2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S) | 978-1-6654-3566-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/DSN-S52858.2021.00030

A Matter of Trust: Verification of Security and Performance for Network Platform as a Service

Tim Alberdingk Thijm*, Gary Atkinson†, Lalita Jagadeesan†, Marina Thottan†
*Princeton University
tthijm@cs.princeton.edu
†Nokia Bell Labs
{gary.atkinson, lalita.jagadeesan, marina.thottan}@nokia-bell-labs.com

***Abstract*—Network platform as a Service (NPaaS) is an emerging concept in the telecommunications industry, in which services are deployed across a multitude of communication service provider (CSP) networks. In addition to connectivity requirements, these services will have associated and varied security and performance requirements. While such requirements must be verified across multiple CSP networks, CSPs will be selective about the degree of network information and guarantees they are willing to provide to NPaaS providers, application service providers, and other CSPs. Furthermore, all such providers are likely to be wary about network information or guarantees provided by CSPs with whom they do not have trust relationships. In this short paper, we present some challenges facing NPaaS security and performance, and demonstrate through an example case study how recent formal network verification techniques and the open-source network verification toolset NV can be used as a framework to verify security and performance requirements in an NPaaS context.**

***Index Terms*—network verification, inter-CSP services, network platform as a service, NPaaS, security, performance, trust, NV**

I. Introduction

An emerging concept in the telecommunications industry consists of services that traverse a multitude of communication service provider (CSP) networks, where the specifics of the constituent CSP networks are hidden from these services. A network platform, itself a service, enables these inter-CSP network services to be created and seamlessly deployed. For example, the Twilio [1] platform as a service enables developers to rapidly incorporate inter-CSP messaging services into their applications, while Pulumi [2] enables developers to automatically deploy their applications across multiple cloud providers. This concept can be extended to encompass a broad range of service types, from gaming to video communication to storage, as depicted in Figure 1.

In addition to connectivity requirements, many such services will have significant security and performance expectations from customers/end users. A key challenge for any multi-service Network Platform as a Service (NPaaS) is thus to ensure that security and performance requirements specified by application developers are guaranteed to hold on the paths traversed across multiple CSP networks.

For example, a necessary security requirement for video communications services may be that, depending on the origin, destination, and/or end customer (e.g., a government), services never traverse a network in specified countries nor traverse network equipment by specified vendors. Conversely, the security of a storage service may require that only networks within specified countries are traversed (e.g., for General Data Protection Regulation (GDPR) [3] privacy reasons). Performance properties may also be required; for example, governing the latency of gaming services.

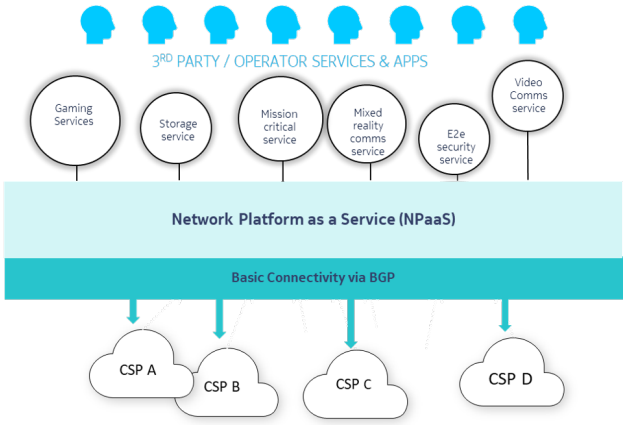

Fig. 1: Network Platform as a Service (NPaaS)

At the same time, CSPs will be selective about the degree of network information and guarantees they are willing to provide to NPaaS providers, application service providers, or other CSPs. Furthermore, for the transmission of sensitive services, all such providers are likely to be wary of network information and guarantees from CSPs with whom they do not have trust relationships. Thus, satisfying security and performance requirements for such services requires incorporating *on a per-service basis* that (a) each CSP retains control over the degree of network information and guarantees it provides to others and (b) information and/or guarantees from specified CSPs can be considered potentially untrustworthy.

A. The Border Gateway Protocol (BGP)

The defacto approach for establishing connectivity among multiple CSP networks is the Border Gateway Protocol (BGP) [4]. BGP is a distributed networking protocol in which network routers communicate available routes with their neighbors: each router selects the best route among the ones

*Work conducted when the author was at Nokia Bell Labs.

978-1-6654-3566-6/21/$31.00 ©2021 IEEE
DOI 10.1109/DSN-S52858.2021.00030

received from its neighbors. BGP has been shown to converge to a stable state [5] where all routers have selected the best routes available to them from neighbors.

Thus, a key challenge for NPaaS providers is to ensure that the different security and performance requirements of services are satisfied by the stable state BGP routing of the overall network. At the same time, NPaaS providers need to, *on a per-service basis* (a) enable CSPs to control the information and guarantees that they provide about their networks, and (b) allow application providers, CSPs, or the NPaaS itself to designate specified CSP networks as potentially untrustworthy.

Existing work on securing BGP [6], [7] has primarily focused on *coordinated* route authentication and authorization in the face of malicious attacks, whereas the kinds of security and performance requirements discussed above instead require *distributed* approaches that accommodate independent and potentially undisclosed routing concerns of CSP networks. Furthermore, as security and performance requirements will vary widely among services running on the same underlying network, our approaches must support such variations.

B. Formal Network Verification and Related Work

A promising approach for guaranteeing desired network requirements is control plane verification, in which a model of the network control plane and its desired properties are formally specified. Algorithms are used to check that the model is guaranteed to satisfy the intended properties, and produce a counter-example if the intended property can be violated. As the underlying models and associated algorithmic analysis are computationally intensive, control plane verification approaches are typically performed statically (off-line).

Control plane verification approaches that check for BGP-related properties include Batfish [8], Bagpipe [9], ARC [10], Minesweeper [11], ShapeShifter [12] and NV [13]. These approaches automatically verify that router configurations satisfy intended network properties of a given Autonomous System (AS)/CSP. ERA [14] includes control plane verification for cloud provider networks. However, these approaches do not focus on verification of security and performance properties across multiple CSPs, where CSPs can on a per-service basis control the information and guarantees they provide, as well as designate specified CSPs as untrustworthy.

C. Our Contributions

In this short paper, we present some challenges facing NPaaS from a security and performance perspective, and present a framework for static control plane verification of key types of security and performance requirements for services across multiple CSPs. Our framework enables specification of information hiding and trust relationships (or lack thereof) on a per-service basis, and can be used in conjunction with the approaches described above. We demonstrate the application of this framework through a proof-of-concept case study using NV [13], an intermediate language for representing network topologies and routing policies. NV's associated Satisfiability Modulo Theories (SMT) constraint solver, based on Minesweeper [11], encodes the stable state BGP routing of a given network representation, and performs automated verification of properties specified as assertions.

Our paper is organized as follows. Section II describes the key challenges for security and performance for NPaaS providers, application service providers, and CSPs. Section III contains our case study verifying proof-of-concept performance and security requirements on inter-CSP services using NV. We end with conclusions and future work in Section IV.

II. Security and Performance Requirements for inter-CSP services

As a motivating example, we consider the network depicted in Figure 2. This network consists of four CSP networks (A, B, C, D), interconnected through peering points (in green), where each node represents a router. We assume that there is a Network Platform as a Service (NPaaS) Provider that is providing services across these CSPs.

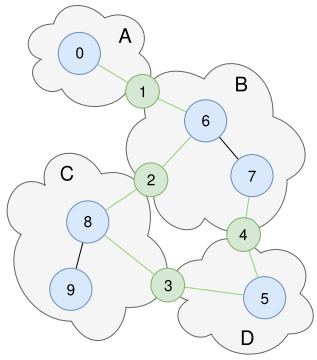

Fig. 2: An example inter-CSP network (numbers are node identifiers)

For simplicity, we assume that the BGP policies at each router are based on local preference and shortest-path; that is, each router selects the shortest route advertised by its neighbors, under its local preference policy. We wish to verify that — under its stable state routing — this inter-CSP network satisfies the kinds of security and performance requirements described in Section I on a per-service basis. Some representative requirements for services are:

- *Performance:* The route length from node 0 to node 5 is less than n hops. (We use route length as an approximation of latency).
- *Security - network avoidance:* Traffic from node 0 to node 5 will not traverse CSP C. (Here we assume that CSP C is in a country or contains network equipment by a vendor that is potentially not trusted).
- *Security - GDPR* All traffic originating in CSP networks A, B, or D, will fully remain in these networks.

III. Our Case Study

As described in Section I, we use the intermediate language NV and its associated toolset as a framework for verifying these types of requirements. NV represents control planes in a functional language syntactically similar to OCaml [15]. Using

a front-end tool such as Batfish [8], we can parse a set of router configurations into an NV program. For our case study, we begin by introducing a set of utility functions partially shown in Listing 1. These include:

- A simplified representation of a BGP route that incorporates standard elements of a BGP announcement: the local preference of a path (`lp`), the length of a path (`length`), and the node originating the announcement (`origin`).
- A transfer function, defining how nodes announce routes to their neighbors. It increments the length of the path to the destination.
- A merge function, defining how nodes receive routes from their neighbors. It compares two routes and selects the one with a greater local preference, or the one with a shorter path length if the local preferences are the same.
- A predicate `assertRoute`. It returns true if a stable state route exists that has a length of at most `len`.

```
type bgp = { lp: int; length: int; origin: tnode }
type attribute = option[bgp]

let incr bgp i = { bgp with length = bgp.length + i }

let transBgp edge x = match x with
| Some r -> Some (incr r 1)
| None -> None

let isBetter x y = match (x, y) with
| (None, _) -> false
| (_, None) -> true
| (Some b1, Some b2) -> (b1.lp > b2.lp) || (b1.lp =
↪  b2.lp && b1.length < b2.length)

let mergeBgp node x y = if isBetter x y then x else y

let assertRoute x len = match x with
| Some r -> r.length <= len
| None -> false
```

Listing 1: A set of BGP utilities `bgp-util.nv`

These utilities provide a sufficient set of definitions for us to model BGP in a variety of network representations.

A. Verification with Full Network Information

We first present verification of the performance and security requirements described in Section II, under the assumption that CSPs are willing to share their complete network topology information as in Figure 2. Section III-B will describe verification under information hiding and trust relationships.

```
include "bgp-util.nv"

let nodes = 10
let edges = { 0n=1n; 1n=6n; 6n=2n; 6n=7n; 8n=3n;
↪  3n=5n; 2n=8n; 8n=9n; 7n=4n; 4n=5n; }

let init node = match node with
  | 5n -> Some { lp = 100; length = 0; origin = 5n }
  | _ -> None

let sol = solution { init = init; trans = transBgp;
↪  merge = mergeBgp }
assert assertRoute (sol[0n]) 5 (* ✓ *)
assert assertRoute (sol[0n]) 3 (* X *)
```

Listing 2: An NV file representing Figure 2 network topology

We begin by showing in Listing 2 an NV representation of the entire inter-CSP network depicted in Figure 2. The listing describes the nodes and edges as shown in the figure (kn represents node k from the figure), and sets initial route information at each node: node 5 (our routing destination) has a route to itself with a local preference of 100. All other nodes initially have no route to node 5. We then instruct NV's verification engine to compute constraints on each node's stable state route (a *solution*) using the `transBgp` transfer and `mergeBgp` merge functions.

We now use assertions in NV to verify the properties described in Section II.

Performance: We provide assertions for NV to verify, where an assertion is a formula over the solution of a given node. Our assertions use the `assertRoute` predicate to check if the route from node 0 to node 5 (`sol[0n]`) is at most m hops. These assertions are checked by NV's SMT solver, which verifies that traffic from node 0 to node 5 does not traverse more than 5 hops in any stable state. If we attempt to verify that it does not traverse more than 3 hops, NV reports a counterexample[1].

Security — network avoidance: To capture whether a route travels through a node we wish to avoid, we extend our previous model to also track all nodes a route crosses through. It is now straightforward to describe an avoidance property `assertAvoided` as shown in Listing 3, where we check that the set of visited nodes does not include the given nodes 8 and 9. In this example, the NV verifier will confirm that traffic from node 0 to node 5 never traverses nodes 8 or 9, but that traffic from node 9 to node 5 does traverse node 8.

Security — GDPR: We next show how to verify a GDPR compliance (*i.e.*, data protection) property. Suppose domains A, B and D represent CSPs who must follow the GDPR, while domain C does not guarantee GDPR compliance. To verify that traffic originating in CSPs A, B or D always routes via GDPR nodes, we use a containment property `assertSubset`, as given in Listing 3. It specifies that we must visit only nodes in a specified set. We can then verify that traffic from node 0 to node 5 stays within nodes in CSPs A, B and D.

B. Verification with Guarantees and Trust Relationships

We now describe how our approach can be extended to allow information hiding and trust (or lack thereof), as discussed in Section II. We assume here that CSPs B and C are unwilling to share the details of their network topologies depicted in Figure 2: their internal networks are considered black boxes on which they merely provide guarantees. For example, CSP B may guarantee that the path length from nodes 1 to 2 (resp. nodes 1 to 4) never exceeds some stipulated length `L1to2` (resp. `L1to4`), as shown in Listing 4. CSP C will similarly do so for `L2to3`. We assume these stipulations are certificate-based authentications on the network identity; however, route length veracity can be tested only indirectly, *e.g.*, by sampling and measuring end-to-end latency.

[1]Counterexamples are solutions satisfying the *negation* of the given assertion (examples of such solutions omitted here to save space).

```
include "bgp-util.nv"

type attribute = option[(set[tnode], bgp)]

let transVia edge x = match x with
  | None -> None
  | Some (via, bgp) -> (match edge with
    | a~b -> Some (via[a := true], incr bgp 1)

let assertAvoided x avoided = match x with
  | Some (via, bgp) -> (via inter avoided = {})
  | None -> false

let assertSubset x contained = match x with
  | Some (via, bgp) -> (subset via contained)
  | None -> false

let sol = solution { init = init; trans = transVia;
↪  merge = mergeVia }
assert assertAvoided (sol[0n]) ({8n, 9n}) (* ✓ *)
assert assertAvoided (sol[9n]) ({8n}) (* X *)
assert assertSubset (sol[0n]) ({0n, 1n, 2n, 3n, 4n,
↪  5n, 6n, 7n}) (* ✓ *)
```

Listing 3: An NV file for verifying security assertions

```
type attribute = option[(set[int], bgp)]

let nodes = 6
let edges = { 0n=1n; 1n=2n; 1n=4n; 2n=3n; 3n=5n;
↪  4n=5n; }

symbolic L1to2 : int
symbolic L1to4 : int
symbolic L2to3 : int

require (L1to2 >= 2) && (L1to4 >= 2) && (L2to3 >= 2)
require (L1to4 <= 3)

let trans edge x = match x with
  | Some (via, bgp) -> (match edge with
    | 0~1 -> Some (via[CSPA := true], incr bgp 1)
    | 1~2 -> Some (via[CSPB := true], incr bgp L1to2)
    | 1~4 -> Some (via[CSPB := true], incr bgp L1to4)
    | 2~3 -> Some (via[CSPC := true], incr bgp L2to3)
    | 3~5 -> Some (via[CSPD := true], incr bgp 1)
    | 4~5 -> Some (via[CSPD := true], incr bgp 1)
    | 1~0 -> Some (via[CSPA := true], incr bgp 1)
    | 2~1 -> Some (via[CSPB := true], incr bgp L1to2)
    | 4~1 -> Some (via[CSPB := true], incr bgp L1to4)
    | 3~2 -> Some (via[CSPC := true], incr bgp L2to3)
    | 5~3 -> Some (via[CSPD := true], incr bgp 1)
    | 5~4 -> Some (via[CSPD := true], incr bgp 1))
  | None -> None

assert assertRoute (sol[0n]) 5
assert assertAvoided (sol[0n]) ({CSPC})
assert assertSubset (sol[0n]) ({CSPA, CSPB, CSPD})
```

Listing 4: An NV file abstracting the network in Fig. 2

Let us further assume that either the service provider, the NPaaS, or CSPs A, B, or D do not trust CSP C. Thus, they would like to verify that that the desired security properties will always be satisfied *no matter the actual value of* `L2to3`.

We use NV's `symbolic` capability, where we introduce symbolic variables with constraints specified by `require` clauses. We stipulate that all the path lengths `L*` must be ≥ 2 as they must traverse an internal node between peering points. NV reports that — given the specified network behaviour — node 0 will prefer routing via CSPs A, B and D over CSP C to node 5 as long as `L1to4` is ≤ 3. Thus, our framework supports the specification of trusted/untrusted relationships.

IV. Conclusions

In this paper, we have described some key challenges from a security and performance perspective facing services that traverse multiple CSP networks, and demonstrated how formal network verification can be used to assess guarantees for security and performance requirements. Such verification will be critical for application service developers to gain confidence in the use of a Network Platform as a Service. Our approach has the following benefits:

- It provides verification of security and performance guarantees for services that traverse multiple CSP networks.
- It enables CSPs to retain control of network information and guarantees provided to NPaaS, application service providers, and other CSPs.
- It allows providers to specify trust relationships (or lack thereof) with CSPs.

As our approach requires that CSPs need only provide abstract guarantees about their networks wrt to specific services, we believe it has the potential to scale to large networks and a wide variety of services. We envision that such an approach would be used in conjunction with existing verification tools such as those described in Section I-B: CSPs would independently verify using those tools that their networks and router configurations indeed satisfy the guarantees they promise to the NPaaS. Our approach would provide verification *across multiple CSPs* ensuring necessary service-level security and performance properties, while allowing and respecting information hiding, guarantees, and trust relationships.

In the future, we plan to extend this framework with additional BGP features, further degrees of trust, and additional security and performance properties.

References

[1] https://customers.twilio.com/208/uber/.
[2] https://www.pulumi.com/.
[3] https://gdpr-info.eu.
[4] https://tools.ietf.org/html/rfc1105.
[5] T. G. Griffin, F. B. Shepherd, and G. Wilfong, "The stable paths problem and interdomain routing," *IEEE/ACM Transactions on Networking*, vol. 10, no. 2, 2002.
[6] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-189.pdf.
[7] G. Huston, M. Rossi, and G. Armitage, "Securing bgp—a literature survey," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 2, 2011.
[8] Batfish: An open source network configuration analysis tool. https://www.batfish.org/.
[9] K. Weitz, D. Woos, E. Torlak, M. D. Ernst, A. Krishnamurthy, and Z. Tatlock, "Scalable verification of border gateway protocol configurations with an smt solver," in *ACM OOPSLA*, 2016.
[10] A. Gember-Jacobson, R. Viswanathan, A. Akella, and R. Mahajan, "Fast control plane analysis using an abstract representation," in *ACM SIGCOMM*, 2016.
[11] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," in *ACM SIGCOMM*, 2017.
[12] ——, "Abstract interpretation of distributed network control planes," in *ACM POPL*, 2019.
[13] N. Giannarakis, D. Loehr, R. Beckett, and D. Walker, "NV: An intermediate language for verification of network control planes," in *ACM PLDI*, 2020.
[14] S. K. Fayaz, T. Sharma, A. Fogel, R. Mahajan, T. Millstein, V. Sekar, and G. Varghese, "Efficient network reachability analysis using a succinct control plane representation," in *USENIX OSDI*, 2016.
[15] https://ocaml.org.