On using SMT-solvers for Modeling and Verifying Dynamic Network Emulators
(Work in Progress)

Erick Petersen\textsuperscript{1,2}, Jorge López\textsuperscript{1}, Natalia Kushik\textsuperscript{2}, Claude Poletti\textsuperscript{1}, and Djamal Zeghlache\textsuperscript{2}
\textsuperscript{1} Airbus Defence and Space, Élancourt, France
\textsuperscript{2} Télécom SudParis, Institut Polytechnique de Paris, Palaiseau, France

Abstract—A novel model-based approach to verify dynamic networks is proposed; the approach consists in formally describing the network topology and dynamic link parameters. A many sorted first order logic formula is constructed to check the model with respect to a set of properties. The network consistency is verified using an SMT-solver, and the formula is used for the run-time network verification when a given static network instance is implemented. The z3 solver is used for this purpose and corresponding preliminary experiments showcase the expressiveness and current limitations of the proposed approach.

Index Terms—Network emulator, Modeling, Verification, Many sorted first order logic formula, SMT-solver

I. INTRODUCTION

As dynamic networks progress rapidly, novel solutions emerge. Testing and evaluating network components in a well emulated environment is necessary to reduce risks and save time at the final deployment. We focus on dynamic networks whose parameters change, e.g., the bandwidth and delay of wireless links may change due to external interference. Depending on the desired network to emulate, the link parameters vary according to properties that are to be respected by the emulators (e.g., large delays on distant objects).

This paper is devoted to the problem of adequate emulation of such networks. We propose to rely on model checking and run-time verification techniques that under certain assumptions allow checking that the emulator is adequate for a given context and is designed correctly. The novelty of the proposed approach for the emulator design and verification relies on the use of Many Sorted First Order Logic (MSFOL) for describing the dynamic network characteristics. The latter allows to automatically validate if the network description is consistent, and after the network is implemented, to verify at run-time that it behaves as requested. Both tasks can be effectively implemented through the use of a Satisfiability Modulo Theories (SMT) solver. Note that existing network simulators and emulators (e.g., \textsuperscript{2}) provide a solid background, however the authors are not aware of any works related to model-based dynamic network verification. We also note that the network description for a network emulator is another challenging issue. Existing solutions can provide convenient interfaces for network description or allow specific scenario files (e.g., \textsuperscript{3}). However, such descriptions remain rather informal and do not facilitate further verification.

We propose to utilize a formal network description that contains both, the properties of the network topology as well as various network parameters. As mentioned before, such description is provided in terms of a MSFOL formula to be verified by an SMT-solver; in this paper, the z3 solver is used. Preliminary experiments showcase the flexibility provided by this approach. Likewise, we explore the performance for large network instances and discuss the current limitations of the formal verification for dynamic networks.

II. MODELING NETWORKS USING MSFOL

Dynamic networks. A static network is a computer network where each link has a set of parameters that do not change, for example bandwidth (capacity) or delay. Differently from static networks, the parameters of the links may change in dynamic networks (in the scope of our current work, we assume the network topology does not change); such change can be the consequence of the physical medium (e.g., in wireless / radio frequency networks) or due to logical changes (e.g., rate limiting the capacity of a given link). Static networks can be modeled as (directed) weighted graphs \((V, E, p_1, \ldots, p_k)\), where \(V\) is a set of nodes, \(E \subseteq V \times V\) is a set of directed edges, and \(p_i\) is a link parameter function \(p_i : E \to \mathbb{N}\), for \(i \in \{1, \ldots, k\}\); without loss of generality, we assume that the parameter functions map to non-negative integers (denoted by \(\mathbb{N}\)) or related values can be encoded with them. Similarly, dynamic networks can be modeled as such graphs, however, \(p_i\) maps an edge to a non-empty set of integer values, i.e., \(p_i : E \to 2^{\mathbb{N}} \setminus \emptyset\), where \(2^{\mathbb{N}}\) denotes the power-set of \(\mathbb{N}\). As an example, consider the dynamic network depicted in Fig.\textsuperscript{1} and its model \(\mathcal{N} = (V, E, p_1(e), p_2(e))\), where:

\[
V = \{1, 2, 3, 4\} \\
E = \{(1, 2), (2, 1), (1, 3), (3, 1), (1, 4), (4, 1), (2, 4), (4, 2), (3, 4)(4, 3)\}
\]

\[
p_1(e) = b((s, d)) = \begin{cases} 
4, 5, 6, \text{ if } d = 2 \\
2, 3, 4, \text{ otherwise }
\end{cases}
\]

\[
p_2(e) = d((s, d)) = \begin{cases} 
1, 2, \text{ if } d = 2 \\
9, 10, \text{ otherwise }
\end{cases}
\]

Semantically, this model represents a dynamic network in which the link’s available bandwidth can vary according to the function \(b\) (for bandwidth), and the link’s delay can vary according to the function \(d\) (for delay). Note that a dynamic network snapshot, at a given time instance, is a static network, and thus, we use both terms interchangeably.

Satisfiability Modulo Theories (SMT). SMT is concerned with the satisfiability of formulas with respect to some back-
ground theory [6]. SMT usually works with a typed version of first order logic, particularly MSFOL. The syntax for MSFOL formulas is standard (see [6]), however, in our work, we use \( x : \sigma \) to indicate that \( x \) is of sort (type) \( \sigma \) (e.g., in quantified formulas \( \forall x : \phi \)). We denote as \( f : \sigma_1 \times \sigma_2 \times \ldots \times \sigma_n \rightarrow \sigma \) that function \( f \) is declared of sort \( \sigma \).

**Modeling dynamic networks using MSFOL formulas.**

It is desirable that our model uses sorts of theories which are available in SMT solvers and ideally those which are decidable. For that reason, (finite) sets are encoded as objects of array sort (denoted \( \mathcal{A} \)), e.g., the set \( V = \{1, 2, 3, 4\} \) is encoded as \( \phi_V = (V : \mathcal{A}_{\mathbb{Z}, \mathbb{Z}}) \land (V[1] = 1) \land (V[2] = 2) \land (V[3] = 3) \land (V[4] = 4) \land (|V| = 4) \). For convenience, we specify that the sort of \( V \) is an array whose indices and values are integers; likewise, we also include the cardinality of \( V \). Directed edges are nothing more than records (tuples with sorts), particularly, pairs of integers. Thus, following our encoding, \( \phi_{\mathcal{E}} = (E : \mathcal{A}_{\mathbb{Z}, \mathbb{Z}, \mathbb{Z}}) \land (E[1] = 1, 2) \land \ldots \land (E[10] = 4, 3) \land (|E| = 10) \).

In order to model the bandwidth of an edge \( (b(e)) \) and the delay of an edge \( (d(e)) \) according to the given functions, we use the formula \( \phi_p = (b : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}) \land (d : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}) \land \forall x : \mathbb{Z} (((x \geq 1) \land (x \leq |E|)) \implies (((d(E[x]) = 2) \implies ((b(E[x]) \geq 4) \land (b(E[x]) \leq 6) \land (d(E[x]) \geq 1) \land (d(E[x]) \leq 2))) \implies (((d(E[x]) \neq 2) \implies ((b(E[x]) \geq 2) \land (b(E[x]) \leq 4) \land (d(E[x]) \geq 9) \land (d(E[x]) \leq 10))))).

The complete model is the conjunction of the three previous formulas, i.e., \( \phi_N = \phi_V \land \phi_E \land \phi_p \). This simple example showcases very little of the flexibility provided by describing dynamic networks as MSFOL formulas, however very complex models can be described with such formalism.

**III. Model checking & Run-time verification for dynamic network emulators.**

**Model checking.** Once the model of a dynamic network is built, a set of properties of interest can be verified. For example, checking that the values of the parameters are not negative can be easily expressed with the MSFOL formula:

\[
\pi_\geq = \forall x : \mathbb{Z} (((x \geq 1) \land (x \leq |E|)) \implies ((b(e) \geq 0) \land (d(e) \geq 0))).
\]

Likewise, checking that all nodes have at least one incoming and one outgoing edge can be easily expressed as: \( \pi_{\uparrow\downarrow} = \forall x : \mathbb{Z} (((x \geq 1) \land (x \leq |V|)) \implies \exists y, z : \mathbb{Z} (((y \neq z) \land (y \geq 1) \land (y \leq |E|)) \land (z \geq 1) \land (z \leq |E|)) \land (dst(E[y]) = V[x]) \land (dst(E[z]) = V[x]))).\)

Having the properties to be checked, \( \pi C \) as a conjunction of them (in our previous example \( \pi C = \pi_\geq \land \pi_{\uparrow\downarrow} \)), and the model representing the dynamic network \( \phi_N \), the model checking process is quite straightforward. First, we check that both \( \phi_N \) and \( \pi C \) are satisfiable; otherwise, either the model (whose verification can be performed beforehand as well) or the properties have inconsistencies. Further, if the formula \( \phi_N \land \pi C \) is satisfiable, then we conclude that the properties are held for the model \( \phi_N \). If the formula is not satisfiable it implies that there does not exist a satisfiable interpretation for both formulas at the same time, i.e., that there is a conflict between \( \phi_N \) and \( \pi C \).

As an example, consider the formula \( \phi_N \) associated with the dynamic network shown in Figure 1 additionally, consider the property \( \pi_{\downarrow\uparrow} = \forall x : \mathbb{Z} (((x \geq 1) \land (x \leq |E|)) \implies ((d(e) > 2)) \) stating that the delay must be at least equal to three. \( \phi_N \land \pi_{\downarrow\uparrow} \) is not satisfiable, as the model states that the delay of certain edges is either one or two, and thus, there is no satisfiable interpretation for the conjunction of formulas (even if there are satisfiable interpretations for each of them).

**Run-time verification.** Once a dynamic network emulator is implemented, it can be continuously verified that the produced static instances do not violate the description of \( \mathcal{N} \). In this case, we consider a conformance relation \( \preceq \) which is similar to a reduction, i.e., a static network \( \mathcal{N}_S \preceq \mathcal{N} \) if the topology \( (V, E) \) of \( \mathcal{N}_S \) is exactly the same of that one of \( \mathcal{N} \), and for each \( i \in \{1, \ldots, k\} \) \( p_{i\mathcal{N}_S}(e) \in p_{i\mathcal{N}}(e) \).

In order to verify this relation \( \preceq \) at run-time, the behavior of the emulator can be monitored for checking that each link \( (v_a, v_b) \in V \times V \) is implemented correctly, and that each value of the \( i \)-th parameter belongs to the set \( p_{i\mathcal{N}_S}(v_a, v_b) \). We assume that the points of observation in this case can be placed at each node \( v \in V \) and each link \( (v_a, v_b) \), correspondingly.

We propose to iteratively check the implementation of each link verifying that the value of \( p_{i\mathcal{N}_S}(v_a, v_b) \) does not violate the description of the dynamic network \( \mathcal{N} \). The latter can be performed through a call to an SMT-solver and whenever the corresponding formula is not satisfiable, an alert is produced. When alerting, relevant information for debugging can be shown, i.e., the link \( (v_1, v_2) \) itself as well as the \( i \)-th parameter which was wrongly assigned when implementing the static instance \( \mathcal{N}_S \). The corresponding procedure is shown in Algorithm 1.

**Preliminary experiments.** In order to showcase the expressiveness of the proposed method, and to assess its limitations, a preliminary experimental evaluation has been conducted. From the model checking point of view, the properties \( \pi_\geq \) and \( \pi_{\uparrow\downarrow} \), as previously described, have been tested alongside the properties listed in Table 1.
the bandwidth \((in, out, \text{and } bw)\) are not described in order to avoid overloading the formulas. The properties have been coded in SMT-LIB and the z3 solver has been used to check their satisfiability in randomly generated graphs. The running time for networks ranging from one to 30 nodes is shown in Fig. 2, note that large instances can require long verification time. We then conclude that such method can be rather inefficient for run-time verification when the whole network is verified at once. For a large network of 50 nodes, the verification time per edge is around 0.069s. For this work in progress, this process has been manually performed; an automated process is envisioned for future work. Note that all z3 code, graphs and properties can be found in our repository [7].

IV. CONCLUSION

We discussed the use of SMT-solvers for verifying dynamic network emulators described as many sorted first order logic formulas whose consistency can be checked using one of such solvers; in our work, we used z3. Once a network is implemented, its static instance can be verified at run-time. We discussed a possible verification solution and proposed an algorithm for checking that the instance does not violate the initial formula. Experimental results confirmed that the verification is the most efficient in an incremental link-by-link way. For future work, we consider studying formula optimization techniques for decreasing the verification time. We plan to consider more dynamic network parameters and the dependencies between them. We expect that such dependencies can be also verified using the proposed approach.

Algorithm 1: Run-time verification of a dynamic network emulator

\[\text{input} : \text{A formula } \phi_N \text{ specifying the dynamic network } N \]
\[\text{output: Alert for a static network violating } \phi \text{; a link and a parameter } \text{`responsible’} \text{ for the violation} \]

\[\text{while true do} \]
\[\text{get the dynamic network instance (static network) } N_S; \]
\[\text{foreach } \text{link } (v_a, v_b) \text{ do} \]
\[\phi \equiv \exists x : Z. ((x \geq 1) \land (x \leq |E|)) \land (E[x] = (v_a, v_b)); \]
\[\text{if } \phi_N \land \phi \text{ is UNSAT then} \]
\[\text{alert}(v_a, v_b); \]
\[\text{foreach } i \in \{1, \ldots, k\} \text{ do} \]
\[\phi_i = \exists x : Z. ((x \geq 1) \land (x \leq |E|)) \land (E[x] = (i, v_a, v_b)); \]
\[\text{if } \phi_N \land \phi_i \text{ is UNSAT then} \]
\[\text{alert}(i, v_a, v_b); \]

\[\text{end while} \]

The network topology cannot be full mesh

\[\pi_M = (|E|/(|V| \times (|V| - 1)) \neq 1 \]

The incoming bandwidth of all nodes is strictly less than \(C\) times the outgoing bandwidth capacity

\[\pi_{IO} = (in : \text{from } \text{nodes} \to Z) \land (out : \text{to } Z) \land (\forall i : \text{Z} \land (i \geq 1) \land \forall j : (i \leq |V|) \Rightarrow (in(V[i]) < C \times out(V[j]))) \]

The sum of the bandwidth of all network cannot exceed the threshold \(B\)

\[\pi_{+} = (bw : \text{Z} \to Z) \land (bw(E) \leq B) \]

TABLE I

| Network properties of interest |
|--------------------------------|
| Description | Formula |
| The links are symmetric (for any link a return link exists) | \[\pi_\text{L} = \forall x : Z. ((x \geq 1) \land (x \leq |E|)) \Rightarrow \exists y : Z. ((y \geq 1) \land (y \leq |E|) \land (\text{src}(E[x]) = \text{dst}(E[y])) \land (\text{dst}(E[x]) = \text{src}(E[y])))\] |
| The edges in the edge array are composed of nodes in the node array | \[\pi_\text{E} = \forall i : Z. (((i \geq 1) \land (i \leq |E|)) \Rightarrow (\exists j, k : Z. ((\text{src}(E[i]) = V[j]) \land (\text{dst}(E[i]) = V[k])))\] |
| The delay of all links is always less or equal to a given constant \(D\) | \[\pi_\text{D} = \forall i : Z. (((i \geq 1) \land (i \leq |E|)) \Rightarrow (d(E[i]) \leq D))\] |
| The bandwidth of all links is greater or equal to the threshold \(B\) | \[\pi_\text{B} = \forall i : Z. (((i \geq 1) \land (i \leq |E|)) \Rightarrow (b(E[i]) \geq B))\] |
| The network topology density is at least \(\delta\) | \[\pi_\text{D} = (|E|/(|V| \times (|V| - 1))) \geq \delta\] |
| The sum of the bandwidth of all network cannot exceed the threshold \(B\) | \[\pi_\text{+} = (bw : \text{Z} \to Z) \land (bw(E) \leq B)\] |

Fig. 2. Time evaluation of emulator verification

REFERENCES

[1] J. Lai, J. Tian, K. Zhang, Z. Yang, and D. Jiang, “Network emulation as a service (neas): Towards a cloud-based network emulation platform,” Mobile Networks and Applications, pp. 1–15, 2020.
[2] R. Priyadarshi, B. Gupta, and A. Anurag, “Deployment techniques in wireless sensor networks: a survey, classification, challenges, and future research issues,” The Journal of Supercomputing, pp. 1–41, 2020.
[3] A. Varga, “Discrete event simulation system,” in Proc. of the European Simulation Multiconference (ESM’2001), 2001, pp. 1–7.
[4] G. F. Riley and T. R. Henderson, “The ns-3 network simulator,” in Modeling and tools for network simulation, 2010, pp. 15–34.
[5] L. De Moura and N. Bjørner, “Z3: An efficient smt solver,” in International conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2008, pp. 337–340.
[6] C. Barrett and C. Tinelli, “Satisfiability modulo theories,” in Handbook of Model Checking. Springer, 2018, pp. 305–343.
[7] E. Petersen, J. López, N. Kushik, C. Poletti, and D. Zeghlache, “SMT verification of network emulators,” https://github.com/prisen/SMETEmVerif, 2020.