Debugging Network Reachability with Blocked Paths

S. Bayless[1], J. Backes[1], D. DaCosta[1], B. F Jones[1], N. Launchbury[1], P. Trentin[1], K. Jewell[1], S. Joshi[1], M. Q. Zeng[1], N. Mathews[1]

[1]Amazon Web Services

Abstract

In this industrial case study we describe a new network troubleshooting analysis used by VPC REACHABILITY ANALYZER, an SMT-based network reachability analysis and debugging tool. Our troubleshooting analysis uses a formal model of AWS Virtual Private Cloud (VPC) semantics to identify whether a destination is reachable from a source in a given VPC configuration. In the case where there is no feasible path, our analysis derives a *blocked path*: an infeasible but otherwise complete path that would be feasible if a corresponding set of VPC configuration settings were adjusted.

Our blocked path analysis differs from other academic and commercial offerings that either rely on packet probing (e.g., TCPTRACE) or provide only partial paths terminating at the first component that rejects the packet. By providing a complete (but infeasible) path from the source to destination, we identify for a user all the configuration settings they will need to alter to admit that path (instead of requiring them to repeatedly re-run the analysis after making partial changes). This allows users to refine their query so that the blocked path is aligned with their intended network behavior before making any changes to their VPC configuration.

1 Introduction

This paper describes a new network connectivity troubleshooting analysis used by VPC REACHABILITY ANALYZER, a service that analyzes Amazon Web Services' (AWS) Virtual Private Cloud (VPC) configurations.

VPCs are user-configured networks of virtual compute devices and resources. AWS VPC offers dozens of networking components and controls to give users flexibility in configuring their networks. Access to these resources is logically isolated within virtual networks configured by the users. As VPCs grow in size and complexity, users can increasingly benefit from automation to identify and resolve misconfigurations, as well as to validate that applications maintain security and availability invariants through infrastructure changes.

VPC REACHABILITY ANALYZER uses the TIROS [2] formal model of AWS VPC networking semantics to identify whether a destination is reachable from a source in a given VPC configuration. If the destination is reachable, then TIROS identifies a *feasible path* from the source to the destination, where a path is a sequence of network components associated with incoming and/or outgoing packet header assignments (protocol, addresses, ports). The outgoing packet header of one component is the incoming packet header of the next component. Paths may also identify relevant VPC configuration details such as the specific routes, firewall rules, or other settings admitting the packet at each step. Each component in a VPC may accept or reject incoming and outgoing packet headers; a *feasible path* is a path in which every component on the path accepts both its incoming and outgoing packet header.

Tiros's analysis is static, *i.e.*, Tiros does not inject traffic into VPC configurations, and is complete for the subset of AWS VPC semantics it supports: if there exists a path connecting the source and destination, Tiros will find it. Since 2018, Tiros has powered the commercially available *Network Reachability* assessment in Amazon Inspector [1], statically identifying ports on EC2 Instances (virtual machines) accessible outside of their VPCs.

In this work, we extend Tiros by introducing a new diagnostic *blocked path* analysis when there is not a feasible path, to help users understand why their query is infeasible. A *blocked path* is a path as defined above, in which at least one component rejects its incoming or outgoing packet, along with one or more *blocking reasons*: elements of the VPC configuration preventing one or more components on the path from accepting packets. The blocked path identifies a sufficient set of blocking reasons, such that if each were addressed the query would be satisfiable.

Previous tools for connectivity diagnosis typically provide a partial path, up to the first component/rule that rejects the packet; in some cases those tools also identify a single blocking reason. Remediations based on a partial path may address that initial blocking reason only to discover that remediations are still necessary, or that the remediation may be working towards a path that the user ultimately will reject. Providing a complete blocked path connecting the source and destination allows users to ensure that their intent is aligned with our diagnosis before taking any corrective actions.

Our contributions in this work are:

1. Identifying the notion of a blocked path as a useful medium for conveying a network diagnosis and aligning it with a user's intent,
2. Demonstrating how blocked paths can be efficiently derived at scale,
3. Describing VPC Reachability Analyzer, a commercial tool based on these insights.

2 Background

2.1 Related Works

Many previous works have proposed network reachability diagnosis tools, including both widely-used industry tools and academic literature. These tools can be broadly divided into model-based and non-model-based approaches.

Non-model-based network diagnostic tools include system applications such as IPTRACE and TCPTRACE, commercial tools such as *Cisco Packet Tracer* [7], and academic works such as *Tulip* [12]. These tools trace live packets through a network or routing device, identifying the sequence of addresses of devices that accept the packet. Packet tracing tools lack visibility into the configuration settings that block and route packets.

Model-based tools [2, 5, 6, 13, 16] statically analyze reachability between a specified source and destination in a network or routing device. Rather than transmitting live packets, these tools use formal methods such as constraint solvers to rigorously identify feasible paths. Existing model-based tools provide control-plane level information when there is a feasible path, but produce either no information for unreachable paths, or identify only the first (out of potentially many) reasons why a path is blocked.

Our blocked path analysis is based on deriving minimal correction subsets (described below), which several previous works have proposed for general-purpose SAT-based error diagnosis or repair [4, 9, 17, 8].

2.2 Minimal Correction Subsets

The blocked path analysis we describe in Section 3 relies on two related concepts: Maximal Satisfiable Subsets (MSS) and Minimal Correction Subsets (MCS), which we define below. Following the definitions from [14]:

Definition 1 (MSS). *$\mathcal{S} \subseteq \mathcal{F}$ is a Maximal Satisfiable Subset of constraints $\mathcal{F}$ iff $\mathcal{S}$ is satisfiable and $\forall c \in \mathcal{F} \setminus \mathcal{S}, \mathcal{S} \cup \{c\}$ is unsatisfiable.*

Definition 2 (MCS). *$\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset of constraints $\mathcal{F}$ iff $\mathcal{F} \setminus \mathcal{C}$ is satisfiable and $\forall c \in \mathcal{C}, (\mathcal{F} \setminus \mathcal{C}) \cup \{c\}$ is unsatisfiable.*

The complement of an MCS, $\mathcal{F} \setminus MCS(\mathcal{F})$, is guaranteed to be a maximal satisfiable subset of $\mathcal{F}$; for this reason the MCS is sometimes called the coMSS.[1]

In general, the MCS and MSS are not guaranteed to be unique. There is a close connection between the definition of a Maximal Satisfiable Subset and MaxSAT [10]: The largest MSS (and therefore smallest MCS) corresponds to a solution to MaxSAT. Indeed, one approach for computing the MCS is to compute MaxSAT and take the complement. Efficient algorithms for directly computing the (not necessarily smallest) MCS without computing MaxSAT are available and are typically much faster than computing MaxSAT; a good survey of MCS algorithms including an empirical evaluation can be found in [14].

In constraint optimization problems, it is common to consider hard and soft constraints, in which only the soft constraints may be relaxed. Definition 2 assumes that all constraints are soft, but can be easily extended to support a mix

[1] Note that a minimal correction subset is a distinct concept from an unsatisfiable core [11]. An unsatisfiable core is always unsatisfiable, but its complement $\mathcal{F} \setminus CORE(\mathcal{F})$ is not guaranteed to be satisfiable; in contrast, an MCS may or may not be satisfiable, but its complement is guaranteed to be satisfiable.

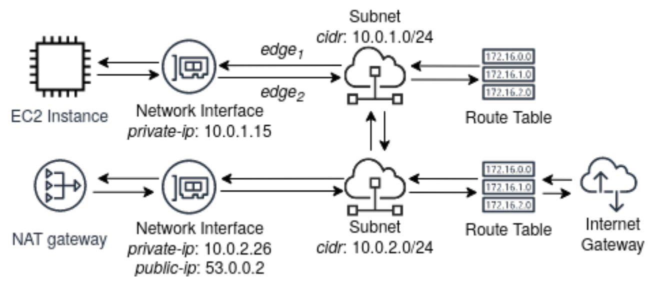
of soft and hard constraints (where the MCS must contain only soft constraints). In this case, the MCS is only well defined if the hard constraints are satisfiable.

In Section 4, we will use a function COMPUTEMCS(*Soft*, *Hard*) that supports both hard and soft constraints. COMPUTEMCS returns a minimal correction set $\mathcal{C} = MCS(Soft \cup Hard)$, with $\mathcal{C} \subseteq Soft$. Our implementation of COMPUTEMCS uses a simple binary search, similar to FastDiag [4], or Algorithm BFD from [14]. We add activation literals to the soft constraints to allow the underlying solver instance to be re-used incrementally while testing different subsets of soft constraints for satisfiability.

2.3 Network Reachability

We use the SMT-encoding of AWS VPC network semantics previously described in TIROS [2]. In this section, we briefly review this graph-based encoding; we refer readers to [2] for more details.

We take as input a configuration describing one or more user VPCs, and a user-specified reachability query, consisting of a source and destination component in the VPC. For example, the source of the query may be an internet gateway, and the destination may be an EC2 Instance. A query may also optionally specify additional constraints, such as the protocol, a range of source or destination addresses or ports for the packet, or an intermediate component that must (or must not) be on the path.



Packet Header	
protocol	bv:8
srcAdr	bv:32
dstAdr	bv:32
srcPort	bv:16
dstPort	bv:16

$$((dstAdr \neq 10.0.1.15) \implies \neg edge_1)$$
$$((srcAdr \neq 10.0.1.15) \implies \neg edge_2)$$

Fig. 1. Simplified example symbolic graph representation of a VPC (*left*), with symbolic packet header consisting of bitvectors (*right*). Edges in the graph are associated with theory atoms, and are traversable only if those atoms are assigned true. Two example constraints, enforcing that a network interface is only accessible if the packet is addressed to/from that interface are shown. These constraints relate edge atoms in the symbolic graph to the bitvectors in the symbolic packet header to enforce AWS VPC semantics.

We encode VPC configurations as constrained symbolic graphs using the SMT solver MonoSAT [3], with fixed-width bitvectors representing the protocol, port, and addressing information in a symbolic packet header. Figure 1 shows a symbolic graph along with a packet header and example constraints.

VPC components are represented as a nodes in the symbolic graph. Each component has semantics governing which packets it will accept; these semantics are encoded as constraints that restrict which edges incident to that component's node are traversible, depending on the assignment of the packet header variables. A satisfying assignment to the full set of constraints corresponds to a feasible path. In such an assignment, the bitvector variable assignments provide the packet header(s) and the graph theory model provides a path of network component nodes connecting the source and destination of the users query.

Some components (such as NAT gateways) transform and retransmit packets. Tiros supports this by unrolling the VPC configuration graph into multiple copies with separate packet header variables. Edges from packet-transforming components connect to their components in the next unrolled section of the graph. Tiros unrolls the graph to a sufficient depth to model the behavior of the components for each query.

Query source and destination reachability is enforced with a single graph theory reachability predicate requiring a feasible path in the VPC configuration graph from the source to the destination of the query. Query restrictions requiring intermediate components are enforced using additional reachability predicates. Query restrictions requiring that a given resource not occur on a path are enforced by excluding that resource from the VPC configuration graph representation. Packet header restrictions are enforced using bitvector constraints.

If the constraints are satisfiable, Tiros extracts a reachable path satisfying the query from the satisfying assignment to the constraints. In the next section, we will discuss how we extend Tiros to also provide diagnostic feedback in the case where the constraints are unsatisfiable.

3 Blocked Paths for Network Configuration Diagnosis

We introduce the notion of *blocked path* for analyzing infeasible network connections. As shown in Figure 2, a blocked path is an infeasible but otherwise complete path from a source to a destination, in which one or more edges or nodes are annotated with *blocking reasons*: configuration settings or network semantics that explain why that transition in the path is infeasible.

Unlike a live packet trace, a blocked path continues past components that reject or redirect the packet so as to reach the user's intended destination, potentially transiting through multiple infeasible steps along the way.

Definition 3 (Blocked path).

1. *A blocked path is a complete (but infeasible) path from a source to a destination in a network, satisfying the user's query.*

2. *A blocked path is* actionable*: it is a path that could, with the right control plane configuration adjustments, be a feasible path.*
3. *A blocked path identifies a sufficient set of* blocking reasons *(network semantics or control-plane settings) that would need to be addressed to admit the packet along that blocked path. This may include multiple blocking reasons along the path, as opposed to just the first blocking reason.*

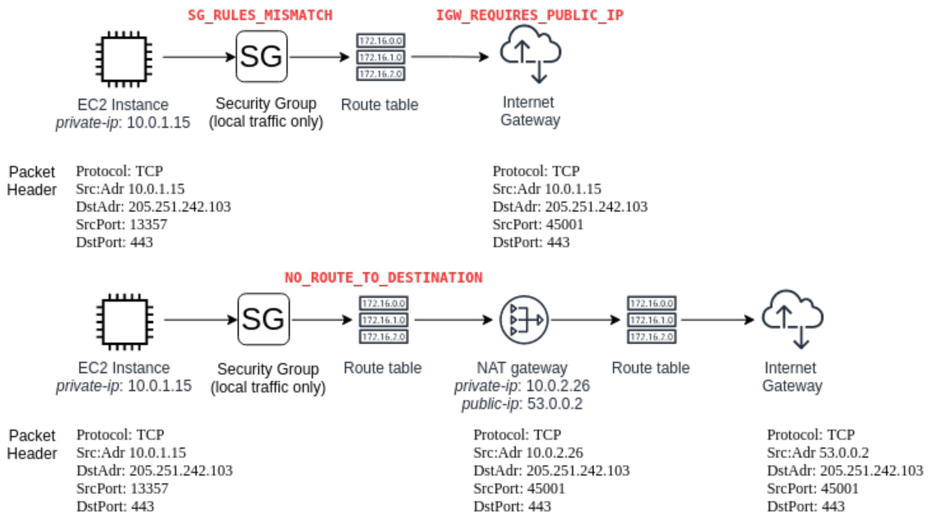


Fig. 2. Two alternative blocked paths from an EC2 instance to an internet gateway. These blocked paths take different routes, and have different *blocking reasons* (shown in red) that explain why those paths are infeasible. In the first blocked path, there are two blocking reasons: the security group egress rule rejects packets destined for the Internet, and the internet gateway requires that the source instance must have a public IP address. Note that although the packet would be rejected by the security group, the blocked path continues past the security group to identify a complete (but infeasible) path to the internet gateway. The second blocked path transitions through an intermediate NAT gateway, which satisfies the security group rule and also has a public IP address. However, this path is still blocked, because the route table does not have an applicable route to the NAT gateway.

Validating User Intent

Showing a complete path from the source to destination, along with all the relevant configuration settings blocking that path, allows users to confirm that this course of action matches their intended network behavior before making any changes. However, in many cases there are multiple ways to adjust a configuration to admit a path, resulting in different blocked paths.

For example, Figure 2 shows two example blocked paths to an internet gateway from an EC2 instance lacking a public IP address. Our analysis might initially produce for the user the shorter blocked path. Two remediation steps are required to admit this shorter path: The user must adjust the security group rule of the instance to admit egress packets to the public internet, and the user must also associate a public IP address with the source instance. Upon seeing the complete blocked path, the user may immediately determine that this would be the wrong solution for their network.

If the proposed blocked path doesn't match the user's intent, we allow users to submit a refined query so as to generate an alternative blocked path. For instance, the user may specify allowed address or port ranges for the packet, or specify components that must or must not appear on the path. Similarly, the user may submit a refined query specifying that a NAT gateway must be an intermediate component on the path. In this case, we might produce the longer blocked path from Figure 2.

Actionable Blocked Paths

In some cases, there may not exist any combination of VPC configuration adjustments that would allow a query to be satisfied. For example, under typical conditions in VPCs, route tables cannot be adjusted to redirect packets that are destined for a local address within the VPC. It is possible for users to specify queries that cannot be satisfied without violating this local route restriction.

In principle, it is possible to derive a blocked path with non-user-configurable blocking reasons, however the resulting paths may behave in misleading or confusing ways, and in general will not be possible for users to actually achieve in any real configuration of their VPC. If possible, we want to ensure that the path contains only user-configurable blocking reasons, so that we produce an *actionable* finding for users. However, we still want to be able to provide useful diagnostics in cases where no actionable blocked path is possible (*e.g.*, to explain to the user that the local route restriction will prevent their path).

In Section 4, we describe how we determine when it is not possible to produce a blocked path without including non-configurable blocking reasons. In this case, we produce a partial path up to that first non-configurable blocking reason.

Additionally, in some cases a user may specify a query that remains unsatisfiable even if all of the network semantics in our model are relaxed. This can occur if the user specifies components that do not exist, or that are in isolated, disconnected networks (for which no relaxation of the edge constraints will admit a path). In this case, our blocked path analysis fails, and TIROS falls back on other techniques to produce diagnostic information.

In Section 5 we show that in most cases, our analysis succeeds and produces an actionable blocked path.

4 Deriving Blocked Paths from Unsatisfiable Queries

We group VPC configuration semantics into three disjoint sets of constraints: $(U \cup N \cup H)$. Set U contains constraints that enforce user-configurable control-plane settings (such as a user-defined route or firewall rule), while set N contains non-configurable for user-visible network semantics (such as the local route restriction).

Set H contains elements of the constraints that are either not user-visible (such as internal implementation details) or that should never be relaxed (such as the reachability predicate or any other constraints defined by the user's query). For example, many of our constraints involve containment comparisons between CIDRs and bitvectors representing IP addresses. An individual CIDR comparison is encoded as a fresh literal represnting the truth value of the comparison, along with multiple clauses that enforce the comparison semantics. The intermediate clauses that enforce the comparison semantics are implementation details that we include in set H, ensuring they are not included in the blocking reasons.

When a query is unsatisfiable, we derive a blocked path and corresponding blocking reasons from a Maximal Satisfiable Subset and Minimal Correction Subset of $(U \cup N \cup H)$, with set H being treated as hard constraints that must not be included in the MCS.

If possible, we want to produce an MCS containing only configurable blocking reasons from U. This ensures that the resulting blocked path is actionable. If we directly compute the MCS of the full constraint set $U \cup N \cup H$, with both U and N as soft constraints, non-configurable constraints from N may be included in the MCS even in cases where there exists an MCS containing only constraints from U. On the other hand, we still want to be able to produce an MCS in the case where the non-configurable and hard constraints $(N \cup H)$ are, by themselves, unsatisfiable.

In Algorithm 1, we resolve this by breaking the computation of the MCS into two steps, initially computing an MCS of $N \cup H$, and only allowing constraints from N into the blocking reasons if MCS$(N \cup H)$ is non-empty.

When $N \cup H$ is satisfiable, Algorithm 1 produces a blocked path that only contains the configurable blocking reasons from U.

Algorithm 1 constructs two correction sets, $MCS_N \subseteq N$ and $MCS_U \subseteq U$, with $MCS_N \cup MCS_U$ a valid MCS of $(U \cup N \cup H)$. We then extract a path p from a satisfying assignment to the corresponding MSS $(U \cup N \cup H) \setminus (MCS_N \cup MCS_U)$. Finally, as shown below, we return either a complete or a partial blocked path, by associating blocking reasons from the MCS with nodes on that path.

Algorithm 1 relies on two helper methods, ExtractPath and BuildPath. ExtractPath retrieves the satisfying theory model (a sequence of edges) for the query reachability predicate from a satifiable formula, using the graph theory in the SMT-solver MonoSAT, and associates packet header assignments with each step of that path from the corresponding bitvector assignments. BuildPath maps the literals of the MCS to descriptive strings representing blocking reasons, and associates those strings with steps on the blocked path.

Algorithm 1 Blocked Path Analysis

```
1: function DeriveBlockedPath($U, N, H$)   ▷ Precondition: $U \cup N \cup H$ is UNSAT.
2:     if UNSAT($H$) then
3:         throw Error: No blocked path can be produced.
4:     else
5:         // Note: If $N \cup H$ is SAT, then $MCS_N = \emptyset$.
6:         $MCS_N \leftarrow$ computeMCS($N, H$)
7:         // Note: $(N \cup H) \setminus MCS_N$ is SAT; $MCS_U$ is well-defined.
8:         $MCS_U \leftarrow$ computeMCS($U, (N \cup H) \setminus MCS_N$)
9:         $p \leftarrow$ ExtractPath($(U \cup N \cup H) \setminus (MCS_N \cup MCS_U)$)
10:        return BuildPath($p, MCS_N, MCS_U$)
11:    end if
12: end function
```

We can see that $MCS_N \cup MCS_U$ meets the definition of a minimal correction set of $U \cup N \cup H$ by observing that:

$$\begin{aligned}
\text{SAT}((U \cup N \cup H \setminus (MCS_N)) \setminus (MCS_N \cup MCS_U)) &\quad \text{line 8} \\
\implies \text{SAT}((U \cup N \cup H) \setminus (MCS_N \cup MCS_U))) &\\
\forall c \in MCS_N, \text{UNSAT}((N \cup H) \setminus (MCS_N \setminus \{c\}) &\quad \text{line 6} \\
\forall c \in MCS_U, \text{UNSAT}((U \cup N \cup H) \setminus (MCS_U \setminus \{c\})) &\quad \text{line 8} \\
\implies \forall c \in (MCS_N \cup MCS_U), \text{UNSAT}((U \cup N \cup H) \setminus ((MCS_N \cup MCS_U) \setminus \{c\})) &
\end{aligned}$$

If $N \cup H$ is satisfiable, then MCS_N is empty and MCS_U, containing only configurable constraints, is an MCS of $(U \cup N \cup H)$. In this case, BuildPath constructs a complete blocked path consisting entirely of configurable blocking reasons.

If $N \cup H$ is unsatisfiable, then MCS_N is non-empty and $MCS_N \cup MCS_U$ contains at least one non-actionable constraints. In this case, the path p may behave unexpectedly and may not be realizable in a VPC configuration after adjustment. If MCS_N is non-empty, BuildPath forms the blocked path as above, but returns only the prefix of that blocked path up to and including the first edge or node associated with a non-actionable setting.

Above, we discussed the cases where $N \cup H$ is satisfiable or unsatisfiable. There is also a third possibility: The hard constraints H, representing the constraints enforcing the user's query or implementation details of our model, may by themselves be unsatisfiable. For example, H may be unsatisfiable if the user specifies a source and destination that are in separate, disconnected networks.

If H is unsatisfiable, Algorithm 1 fails, and is unable to produce even a partial blocked path. In this case, we fall back on other techniques to provide useful diagnostic information for users. In practice, the typical reason that H is unsatisfiable is that the source and destination are in disconnected VPCs (so the reachability constraint is unsatisfiable). We use a static analysis pass to identify this case and handle it separately in our service.

In the case that Algorithm 1 produces a complete (*resp.* partial) blocked path, the underlying MCS algorithm guarantees that the blocked path will have the fewest possible number of blocking reasons from among all complete (*resp.* partial) blocked paths. In general this blocked path is not unique.

In our implementation of Algorithm 1, the graph-based decision heuristic in MonoSAT will prioritize finding shortest-length paths in most cases, but does not guarantee that a shortest-length path is always found.

5 Evaluation

VPC Reachability Analyzer, a commercial offering available from AWS since December 2020, uses the blocked path analysis we have described to derive findings for queries between unreachable endpoints.

To demonstrate the practical impact of this blocked path analysis, we randomly selected 1000 unreachable queries processed by VPC Reachability Analyzer. We executed the blocked path analysis for those queries on an 'm5.24xlarge' EC2 instance using GNU Parallel [15], running Amazon Linux 2, using MonoSAT version 1.6.0.

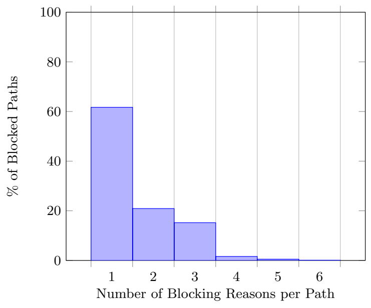


Fig. 3. Number of blocking reasons per blocked path (among the 63% of unreachable queries for which the blocked path analysis produced a complete blocked path). 97% percent of blocked paths have three or fewer blocking reasons; 60% have just a single blocking reason.

Excluding the time to complete the blocked path analysis, the average time required to initially determine satisfiability of the constraints was 2.1 seconds (P50: 1.7 s, P99: 7.4 s). The blocked path analysis was as fast or faster than the initial solving time, requiring 0.3 seconds on average (P50: 0.05 s, P99: 6.6 s).

As described in Section 4, in some cases, the blocked path analysis can produce only a partial path, or no results at all. Of those 1000 unreachable queries, 63.2% resulted in complete blocked paths, 7.4% resulted in partial blocked paths, and the remainder (29.4%) produced no analysis (in which case VPC Reachability Analyzer applies other techniques so that it can still provide useful diagnostics).[2]

As can be seen in Figure 3, most blocked paths have just one blocking reason, and 97% have at most three. This demonstrates that our analysis produces actionable, concise findings on real production data, a key requirement of a useful diagnosis service.

6 Conclusion

The blocked path analysis we have introduced provides key advantages over previous network diagnostic techniques. By showing users a blocked path from a source to a destination, we allows users the opportunity to refine their query such that their intended path is aligned with our analysis. Furthermore, showing all blocking reasons on a blocked path allows users to understand the VPC configuration adjustments necessary to realize a path for their query.

Our blocked path analysis is a fully static analysis (requiring no packets to be injected into the network), can be computed efficiently using standard techniques from the formal methods literature, and is now used successfully in production by VPC Reachability Analyzer.

References

1. Amazon Inspector. https://docs.aws.amazon.com/inspector/, accessed: December 2018
2. Backes, J., Bayless, S., Cook, B., Dodge, C., Gacek, A., Hu, A.J., Kahsai, T., Kocik, B., Kotelnikov, E., Kukovec, J., McLaughlin, S., Reed, J., Rungta, N., Sizemore, J., Stalzer, M., Srinivasan, P., Suboti, P., Varming, C., Whaley, B.: Reachability analysis for AWS-based networks. In: International Conference on Computer Aided Verification. pp. 231–241. Springer (2019)
3. Bayless, S., Bayless, N., Hoos, H.H., Hu, A.J.: SAT modulo monotonic theories. In: Proceedings of AAAI. pp. 3702–3709 (2015)
4. Felfernig, A., Schubert, M., Zehentner, C.: An efficient diagnosis algorithm for inconsistent constraint sets. Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AI EDAM **26**(1), 53 (2012)
5. Fogel, A., Fung, S., Pedrosa, L., Walraed-Sullivan, M., Govindan, R., Mahajan, R., Millstein, T.: A general approach to network configuration analysis. In: Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation. pp. 469–483. NSDI'15, USENIX Association, Berkeley, CA, USA (2015), http://dl.acm.org/citation.cfm?id=2789770.2789803
6. Jayaraman, K., Bjørner, N., Outhred, G., Kaufman, C.: Automated analysis and debugging of network connectivity policies. Microsoft Research pp. 1–11 (2014)

[2] Of the queries for which no blocked path analysis was performed, 80% were due to users specifying endpoints in disconnected VPCs. We perform a disconnected component analysis to identify this case. Others were due to users specifying resources they lack access to, or that we do not support.

7. Jazib Frahim, Omar Santos, A.O.: Cisco ASA All-in-One Firewall, IPS, and VPN Adaptive Security Appliance, 3rd edition. Cisco Press (2014)
8. Junker, U.: Preferred explanations and relaxations for over-constrained problems. In: AAAI-2004 (2004)
9. Koitz, R., Wotawa, F.: Sat-based abductive diagnosis. In: DX@ Safeprocess. pp. 167–176 (2015)
10. Li, C.M., Manya, F.: Maxsat, hard and soft constraints. Handbook of satisfiability **185**, 613–631 (2009)
11. Lynce, I., Marques-Silva, J.P.: On computing minimum unsatisfiable cores (2004)
12. Mahajan, R., Spring, N., Wetherall, D., Anderson, T.: User-level internet path diagnosis. ACM SIGOPS Operating Systems Review **37**(5), 106–119 (2003)
13. Mai, H., Khurshid, A., Agarwal, R., Caesar, M., Godfrey, B., King, S.T.: Debugging the data plane with anteater. In: Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, ON, Canada, August 15-19, 2011. pp. 290–301 (2011). https://doi.org/10.1145/2018436.2018470, http://doi.acm.org/10.1145/2018436.2018470
14. Marques-Silva, J., Heras, F., Janota, M., Previti, A., Belov, A.: On computing minimal correction subsets. In: Twenty-Third International Joint Conference on Artificial Intelligence. Citeseer (2013)
15. Tange, O.: GNU Parallel 2018. Ole Tange (Mar 2018). https://doi.org/10.5281/zenodo.1146014, https://doi.org/10.5281/zenodo.1146014
16. Tian, B., Zhang, X., Zhai, E., Liu, H.H., Ye, Q., Wang, C., Wu, X., Ji, Z., Sang, Y., Zhang, M., et al.: Safely and automatically updating in-network acl configurations with intent language. In: Proceedings of the ACM Special Interest Group on Data Communication, pp. 214–226 (2019)
17. Walter, R., Felfernig, A., Küchlin, W.: Constraint-based and sat-based diagnosis of automotive configuration problems. Journal of Intelligent Information Systems **49**(1), 87–118 (2017)