Butterfly: Scalable Probabilistic Verification for Network Resilience under Failures

Saideh Ahangary, Herbert Bos, Klaus v. Gleissenthall, Asia Slowinska
Vrije Universiteit Amsterdam
{s.ahangary, h.j.bos, k.freiherrvongleissenthal}@vu.nl, asia@vusec.net

***Abstract*—Reliable network operation requires reasoning not only about connectivity under failures, but also about performance properties such as congestion when failures occur. Existing probabilistic verification tools, such as NetDice, can efficiently bound the probability that properties hold under failures. However, they assume static routing and do not model how adaptive routing dynamically redistributes traffic in response to failure or overload.**

We present Butterfly, a probabilistic network verification framework that integrates congestion-aware routing into symbolic analysis. Butterfly jointly models routing adaptation and probabilistic failures using SMT-based reasoning to check whether congestion-free operation can be maintained with sufficiently high likelihood. To improve scalability, Butterfly prunes irrelevant topology regions and reuses solver states across failure scenarios. We evaluate Butterfly on regional and full-scale ISP topologies, including the topology of a major European mobile operator. Our results show that Butterfly verifies congestion-related properties within practical time and memory limits.

***Index Terms*—Probabilistic Network Verification, Congestion-Aware Routing, Network Reliability, SMT-Based Analysis, Failure Resilience, Adaptive Routing**

I. Introduction

Modern networks, from Software-Defined Network (SDN)-enabled data centers to Internet Service Provider (ISP) backbones, must deliver both reliability and performance despite component failures. Operators reason about two classes of correctness properties: *hard properties*, such as reachability, which ensure uninterrupted connectivity under admissible failures, and *soft properties*, such as congestion-freedom, which guarantee that traffic remains within service-level agreement (SLA) limits. While transient overloads may be tolerable, sustained congestion on critical links leads to SLA violations and degraded user experience.

Operational failures are a primary cause of such violations. When links or routers fail, rerouting mechanisms shift traffic onto fewer paths and overload specific links, which causes persistent congestion and packet loss. Empirical studies show that more than 90% of service outages arise from failure-induced traffic load violations [1]. Beyond degraded performance, these violations generate substantial financial losses. Telecom providers have reported multi-million-dollar penalties after prolonged outages [2], and Google issued 10–25% customer credits following a large-scale connectivity incident [3].

Recent advances in probabilistic network verification, such as NetDice [4], model uncertainty by assigning probabilities to network states and verifying properties under deterministic routing derived from static link weights (e.g., shortest paths or Equal-Cost Multi-Path (ECMP)). However, this assumption does not reflect operational reality. In practice, routing adapts to failures, and traffic-engineering systems may adjust paths in response to load. Moreover, multiple feasible paths may exist. Whether congestion arises depends not only on the failure scenario, but also on the routing decisions taken in response. Verifying congestion under fixed routing therefore provides only a partial view of network correctness.

This gap motivates our central question: *Can we formally verify congestion properties under probabilistic failures while accounting for adaptive routing decisions?* Addressing this question requires bringing routing optimization and probabilistic verification into the same analysis.

We present **Butterfly**, a probabilistic verification framework that integrates congestion-aware routing directly into the verification process. Butterfly reasons about routing decisions under probabilistic failures, and checks whether a feasible routing assignment can preserve connectivity while keeping selected critical links within capacity. By formulating verification as an incremental optimization problem and reusing solver state across related failure scenarios, Butterfly scales to a large ISP topology with thousands of nodes and links in our evaluation.
Contributions. Butterfly advances probabilistic verification along three dimensions: (i) congestion-aware reasoning that models routing choices under failures, (ii) scalable symbolic exploration via incremental solving and solver-state reuse, and (iii) constraint-space pruning that reduces verification complexity by focusing on topology regions relevant to the analyzed flows. Together, these contributions allow operators to estimate how likely a network is to remain congestion-free under modeled failures.

II. Related Work

Formal network verification has been widely used to check properties such as reachability, isolation, and loop freedom. Systems such as Veriflow [5], Batfish [6], ARC [7], and Minesweeper [8] efficiently analyze network configurations, but do not model probabilistic failures. NetDice [4] extends this line of work by introducing probabilistic reasoning over failures, but assumes fixed routing and does not account for alternative feasible paths under failure.

Traffic-load verification has been explored in QARC [9], Jingubang [10], and YU [1]. These systems analyze congestion under failures, but do not combine probabilistic failure analysis with congestion-aware routing decisions.

Traffic engineering and congestion-aware routing systems, such as B4 [11], SWAN [12], and Jupiter [13], dynamically adapt routing to improve utilization and performance. How-

ISBN 978-3-903176-82-9

ever, these approaches optimize traffic distribution and do not provide formal guarantees under probabilistic failures.

Butterfly combines these directions by checking whether congestion-free routing exists under probabilistic failures.

III. SYSTEM MODEL

A. Network and Property Model

We model the network as an undirected graph $G = (V, E, W)$, where V denotes the set of nodes, E the set of links, and W the corresponding link weights. We assume that nodes can be partitioned into five layers that reflect typical large-scale topologies: access, backhaul, aggregation, border and route-reflector layers. This layered organization is an assumption used by our *pruning* (scalability) procedure; the main routing and verification algorithm itself operates on the graph G and does not rely on the assumption for *correctness*.

Rather than an individual transport-layer connection, a *flow* represents an aggregate, SLA-level traffic demand between a source (e.g., a router) and a destination (e.g., a prefix outside the AS). Formally, we define a flow as a tuple $f = (s, d, t)$, where s and d denote the source and destination, and $t \in \mathbb{N}$ the traffic demand for flow f. The set of all flows in the network is denoted by $\mathcal{F} = \{f_1, f_2, \ldots, f_{|\mathcal{F}|}\}$, where each $f \in \mathcal{F}$ corresponds to a distinct source–destination demand pair.

Given a property φ (e.g., congestion-freedom), the goal is to verify whether it holds under failures. Properties are classified as *single-flow* (e.g., reachability) or *multi-flow* (e.g., ensuring aggregate load does not exceed link capacity).

B. Routing Protocols

Butterfly models standard intra- and inter-domain routing.

Intra-AS routing. Routing within the AS uses static routes and IGP (OSPF/IS-IS). Static routes handle reliable traffic, while IGP computes shortest paths and adapts to failures.

Inter-AS routing. External connectivity is managed via BGP. Border routers exchange updates with external peers, and route reflectors distribute them internally to ensure consistent next-hop selection.

C. Failure and Property Distributions

Failure Model. We model network failures using random variables over links and nodes. For each link $e \in E$, we define an independent Bernoulli random variable $L_e \in \{0, 1\}$, where $L_e = 1$ indicates that e is operational and $L_e = 0$ indicates failure. We assume uniform link reliability $\mathbb{P}[L_e = 1] = p_\ell$ and independence across links, following prior work [4].

To capture shared-risk router failures, we adopt a *node-correlation group model*. Let p denote the target node failure probability. Nodes are partitioned into disjoint groups representing failure domains. For layered topologies, we use five groups corresponding to the *access*, *backhaul*, *aggregation*, *border*, and *route-reflector* layers.

For each group g, we introduce a Bernoulli random variable $G_g \in \{0, 1\}$ indicating a group-level failure with probability $\mathbb{P}[G_g = 1] = p_g$. For each node v in group g, we introduce an independent Bernoulli variable $I_v \in \{0, 1\}$ representing a residual node-specific failure, with probability $\mathbb{P}[I_v = 1] = p_i$. The operational state of node v is defined as $N_v = (1 - G_g)(1 - I_v)$, so that $N_v = 1$ indicates that v is operational.

We set p_i to control the marginal node failure probability as a function of p_g:

$$\mathbb{P}[N_v = 0] = p_g + (1 - p_g)p_i, \qquad p_i = \frac{p - p_g}{1 - p_g}.$$

When $p_g \geq p$, we set $p_i = 0$, in which case the marginal node failure probability is p_g. For flat topologies, we use a single group with $p_g = 0$, which reduces to independent node failures with probability p.

Network State and Failure Scenarios. A network state is defined as $\omega = \big((L_e)_{e \in E}, (N_v)_{v \in V}\big)$, where L_e and N_v denote the operational states of links and nodes. The joint probability distribution over ω is induced by:

(i) independence of link variables (L_e), (ii) independence of node residual variables (I_v) across nodes, and (iii) independence of group variables (G_g) across groups.

Under state ω, a link $e = (u, v)$ is usable if and only if $L_e = 1 \ \wedge \ N_u = 1 \ \wedge \ N_v = 1$. Thus, node failures implicitly disable all incident links.

A *failure scenario* corresponds to a realization of ω. Given a user-defined property φ (e.g., reachability or congestion-freedom), we define

$$\varphi : \Omega \to \{0, 1\}, \qquad \omega \mapsto \varphi(\omega),$$

where $\varphi(\omega) = 1$ if the property holds under state ω. The probabilistic verification objective is to bound

$$\mathbb{P}[\varphi(\omega) = 1] = \sum_{\omega : \varphi(\omega) = 1} \mathbb{P}(\omega). \tag{1}$$

D. Congestion Metrics

Each flow $f = (s, d, t_f) \in \mathcal{F}$ contributes a traffic demand t_f between its source s and destination d. Let $x_f(e) \in \{0, 1\}$ be a binary variable indicating whether flow f is routed over link $e \in E$. The total *load* on link e is the aggregate traffic of all flows traversing it, given by $\sum_{f \in \mathcal{F}} t_f \cdot x_f(e)$.

The *utilization* of a link e under a routing assignment r ($r : \mathcal{F} \to 2^E$) is defined as the ratio of its load to its capacity $c(e)$:

$$U_r(e) = \frac{\sum_{f \in \mathcal{F}} t_f \cdot x_f(e)}{c(e)}. \tag{2}$$

A link is considered *congested* if its utilization exceeds unity, i.e., $U_r(e) > 1$. While utilization is defined for all links, congestion-freedom is enforced only over a subset of operator-selected *critical links* $C \subseteq E$. These links may be manually specified based on historical data or automatically identified via network sensitivity analysis. Later in Section IV, we introduce *hot edges*, links that frequently appear on shortest paths and often overlap with C.

A network configuration is valid only if all critical links satisfy their capacity limits. The congestion-freedom property requires that:

$$\varphi_{\text{cong free}} : \forall e \in C, \quad U_r(e) \leq 1. \tag{3}$$

We encode this as a routing constraint, requiring all feasible solutions to maintain utilization within limits. The resulting formulation is solved symbolically (e.g., using Z3) to verify the existence of a valid routing and to compute its probability under failure scenarios.

E. Probabilistic Verification

Each network state ω represents a unique combination of link failures and survivals. Butterfly checks whether a given property φ holds in that state. If φ is satisfied, the probability of that state contributes to the overall probability of correctness.

Formally, the probability that a property φ holds under the probabilistic failure model is defined as in Eq. (1).

IV. Approach

We present Butterfly, a probabilistic verifier for congestion-aware routing under node and link failures. It jointly models routing and link utilization, enabling congestion-aware verification. We build on NetDice and describe our symbolic encoding and probabilistic integration.

A. Recap of NetDice

NetDice [4] models the network as a weighted graph with independent link failures and explores the failure space using hot-/cold-edge pruning. **Hot edges** are those whose failure may affect the property, while **cold edges** can be safely ignored.

For each network state ω, NetDice evaluates a property φ (e.g., reachability) and computes its probability as $\mathbb{P}[\varphi(\omega) = 1]$. In the case of congestion, routing is recomputed per failure scenario based on control-plane protocols (e.g., OSPF/BGP), and then the resulting paths are checked against link capacities.

However, routing remains protocol-driven and fixed after recomputation: NetDice does not adapt routing to mitigate congestion. It only verifies whether the protocol-derived routing leads to overload.

Limitation. Consider a link e operating near capacity with an alternative e'. If e fails, traffic shifts to e' and feasibility is checked. But if e remains up yet becomes overloaded, NetDice does not reroute traffic to e'. In contrast, Butterfly encodes routing as a decision variable and verifies whether a congestion-free routing exists under each failure, enabling adaptive traffic redistribution.

B. BGP Configuration Modeling

We model BGP via a preprocessing step prior to SMT encoding. Following the NetDice worst-case assumption, all inter-domain attributes are equal (e.g., `LOCAL_PREF`, `AS_PATH`, `MED`, and community policies), so best-path selection depends only on IGP distance to reachable border routers.

Let $\mathcal{B}$ and $\mathcal{R}$ denote border routers and route reflectors in $G = (V, E)$. For each flow i, source s and failure scenario, we compute the connected component $\mathrm{CC}(s_i)$ and restrict routing to reachable nodes:

$$\mathcal{B}'_i = \mathcal{B} \cap \mathrm{CC}(s_i), \quad \mathcal{R}'_i = \mathcal{R} \cap \mathrm{CC}(s_i).$$

BGP reachability and exit selection are handled outside the SMT solver, while Z3 verifies forwarding and congestion constraints. This models a conservative setting where all reachable border routers are eligible and differ only by IGP cost.

C. Reasoning over Failures and Congestion-Aware Routing

Building on NetDice, Butterfly treats routing as part of the verification space, jointly reasoning about path availability under failures and the existence of a routing that satisfies link capacity constraints.

Formally, let $r(f)$ denote the route assigned to flow $f \in \mathcal{F}$. Under a given network state ω, NetDice evaluates

$$\varphi_{\text{NetDice}}(\omega) = \begin{cases} 1, & \text{if no link is overloaded under a fixed routing,} \\ 0, & \text{otherwise.} \end{cases}$$

In contrast, Butterfly verifies the existence of a routing assignment r such that

$$\exists r(f) : \forall e \in C, \quad U_r(e) \leq 1,$$

where $U_r(e)$ denotes the utilization of link e induced by routing $r : \mathcal{F} \to 2^E$, as defined in Section III-D. That is, Butterfly checks whether there exists a routing configuration that keeps the utilization of all critical links within capacity limits under the given failure scenario (see Eq. (3)).

D. SMT-Based Congestion Verification

Butterfly extends NetDice by replacing static checks with SMT-based reasoning over congestion-aware routing. For each failure scenario, it encodes routing feasibility as hard constraints and preferences as soft constraints. The instance is satisfiable iff a congestion-free routing exists; otherwise, the property is violated.

1) Encoding Variables

As defined in § III-D, for each flow $f \in F$ and link $e \in E$, $x_e^f \in \{0, 1\}$ indicates whether flow f routes over link e.

2) Hard Constraints:

Hard constraints capture the logical feasibility conditions that must always hold: conditions (a)–(c) must be simultaneously satisfied for a scenario to be considered congestion-free.

(a) Flow-Conservation Constraints For each flow f and node $v \in V$:

$$\sum_{(v,u)\in E} x^f_{(v,u)} - \sum_{(u,v)\in E} x^f_{(u,v)} = \begin{cases} 1, & \text{if } v = s_f \\ -1, & \text{if } v = d_f \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Although the topology is modeled as an undirected graph $G = (V, E)$, each link $\{u, v\}$ is represented by two directed arcs (u, v) and (v, u) sharing the same failure variable and capacity. This preserves bidirectional behavior and enables standard flow-conservation constraints, ensuring flows start at the source, end at the destination, and are conserved at intermediate nodes [14], [15].

(b) Failure-Aware Routing

Routing decisions must respect link availability:

$$x_e^f \leq L_e, \qquad \forall f \in \mathcal{F},\ \forall e \in E \tag{5}$$

If a link fails ($L_e = 0$), the solver is forced to set $x_e^f = 0$, effectively removing the link from the feasible topology.

Node failures are handled during preprocessing. If a node $v \in V$ fails in a given network state, all incident links e are treated as failed by setting their corresponding L_e variables to 0 before solving the SMT constraints. Thus, node failures are reduced to link failures in the encoding.

(c) Congestion (Capacity) Constraints

Ensure that the total carried traffic demand on any critical link does not exceed its capacity (see Eq. (3)).

3) Soft Constraint and Optimization Objective

When multiple feasible routings exist, Butterfly biases the solver toward those that best reflect practical routing policies by minimizing the total path cost:

$$\min \sum_{f \in \mathcal{F}} \sum_{e \in E} w(e) \cdot x_e^f \tag{6}$$

where $w(e)$ is the weight of link e. This soft constraint does not affect satisfiability; it selects the minimum-cost routing among feasible solutions. Using Z3's Optimize module, Butterfly enforces hard constraints while minimizing this objective.

4) Satisfiability Semantics

For each network state ω, the SMT instance is:

- *SAT*: a congestion-free routing exists; the state contributes $\mathbb{P}[\omega]$,
- *UNSAT*: no feasible routing exists; the state is a violation.

E. Evaluating the Property under Failures

Butterfly extends NetDice by replacing reachability checks with SMT-based feasibility for congestion-free routing. For each network state ω, the solver checks whether a feasible routing exists; if so, the state contributes $\mathbb{P}[\omega)$ to $\mathbb{P}[\varphi(\omega) = 1]$, otherwise it is a violation. As in NetDice, Butterfly leverages hot-/cold-edge pruning to avoid exploring irrelevant states.

After solving, Butterfly identifies *property-related hot edges* (used by flows) and *protocol-related hot edges* (required for BGP correctness). Their union defines the set of *hot edges*, and failure exploration is restricted to these edges.

Butterfly explores failures by iteratively disabling hot edges and re-evaluating feasibility. Valid states contribute their probability and are expanded recursively. A heap-based priority queue orders states by probability, ensuring efficient exploration of the most relevant scenarios.

1) Topology-Aware Constraint Pruning

Operator networks are typically hierarchical across different layers. For a flow $f = (s, d, t)$, only a subset of nodes and links can participate in feasible $s \to d$ paths. Butterfly exploits this by constructing a *relevant subgraph* $G_f = (V_f, E_f)$ containing all nodes and links on at least one simple path between s and d, along with inter-layer connections.

Constraint generation is then restricted to G_f, setting $x_{(u,v)}^f = 0$ for all $(u, v) \notin E_f$. This reduces the number of variables and constraints in practice, often by nearly an order of magnitude, while preserving satisfiability as long as G_f contains all feasible paths for flow f.

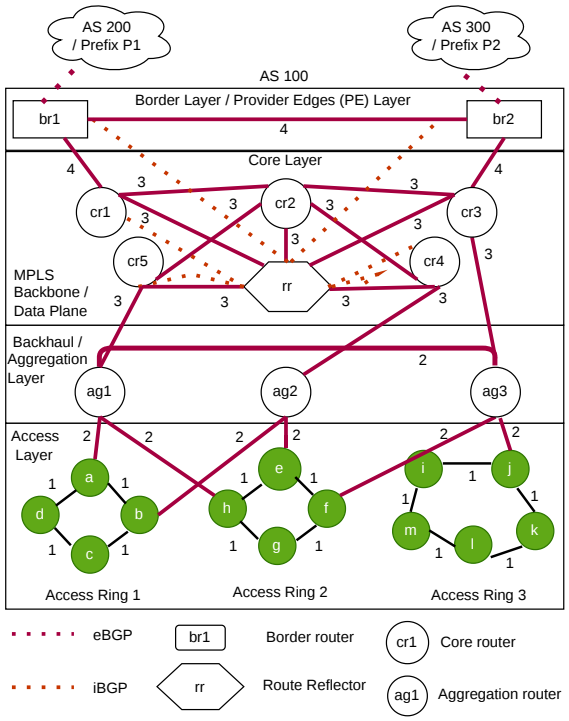


Fig. 1: Example network topology.

Lemma 1 (Soundness of Pruning)**.** *Let $\Phi(G)$ denote the SMT encoding of the full network and $\Phi(G_f)$ the encoding restricted to the subgraph G_f. Then, for any flow f, $\Phi(G)$ is satisfiable if and only if $\Phi(G_f)$ is satisfiable.*

Proof. By construction, every feasible $s_f \to d_f$ path in G is contained within G_f, since G_f includes all nodes and links that lie on at least one simple path between s_f and d_f in G. Thus, any satisfying assignment for $\Phi(G)$ can be projected onto G_f to yield a satisfying assignment for $\Phi(G_f)$. Conversely, any feasible routing in G_f is trivially feasible in G because $G_f \subseteq G$. Thus, pruning links outside E_f preserves satisfiability and does not eliminate any feasible solution. □

2) Incremental Solving

Butterfly uses Z3's incremental solving to reuse constraints across failure scenarios. For each new failure, it applies `push`, asserts $L_e = 0$, checks satisfiability, and then restores the state with `pop`. This avoids rebuilding the model and enables efficient reuse of solver state.

F. Illustrative Example

We illustrate Butterfly on the topology in Fig. 1, inspired by real configurations in a mobile WAN with access, aggregation, core, and border layers. We consider two flows: $f_1 = (h, P_1, 7 \text{ Gbps})$ and $f_2 = (a, P_2, 6 \text{ Gbps})$. links have capacity 10 Gbps. For simplicity, we consider only link failures in our example and assume that links fail independently with probability 0.001, consistent with reliability values observed in large ISP backbones. Node failures are not considered in this setting. The objective is to verify that the critical link

$l_{ag1,cr5}$ is not overloaded under failures. Butterfly applies the flow-conservation rule for all nodes in Fig. 1 regarding both flow f_1 and f_2. All constraints corresponding to Access Ring 3 can be removed when analyzing flow f_1 and f_2, as this ring lies outside the relevant reachability domain of these flows. We must ensure that the total traffic on the critical link does not exceed its capacity (see Eq. (3)):

$$\frac{7 \cdot x^{f_1}_{ag1,cr5} + 6 \cdot x^{f_2}_{ag1,cr5}}{10} \leq 1 \ . \tag{7}$$

The solver assigns $x^f_e = 1$ to links that carry traffic, thereby defining end-to-end paths for each flow. The set of *hot edges*:

$$\{\, l_{h,ag1},\ l_{ag1,cr5},\ l_{cr5,cr2},\ l_{cr2,cr1},\ l_{cr1,br1},\ l_{a,ag1},\ l_{ag1,ag3},\ l_{ag3,cr3},\ l_{cr3,br2},\ l_{rr,cr1},\ l_{rr,cr3} \,\}$$

consists of links that are critical either for carrying flows f_1 and f_2 without congesting critical link and for maintaining correct BGP behavior.

The probability that all hot edges are operational is $(0.999)^{11} \approx 0.989$. Butterfly explores failures over these edges by disabling failed links in the model and checking whether a feasible routing still exists. It accumulates the probabilities of scenarios that satisfy the congestion constraint while discarding violating ones, recursively exploring only relevant failures to efficiently estimate the probability of congestion-freedom.

V. Experimental Evaluation

We evaluate Butterfly along various dimensions. **(Q1)** How does Butterfly scale with the number of concurrent flows and topology size in terms of runtime and memory consumption? **(Q2)** How does topology-aware constraint pruning reduce verification complexity? **(Q3)** How does Butterfly's incremental solving strategy improve the efficiency of probabilistic failure-scenario exploration? **(Q4)** Does Butterfly enable practical multi-flow verification in large real-world topologies where prior approaches fail to complete? We conducted all experiments on a MacBook Pro with an M2 processor 24 GB of RAM.

A. Production WAN Benchmarking

We evaluate our approach on a real-world ISP topology obtained from a European network operator. The dataset is proprietary and has been anonymized due to confidentiality constraints; we refer to it as AS-5951 throughout the paper. The topology comprises 5,119 nodes and 6,442 links, representing a nation-scale ISP backbone spanning multiple layers, including access, backhaul, aggregation, route reflector, and border layers. The BGP configuration includes 24 external peers and 8 route reflectors, and link weights are set according to operational IGP costs.

We also extract three regional sub-topologies representing different scales. Region-1 consists of 226 routers and 1,060 links, Region-2 includes 323 routers and 1,170 links, and Region-3 comprises 627 routers and 1,500 links; each uses 2 route reflectors.

B. Methodology

We evaluate Butterfly on congestion as a multi-flow property, scaling the number of concurrent flows from $k = 2$ to 20 (vs. $k \leq 8$ in prior work). Each configuration is repeated 10 times. Flows are random source–destination pairs with traffic demands uniformly sampled from $[1, 1000]$ (precision 10^{-4}).

Despite the modest number of flows, this setting is realistic, as a few high-bandwidth flows can already saturate critical links and cause SLA violations [16].

To assess congestion, we select a random internal link as the critical link (capacity 500 units) and verify whether it remains uncongested. Failures follow standard assumptions: links and nodes fail independently with probabilities 0.001 and 0.0001, respectively [4], [17], [18].

C. Scalability (Q1 and Q4)

Fig. 2(a)-(b) show that Butterfly's runtime scales near-linearly with the number of flows across both regional and full (AS 5951) topologies, remaining within practical limits (2-hour threshold as NetDice's timeout value) and exhibiting no exponential growth. In contrast, NetDice fails to complete within the timeout, highlighting Butterfly's scalability. Fig. 3 shows that memory usage also scales smoothly with flow count, reaching about 1.0–1.45 GB for regional topologies at 20 flows and ∼1.1,GB for AS 5951. Overall, memory growth is steady and predictable.

D. Constraint Pruning and Solver Complexity (Q2)

Fig. 4 shows that topology-aware pruning reduces problem size by over 80%. At 20 flows, variables drop from 129k to 21k and constraints from 361k to 47k. This reduction grows with flow count and is key to scalable multi-flow verification.

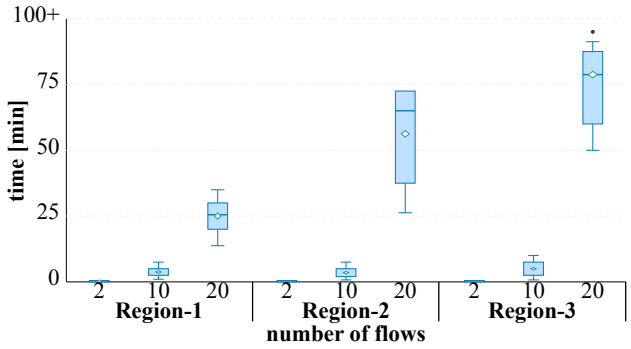
E. Incremental Solving for Efficient Failure Exploration (Q3)

In NetDice, each failure scenario (e.g., failing a hot edge) triggers a full recomputation of routing, where paths between the source and destination are recomputed from scratch for every flow. In contrast, Butterfly leverages Z3's incremental solving (`push`/`pop`) to avoid re-solving the entire problem: only constraints affected by the failure are updated, while the rest of the solver state is reused. This enables efficient exploration by preserving previously asserted constraints and learned clauses across scenarios. Empirically, as shown in Fig. 5, Butterfly achieves significantly higher reuse: 100% reuse in 30% of states and >80% in 66.5%, compared to <1% and 16.5% for NetDice. This demonstrates the efficiency gains of incremental solving for large-scale failure exploration.

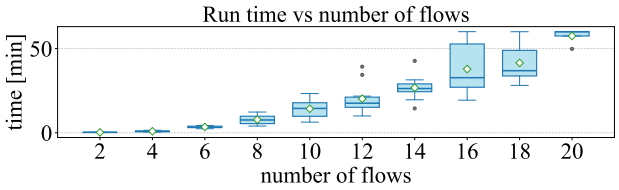
VI. Conclusions

We introduced Butterfly, a scalable framework for probabilistic verification of congestion-aware routing. By modeling the interaction between adaptive routing and probabilistic failures, Butterfly captures how networks redistribute traffic under stress.

Combining topology-aware pruning with incremental SMT solving, Butterfly reduces verification complexity while preserving probabilistic soundness. Our evaluation on a large ISP



(a) Region-1, 2, 3, congestion



(b) AS 5951, congestion

Fig. 2: Runtime comparison across regions and properties (median runtime per flow group).

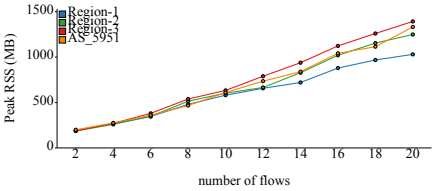


Fig. 3: Peak RSS across topologies (median per flow count).

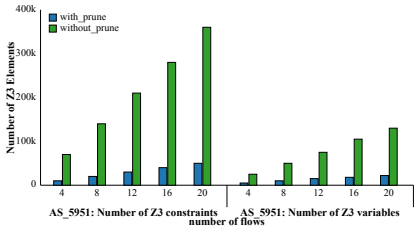


Fig. 4: Number of Z3 variables and constraints with and without pruning across flow counts in AS 5951.

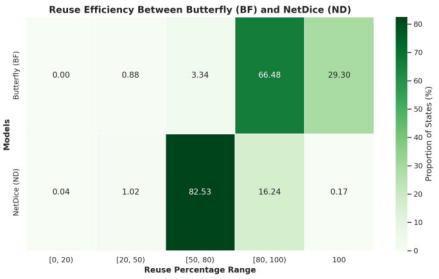


Fig. 5: Reuse efficiency comparison between Butterfly (BF) and NetDice (ND) across reuse percentage ranges.

topology shows that it scales to thousands of nodes and verifies congestion properties within practical time limits.

References

[1] R. Li, Y. Yuan, F. Ye, M. Liu, R. Yang, Y. Yu, T. Guo, Q. Ma, X. Zeng, C. Xu, D. Cai, and E. Zhai, "A general and efficient approach to verifying traffic load properties under arbitrary k failures," in *ACM SIGCOMM 2024*.

[2] FasterCapital, "The cost of downtime: Navigating penalty clauses in slas," https://www.fastercapital.com/content/Penalty-Clauses--The-Cost-of-Downtime--Navigating-Penalty-Clauses-in-SLAs.html, 2025, accessed: 2025-09-08.

[3] Google Cloud Status Dashboard, "Google compute engine networking outage (incident 16007): Sla credits," https://status.cloud.google.com/incident/compute/16007.

[4] S. Steffen, T. Gehr, P. Tsankov, L. Vanbever, and M. Vechev, "Probabilistic verification of network configurations," in *ACM SIGCOMM 2020 Conference*.

[5] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: Verifying network-wide invariants in real time."

[6] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein, "A general approach to network configuration analysis," in *USENIX Conference on Networked Systems Design and Implementation, NSDI'15*.

[7] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.

[8] R. Beckett, A. Fogel, R. Mahajan, T. Millstein, and D. Walker, "Minesweeper: A general approach to network configuration verification," in *Proceedings of SIGCOMM*, 2017, pp. 456–469.

[9] K. Subramanian, A. Abhashkumar, L. D'Antoni, and A. Akella, "Detecting network load violations for distributed control planes," in *ACM SIGPLAN*, 2020.

[10] R. Li, F. Ye, Y. Yuan, R. Yang, B. Tian, T. Guo, H. Wu, X. Zhu, Z. Guan, Q. Ma, X. Zeng, C. Xu, D. Cai, and E. Zhai, "Reasoning about network traffic load property at production scale," in *USENIX Symposium on Networked Systems Design and Implementation, NSDI'24)*.

[11] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: experience with a globally-deployed software defined wan," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, p. 3–14, 2013.

[12] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," *SIGCOMM Comput. Commun. Rev.*, 2013.

[13] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," in *ACM Conference on Special Interest Group on Data Communication*, 2015.

[14] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2009.

[16] R. Mahajan, S. Floyd, and D. Wetherall, "Controlling high-bandwidth flows at the congested router," in *International Conference on Network Protocols (ICNP)*, 2001.

[17] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," *SIGCOMM Comput. Commun. Rev.*, 2013.

[18] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage, "California fault lines: understanding the causes and impact of network failures," *SIGCOMM Comput. Commun. Rev.*, 2010.