replay Troubleshooting for Networks. In *USENIX ATC*, Portland, OR, USA, 2011.

[28] Colin Scott, Andreas Wundsam, Barath Raghavan, Aurojit Panda, Andrew Or, Jefferson Lai, Eugene Huang, Zhi Liu, Ahmed El-Hassany, Sam Whitlock, H.B. Acharya, Kyriakos Zarifis, and Scott Shenker. Troubleshooting Blackbox SDN Control Software with Minimal Causal Sequences. In *ACM SIGCOMMM*, Chicago, IL, USA, 2014.

[29] Colin Scott, Vjekoslav Brajkovic, George Necula, Arvind Krishnamurthy, and Scott Shenker. Minimizing Faulty Executions of Distributed Systems. In *USENIX NSDI*, Santa Clara, CA, USA, 2016.

[30] Azzam Alsudais and Eric Keller. Hey Network, Can You Understand Me? In *IEEE INFOCOM Workshop on Software-Driven Flexible and Agile Networking*, Atlanta, GA, USA, 2017.

[31] Gurobi Optimization, Inc. Gurobi Optimizer Reference Manual, 2016.

A ILP Formulation

$$\max \sum_{(d,P)\in\mathcal{R}} 1 \sum_{1\le i\le k} \sum_{v\in U_1\cup\ldots\cup U_l} w_{d,P}\cdot y_{d,P,i,v}$$

$$\sum_{v\in U_j} x_{i,v} \le 1 \quad (1)$$

$$\sum_{v\in U_1\cup\ldots\cup U_l} x_{i,v} \le t \quad (2)$$

$$y_{d,P,i} - y_{d,P,v} + x_{i,v} \le 1 \quad (3)$$

$$y_{d,P,i} + x_{i,v} - y_{d,P,i,v} \le 1 \quad (4.1)$$

$$y_{d,P,i,v} - y_{d,P,i} \le 0 \quad (4.2)$$

$$y_{d,P,i,v} - x_{i,v} \le 0 \quad (4.3)$$

$$\sum_{1\le i\le k} y_{d,P,i} \le 1 \quad (5)$$

$$x_{i+1,v} - x_{i,v} \ge 0 \quad (6)$$

$$y_{d,P,i}, x_{i,v}, y_{d,P,i,v} \in \{0,1\}$$

Figure 7: An integer program for computing a specification set to explain the routing paths. $i \in \{1,\ldots,k\}, j \in \{1,\ldots,l\}, (d,P) \in \mathcal{R}, v \in \{U_1 \cup \ldots \cup U_l\}$

In the following, we show how to formulate the inference problem from Section 4 as an integer linear program (ILP) where the objective encodes the score function Φ and the constraints encode the path specification search space $\mathcal{S}^t_{U_1,\ldots,U_l}$.

Variables We have two kinds of variables: the x-variables which encode the path specification set, and the y-variables which encode the features and specifications that paths meet. For each path specification, we introduce a set of variables, one for every feature value that may be in the path specification. Formally, we have a variable $x_{i,v}$ for every $1 \le i \le k$ and $v \in U_1 \cup \ldots \cup U_l$. These variables are indicator functions and range over $x_{i,v} \in \{0,1\}$. That is, if $x_{i,v} = 1$, it means that v is part of the i^{th} path specification ($v \in S_i$), otherwise v is excluded. Thus, an assignment to the x's uniquely defines a path specification set.

The y variables encode whether paths meet the path specifications and which of their features are described by the specs. Concretely, for every routing path $(d,P) \in \mathcal{R}$, we maintain multiple binary variables:

- $y_{d,P,i}$: encodes whether (d,P) meets the i^{th} specification.
- $y_{d,P,v}$: indicates whether (d,P) contains a feature value of v. Note that the values $y_{d,P,v}$ are known a-priori and need not be computed during optimization.
- $y_{d,P,i,v}$: encodes whether the feature v of P is described by the i^{th} specification.

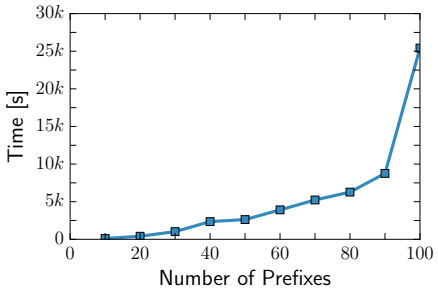


Figure 8: Running time using the ILP (optimal, but slow).

These variables allow us to capture precisely in what detail a path is being described by a specification that it meets. Note that $y_{d,P,i,v}$ can be 1 only if (d,P) meets the i^{th} specification and the i^{th} specification has feature v (i.e., $y_{d,P,i} = x_{i,v} = 1$). This requirement will be encoded as part of the general constraints.

Objective function We encode the objective function of Definition 1 as the weighted sum of $y_{d,P,i,v}$ variables.

Constraints The path specification space is expressed as a set of constraints which states that each path specification can have at most one feature value for the same feature (constraint set (1) in Fig. 7) and at most t features across all features (constraint set (2) in Fig. 7). The next constraint sets encode the score function. Constraint set (3) encodes whether the routing path (d,P) meets the i^{th} specification. Intuitively, the constraints can be presented as $y_{d,P,i} \le 1 + (y_{d,P,v} - x_{i,v})$, which means that $y_{d,P,i}$ can be 1 (to indicate that (d,P) meets the i^{th} specification) only if $y_{d,P,v} \ge x_{i,v}$ for all v, which indicate that the routing path meets all features in the i^{th} specification. Constraint set (4) in Fig. 7 encodes whether the feature value v of a routing path (d,P) is described, which may only be true if (d,P) meets the specification and that the specification contains v. Lastly, the constraint set (5) guarantees that each feature value v met by (d,P) is counted only once. The total number of variables and constraints is $O(k \cdot |\mathcal{R}| \cdot |U_1 \cup \ldots \cup U_l|)$.

As we explain in Section 5, it is useful to impose a certain shape or relation between the path specifications. In particular, we will see why it is useful to require path specifications to be extensions of one another. Constraint set (6) in Fig. 7 encodes this optional requirement.

Scalability To show the need for an efficient algorithm like ComPass, we evaluate the scalability of solving the corresponding ILP (Fig. 7, with all constraints, including (6)) using the `gurobi` solver [31]. Fig. 8 shows the running times for the ATT North America network with

a forwarding state encompassing between 10 to 100 prefixes (up to three orders of magnitude smaller than the experiments in §9.2). Unsurprisingly, the running time quickly explodes due to the large number of variables and constraints. With only *100 prefixes*, the ILP already requires more than 25k seconds to complete.

B Proofs

In this section, we provide the proofs for the lemmas and theorems presented in the paper.

Lemma 1 proof sketch Denote the optimal solution as the specification set: $\{x_1^1,,...,x_{t_1}^1\},\ldots,\{x_1^k,...,x_{t_k}^k\}$. By the score definition and since $t_i \leq t$, $\forall i,j.\frac{j}{t}\Phi(\{x_1^i,,...,x_{t_i}^i\}) \leq \Phi(\{x_1^i,...,x_j^i\})$. W.l.o.g., assume that $\{x_1^1,,...,x_{t_1}^1\}$ has the highest score. Then, $\Phi(\{x_1^1,,...,x_{t_1}^1\}) > OPT/k$. We split to cases. If $k \leq t_1$, then the score of $\{x_1^1\},\{x_1^1,x_2^1\},...,\{x_1^1,...,x_k^1\}$ is at least $k/t \cdot (OPT/k)$. Since $\{\{x_1^1\},...,\{x_1^1,...,x_1^k\}\}$ is a node in $\mathcal{G}_{\subseteq}$, the claim follows. Otherwise, if $t_1 < k$, then $\{\{x_1^1\},...,\{x_1^1,,...,x_{t_1}^1\}\}$ is a node in $\mathcal{G}_{\subseteq}$ and since $\Phi(\{x_1^1,,...,x_{t_1}^1\}) > OPT/k$, the claim follows.

Theorem 1 proof sketch Let the optimal solution be $OPT = \{\{a\},\{a,b\},...,\{a,b,...,m\}\}$ and the specification set that ComPass returned be the specification set $S_{ComPass} = \{\{a'\},\{a',b'\},...,\{a',b',...,m'\}\}$. By the assumption, $\Phi(\{a,b\}) \leq f \cdot \Phi(\{a\}) \leq f \cdot \Phi(\{a'\})$. By induction, $\Phi(\{a,b,...,j\}) \leq f^{|\{a,...,j\}|} \cdot \Phi(\{a'\})$. Since the length of the largest specification in OPT is $\min\{k,t\}$, the length of the optimal solution is at most $\Sigma_{1\leq j\leq \min\{k,t\}} f^j \cdot \Phi(\{a\}) = \frac{1-f^{\min\{t,k\}}}{1-f} \cdot \Phi(\{a\})$. By the greedy operation, we have $\Phi(\{a\}) \leq \Phi(\{a'\})$. Since $\Phi(\{a'\}) \leq \Phi(S_{ComPass})$, we get $\Phi(\{a\}) \leq \Phi(S_{ComPass}) \leq \frac{1-f^{\min\{t,k\}}}{1-f} \cdot \Phi(\{a\})$, which means that ComPass is a $\frac{1-f}{1-f^{\min\{t,k\}}}$-approximation algorithm.