Some Complexity Results for Stateful Network Verification

Yaron Velner[1], Kalev Alpernas[1], Aurojit Panda[2], Alexander Rabinovich[1], Mooly Sagiv[1], Scott Shenker[2], and Sharon Shoham[3]

[1] Tel Aviv University, Israel
[2] University of California, Berkeley
[3] The Academic College of Tel Aviv Yaffo, Israel

Abstract. In modern networks, forwarding of packets often depends on the history of previously transmitted traffic. Such networks contain *stateful* middleboxes, whose forwarding behavior depends on a mutable internal state. Firewalls and load balancers are typical examples of stateful middleboxes.
This paper addresses the complexity of verifying safety properties, such as isolation, in networks with finite-state middleboxes. Unfortunately, we show that even in the absence of forwarding loops, reasoning about such networks is undecidable due to interactions between middleboxes connected by unbounded ordered channels. We therefore abstract away channel ordering. This abstraction is sound for safety, and makes the problem decidable. Specifically, we show that safety checking is EXPSPACE-complete in the number of hosts and middleboxes in the network. We further identify two useful subclasses of finite-state middleboxes which admit better complexities. The simplest class includes, e.g., firewalls and permits polynomial-time verification. The second class includes, e.g., cache servers and learning switches, and makes the safety problem coNP-complete.
Finally, we implement a tool for verifying the correctness of stateful networks.

1 Introduction

Modern computer networks are extremely complex, leading to many bugs and vulnerabilities which affect our daily life. Therefore, network verification is an increasingly important topic addressed by the programming languages and networking communities (e.g., see [17,8,15,16,14,30,21,13]). Previous network verification tools leverage a simple network forwarding model which renders the datapath *immutable*; *i.e.,* normal packets going through the network do not change its forwarding behavior, and the control plane explicitly alters the forwarding state at relatively slow time scales. Thus, invariants can be verified before each control-plane initiated change and these invariants will be enforced until the next such change. While the notion of an immutable datapath supported by an assemblage of routers makes verification tractable, it does not reflect reality. Modern enterprise networks are comprised of roughly $2/3$ routers [4] and $1/3$ *middleboxes* [31]. A simple example of a middlebox is a stateful firewall which permits traffic from untrusted hosts only after they have received a message from a trusted

[4] In this paper we do not distinguish between routers and switches, since they obey similar forwarding models.

host. Middleboxes — such as firewalls, WAN optimizers, transcoders, proxies, load-balancers, intrusion detection systems (IDS) and the like — are the most common way to insert new functionality in the network datapath, and are commonly used to improve network performance and security. While useful, middleboxes are a common source of errors in the network [25], with middleboxes being responsible for over 40% of all major incidents in networks.

This paper addresses the problem of verifying safety of networks with middleboxes, referred to as *stateful* networks. From a verification perspective, it is possible to view a middlebox as a procedure with local mutable state which is atomically changed every time a packet is transmitted. The local state determines the forwarding behavior.[5] Thus, the problem of network verification amounts to verifying the correctness of a specialized distributed system where each of the middleboxes operates atomically and the order of packet arrivals is arbitrary.

We model such a network as a finite undirected graph with two types of nodes: (i) hosts which can send packets, (ii) middleboxes which react to packet arrivals and forward modified packets. Each node in the network has a fixed number of ports, connected by network edges.

Real middleboxes are generally complex software programs implemented in several 100s of thousands of lines of code. We follow [24,23] in assuming that we are provided with middlebox models in the form of *finite-state transducers*. In our experience one can naturally model the behaviour of most middleboxes this way. For every incoming packet, the transducer uses the packet header and the local state to compute the forwarding behavior (output) and to update state for future packets. The transducer can be non-deterministic to allow modelling of middleboxes like load-balancers whose behavior depends not just on state, but also on a random number source. We symbolically represent the local state of each middlebox by a fixed set of relations on finite elements, each with a fixed arity.

The Verification Problem We define network safety by means of avoiding "bad" middlebox states (e.g., states from which a middlebox forwards a packet in a way that violates a network policy). Given a set of bad middlebox states, we are interested in showing that for all packet scenarios the bad states cannot be reached. This problem is hard since the number of packets is unbounded and the states of one middlebox can affect another via transmitted packets.

1.1 What is decidable about middlebox verification

In Sec. 3, we prove that for general stateful networks the verification problem is undecidable. This result relies on the observation that packet histories can be used to count, similarly to results in model checking of infinite ordered communication channels [7]. One may believe that undecidability arises from the presence of forwarding loops in the network which are usually avoided in real networks. However, we show that the verification problem is undecidable even for networks without forwarding loops.

[5] Switches are a degenerate case of middleboxes, whose state is constant and hence their forwarding behavior does not change over time.

In order to obtain decidability, we introduce an abstract semantics of networks where the order of packet processing on each channel (connecting two middleboxes or a middlebox and a host) is arbitrary, rather than FIFO. Thus, middlebox inputs are multisets of packets which can be processed in any order. This abstraction is *conservative*, i.e., whenever we verify that the network does not reach a bad state, it is indeed the case. However, the verification may fail even in correct networks. Since packets are atomically processed, we note that network designers can impose ordering even in this abstract model by sending acknowledgments for received packets. This is useful when enforcing authentication.

In fact, this abstraction closely corresponds to assumptions made by network engineers: since packets in modern networks can traverse multiple paths, be buffered, or be chosen for more complex analysis, network software cannot assume that packets sent from a source to a server are received by a server in order. Network protocols therefore commonly build on TCP, a protocol which uses acknowledgments and other mechanisms to ensure that servers receive packets in order. Since packet ordering is enforced by causality (by sending acknowledgments) and by software on the receiving end, rather than by the network semantics, correctness of such networks typically does not rely on the order of packet processing. Therefore we can successfully verify a majority of network applications despite our abstraction.

1.2 Complexity of Stateful Verification

In Sec. 6, we show that the problem of network verification when assuming a nondeterministic order of packet processing is complete for exponential space, i.e., it is decidable, and in the worst case, the decision procedure can take exponential space in terms of hosts and middleboxes. This is proved by showing that the network safety problem is equivalent to the coverability problem of Petri nets, which is known to be EXPSPACE-complete [26].

Since the problem is complete, it is impossible to improve this upper-bound without further assumptions. Therefore, we also consider limited cases of middleboxes permitting more efficient verification procedures, as shown in Fig. 1. We identify four classes of middleboxes with increasing expressive power and verification complexity: (i) *stateless* middleboxes whose forwarding behavior is constant over time, (ii) *increasing* middleboxes whose forwarding behavior increases over time, (iii) *progressing* middleboxes whose forwarding behavior stabilizes after some fixed time, alternatively, the transition relation of the transducer does not include cycles besides self-cycles, and (iv) *arbitrary* middleboxes without any restriction. For example, NATs, Switches and simple ACL-based firewalls are stateless; hole-punching stateful firewalls are increasing; and learning-switches and cache-proxies are progressing and not increasing.

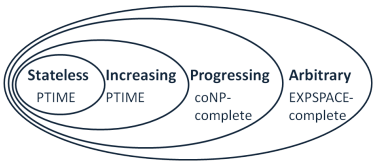


Fig. 1: Middlebox hierarchy.

For stateless and increasing middleboxes, we prove that any packet which arrives once can arrive any number of times, leading to a polynomial-time verification algorithm, using dynamic programming. We note that efficient near linear-time algorithms for stateless verification are known (e.g., see [16]). Our result generalizes these results to increasing networks and is in line with the recent work in [12,19].

For progressing middleboxes, we show that verification is coNP-complete. The main insight is that if a bad state is reachable then there exists a small (polynomial) input scenario leading to a bad state. This means that tools like SAT solvers which are frequently used for verification can be used to verify large networks in many cases but it also means that we cannot hope for a general efficient solution unless P=NP.

Finally, we note that unlike the known results in stateless networks, the absence of forwarding loops does not improve the upper bound, i.e., we show that our lower bounds also hold for networks without forwarding loops.

Packet Space Assumption Previous works in stateless verification [15,13] assume that packet headers have n-bits, simulating realistic packet headers which can be large in practice. This makes the complexity of checking safety of stateless networks PSPACE-hard. Our model avoids packet space explosion by only supporting three fields: source, destination, and packet tags. We make this simplification since our work primarily focuses on middlebox policies (rather than routing). As demonstrated in Sec. 5.1, middlebox policies are commonly specified in terms of the source and destination hosts of a packet and the network port (service) being accessed. For example, at the application level, firewalls may decide how to handle a packet according to a small set of application types (e.g., skype, ssh, etc.). Source, destination and packet tag are thus sufficient for reasoning about safety with respect to these policies. This simplification is also supported by recent works (e.g. [16]) which suggest that in practice the forwarding behavior depends only on a small set of bits.

Lossless Channels Previous works on infinite ordered communication channels have introduced *lossy channel systems* [1] as an abstraction of ordered communication that recovers decidability. Lossy channel systems allow messages to be lost in transit, making the reachability problem decidable, but with a non-elementary lower bound on time complexity. In our model, packets cannot be lost. On the other hand, the order of packets arrival becomes nondeterministic. With this abstraction, we manage to obtain elementary time complexity for verification.

Initial Experience We implemented a tool which accepts symbolic representations of middleboxes and a network configuration and verifies safety. For increasing (and stateless) networks, the tool generates a Datalog program and a query which holds iff a bad state is reachable. Then, the query is evaluated using existing Datalog engines [22].

For arbitrary networks (and for progressing networks), the tool generates a petri-net and a coverability property which holds iff the network reaches a bad state. To verify the coverability property we use LOLA [28] — a Petri-Net model checker.

Main Results The main contributions of the paper are: (i) We define a conservative abstraction of networks in which packets can be processed out of order, and show that

the safety problem of stateful networks becomes decidable, but EXPSPACE-complete. (ii) We identify classes of networks, characterized by the forwarding behaviors of their middleboxes, which admit better complexity results (PTIME and coNP). We demonstrate that these classes capture real-world middleboxes. The upper bounds are made more realistic by stating them in terms of a symbolic representation of middleboxes. (iii) We present initial empirical results using Petri nets and Datalog engines to verify safety of networks.

1.3 Outline of the rest of this Paper

The rest of the paper is organized as follows. Sec. 2 presents a formal model of stateful networks, Sec. 3 defines the verification problem and proves undecidability, Sec. 4 defines an abstract network semantics and proves decidability, Sec. 5 provides a classification of middleboxes, Sec. 6 and Sec. 7 show upper and lower bounds (respectively) complexity results. We finish with some case studies (Sec. 8) and a discussion of related work (Sec. 9).

2 A Formal Model for Stateful Networks

In this section, we present a formal model of networks with stateful middleboxes.

A *network* N is a finite undirected graph of *hosts* and *middleboxes*, equipped with a *packet domain*. Formally, $\mathsf{N} = (H \dot\cup M, E, P)$, where H is a finite set of *hosts*, M is a finite set of *middleboxes*, $E \subseteq \{\{u, v\} \mid u, v \in H \dot\cup M\}$ is the set of (undirected) edges and P is a set of packets. A *host* $h \in H$ consists of a unique id and a set of packets $h_P \subseteq P$ that it can send.

Packets. In real networks, a packet consists of a *packet header* and a *payload*. The packet header contains a source and destination host ids and additional arbitrary stream of control bits. The payload is the content of the packet and may consist of any arbitrary sequence of bits. In particular, the set of packets need not be finite. In this work, P is a set of *abstract packets*. An abstract packet $p \in P$ consists of a header only in the form of a triple (s, d, t), where $s, d \in H$ are the source and destination hosts (respectively) and t is a *packet tag* that ranges over a finite domain T. Intuitively, T stands for an abstract set of services or security policies. Therefore, $P = H \times H \times T$, making it a finite set. Middlebox behavior in our model is defined with respect to abstract packets and is oblivious of the underlying concrete packets.

2.1 Stateful Middleboxes

A *middlebox* $m \in M$ in a network N has a set of *ports* Pr, which consists of all the adjacent edges of m in the network N, and a *forwarding transducer* F.

The forwarding transducer of a middlebox is a tuple $F = (\Sigma, \Gamma, Q_m, q_m^0, \delta_m)$ where $\Sigma = P \times \mathsf{Pr}$ is the input alphabet in which each input letter consists of a packet and an input port, $\Gamma = 2^\Sigma$ is the output alphabet describing (possibly empty) sets of packets over the different ports, Q_m is a possibly infinite set of states, $q_m^0 \in Q_m$ is

the initial state, and $\delta_m : Q_m \times \Sigma \to 2^{\Gamma \times Q_m}$ is the transition relation. Note that the alphabet Σ is finite (since abstract packets are considered). We extend δ_m to sequences $h \in (P \times \mathsf{Pr})^*$ in the natural way: $\delta_m(q, \epsilon) = \{(\epsilon, q)\}$ and $\delta_m(q, h \cdot (p, pr)) = \{(\gamma_i \cdot o', q') \mid \exists q_i \in Q_m . (\gamma_i, q_i) \in \delta_m(q, h) \wedge (o', q') \in \delta_m(q_i, (p, pr))\}$. The language of a state $q \in Q_m$ is $L(q) = \{(h, \gamma) \in (P \times \mathsf{Pr})^* \times (P \times \mathsf{Pr})^* \mid (\gamma, q') \in \delta_m(q, h)\}$. The language of F, denoted $L(F)$, is the language of q_m^0. We also define the set of *histories* leading to $q \in Q_m$ as $h(q) = \{h \in (P \times \mathsf{Pr})^* \mid (\gamma, q) \in \delta_m(q_m^0, h)\}$.

If F is deterministic, i.e., $|\delta_m(q, (p, pr))| \leq 1$, then every history leads to at most one state and output, in which case F defines a possibly partial *forwarding function* $\mathsf{f} : (P \times \mathsf{Pr})^* \times (P \times \mathsf{Pr}) \to 2^{P \times \mathsf{Pr}}$ where $\mathsf{f}(h, (p, pr)) = o$ for the (unique) output o such that $(h \cdot (p, pr), \gamma \cdot o) \in L(F)$. f defines the (possibly empty) set of output packets (paired with output ports) that m will send to its neighbors following every history h of packets that m received in the past and input packet p arriving on input port pr. If F is nondeterministic, a *forwarding relation* is defined in a similar way.

Note that every forwarding function f can be defined by an infinite-state deterministic transducer: Q_m will include a state for every possible history, with ϵ as the initial state. The transition relation δ_m will map a state and an input packet to the set of output packets as defined by f, and will change the state by appending the packet to the history.

Finite-state middleboxes Arbitrary middlebox functionality, defined via infinite-state transducers, makes middleboxes Turing-complete, and hence impossible to analyze. To make the analysis tractable, we focus on abstract middleboxes, whose forwarding behavior is defined by *finite-state* transducers. Nondeterminsm can then be used to overapproximate the behavior of a concrete, possibly infinite-state, middlebox via a finite-state abstract middlebox, allowing a sound abstraction w.r.t. safety. Note that when nondeterministic transducers are considered, the correspondence between packet histories and transducer states no longer holds, as a single history might lead to multiple states.

In the sequel, unless explicitly stated otherwise, we consider abstract middelboxes. We identify a middlebox with its forwarding relation and the transducer that implements it, and use m to denote each of them.

Symbolic representation of middleboxes We use a symbolic representation of finite-state middleboxes, where a state of m is described by the valuation of a finite set of relations $R_1, \ldots, R_k$ defined over finite elements (e.g., packet header fields). The transition relation δ_m is also described symbolically using (nondeterministic) update operations of the relations and output. Technically, we use guarded commands, where guards are Boolean expressions over *relation membership predicates* of the form $\overline{e}$ **in** R and element equalities $e_1 = e_2$. Each e_i is either a constant or a variable that refers to packet fields. Commands are of the form: (i) *insert* tuple $\overline{e}$ to relation R, and (ii) *remove* tuple $\overline{e}$ from relation R. (iii) *output* set of tuples.

Example 1. Fig. 2a contains a symbolic representation of a hole-punching Firewall which uses a unary relation `trusted`. It assumes that port 1 connects hosts inside a private organization to the firewall and that port 2 connects public hosts. By default, messages from public hosts are considered untrusted and are dropped. `trusted` stores public hosts that become trusted once they receive a packet from private hosts.

input(src, dst, tag, prt) :
  $prt = 1 \Rightarrow$
    // hosts within organization
    `trusted`.**insert** dst ;
    output $\{(src, dst, tag, 2)\}$
  $prt = 2 \wedge src$ **in** `trusted` $\Rightarrow$
    // trusted hosts outside organization
    output $\{(src, dst, tag, 1)\}$
  $prt = 2 \wedge \neg(src$ **in** `trusted`$) \Rightarrow$
    output $\emptyset$ // untrusted hosts

(a) A hole-punching firewall.

input(src, dst, tag, prt) :
  $prt = 1 \wedge (dst, src, tag)$ **in** `cache` $\Rightarrow$
    // previously stored response
    output $\{(prxy.host, src, tag, 1)\}$
  $prt = 1 \Rightarrow$
    // new request
    output $\{(prxy.host, dst, tag, 2)\}$
  $prt = 2 \Rightarrow$
    // response to a request
    `cache`.**insert**(src, dst, tag) ;
    output$\{(prxy.host, dst, tag, 2)\}$

(b) A Proxy.

Fig. 2: Symbolic representation of middleboxes.

Fig. 2b contains a simplified, nondeterminitic, version of a Proxy server (or cache server). A proxy stores copies of documents (packet payloads) that passed through it. Subsequent requests for those documents are provided by the proxy, rather than being forwarded. Our modelling abstracts away the packet payloads and keeps only their types. Consequently we use nondeterminism to also account for different requests with the same type. The internal relation `cache` stores responses for packet types.

2.2 Concrete (FIFO) network semantics.

The semantics of a network is given by a transition system defined over a set of configurations. In order to define the semantics we first need to define the notion of *channels* which capture the transmission of packets in the network. Formally, each (undirected) edge $\{u, v\} \in E$ in the network induces two directed *channels*: (u, v) and (v, u). The channel (v, u) is the *ingress channel* of u, as well as the *egress channel* of v. It consists of the sequence of packets that were sent from v to u and were not yet received by u (and similarly for the channel (u, v)). The capacity of channels is unbounded, that is, the sequence of packets may be arbitrarily long.

Configurations and runs. A *configuration* of a network consists of the content of each channel and the state of every middlebox. The *initial configuration* of a network consists of empty channels and initial states for all middleboxes. A configuration c_2 is a *successor* of configuration c_1 if it can be obtained by either: (i) some host h sending a packet $p \in h_P$ to a neighbor, thus appending p to the corresponding channel; or (ii) some middlebox m processing a packet p from the head of one of its ingress channels, changing its state to q' and appending output o to its egress channels if $(o, q') \in \delta_m(q, (p, pr))$ (where q is the current state of m and pr is the port associated with the ingress channel). This model corresponds to asynchronous networks with non-deterministic event order.

A *run of a network from configuration* c_0 is a sequence of configurations $c_0, c_1, c_2, \ldots$ such that c_{i+1} is a successor configuration of c_i. A *run* is a run from the initial configuration. The set of *reachable configurations from a configuration* c_i is the set of all configurations that reside on a run from c_i. The set of *reachable configurations* of a networks is the set of reachable configurations from the initial configuration.

3 Verification of Safety Properties in Stateful Networks.

In this section we define the *safety* verification problem in stateful networks, as well as the special case of *isolation*. We prove their undecidability w.r.t. the FIFO semantics.

To describe safety properties, we augment middleboxes with a special *abort state* that is reached whenever $\delta_m(q, (p, pr)) = \emptyset$, i.e., the forwarding behavior is undefined (not to be confused with the case where $(\emptyset, q') \in \delta_m(q, (p, pr))$ for some $q' \in Q_m$). This lets middleboxes function as "monitors" for safety properties. If $\delta_m(q, (p, pr)) = \emptyset$, and $h \in h(q)$, we say that m *aborts* on $h \cdot (p, pr)$ (and every extension thereof). Similarly, we augment the symbolic representation with an *abort* command.

We define *abort configurations* as network configurations where at least one middlebox is in an abort state.

Safety. The input to the *safety problem* consists of a network N (that possibly contains property middleboxes). The output is True if no abort configuration is reachable in N, and False otherwise.

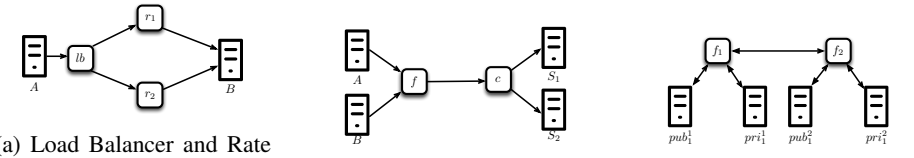
Isolation. An important example of a safety property is isolation. In the *isolation problem*, the input is a network N, a set of hosts $H_i \subseteq H$ and a forbidden set of packets $P_i \subseteq P$. The output is True if there is no run of N in which a host from H_i receives a packet from P_i, and False otherwise. The isolation problem can be formulated as a safety problem by introducing an *isolation middlebox* m_{h_i} for every host $h_i \in H_i$. The role of m_{h_i} is to monitor all traffic to h_i, and abort if a forbidden packet $p \in P_i$ arrives. All other packets are forwarded to h_i. Clearly, isolation holds if and only if the resulting network is safe.

Example 2. Fig. 3 shows several examples of interesting middlebox topologies for verification. In all of the topologies shown we want to verify a variant of the isolation property. In Fig. 3a we want to verify that A, a host, cannot send more than a fixed number of packets to B. Here r_1 and r_2 are rate limiters, *i.e.,* they count the number of packets they have seen going from one host to the other, and lb is a load balancer that evenly spreads packets from A along both paths (to minimize the load on any one path). In Fig. 3b we want to ensure that host A cannot access data that originates in S_1, but should be allowed to access data from S_2, where f is a firewall and c is a proxy (cache) server. Finally in Fig. 3c we show a multi-tenant datacenter (e.g., Amazon EC2), where many independent tenants insert rules into firewalls (f_1 and f_2) and we want to ensure that the overall behavior of these rules is correct. For example, we would like to ensure that pri_1^1 cannot communicate with pri_2^1, and pub_2^1 communicates with pri_1^1 only if pri_1^1 initiates the connection.

3.1 Undecidability of Safety w.r.t. the FIFO Semantics

We prove undecidability of the safety problem by showing that (the specific example of) checking isolation w.r.t. the FIFO semantics is undecidable, even when the network does not have forwarding loops.

It is well known that an automaton with an ordered channel of messages (also known as a *channel machine*) can simulate a Turing machine. This can be used to show that the



(a) Load Balancer and Rate Limiter

(b) Firewall and Proxy

(c) Multi-tenant data center

Fig. 3: Interesting network topologies for verification.

isolation problem over ordered channels is undecidable in the presence of forwarding loops: a forwarding loop allows a packet to traverse the network and reach the same middlebox any number of times. Therefore, it allows one middlebox in the network to simulate a channel machine by having all packets rerouted to it. However, it turns out that forwarding loops are not the root cause for undecidability. In this work, we prove that the isolation problem is still undecidable even in the absence of forwarding loops.

To formally define forwarding loops, we augment every packet sent by a host with a unique packet id (e.g., the host id combined with a time stamp). Middlebox forwarding is oblivious to this augmentation: forwarding functions do not depend on the packet id, nor do they change it. We say that a network has a forwarding loop if there is a run in which a packet with the same packet id is received by the same middelbox twice (i.e., a run in which a packet that originates from a middlebox is received by the same middelbox again, possibly after modifications).

We now prove the undecidability result.

Theorem 1. *The isolation problem under the FIFO network semantics is undecidable even for networks with finite-state middleboxes and without forwarding loops.*

Proof. Unbounded channels can simulate unbounded counter values (e.g., the number of packets in the channel is the counter value, and if the value of the counter is zero, then the channel holds only a packet with dedicated zero symbol). The idea is to simulate a two counter machine with a middlebox with two ports (channels), however, we need to overcome two obstacles. The first obstacle is that a middlebox cannot explicitly choose the port from which it receives the next packet. The second obstacle is that we want to avoid forwarding loops, and thus a naive approach where the middlebox sends the counter value packets in a loopback port will not work. We can overcome the first obstacle by adding a special sink state (which is safe), and if the next packet arrives from an undesirable port, then the middlebox goes to the safe sink state and stays there forever (and safety is not violated in such undesirable scenario). To overcome the second obstacle we replace every loopback port by a *repeater* middlebox and a repeater host.

4 Abstract Network Semantics

In this section we define an abstract network semantics, called the *unordered semantics*, which recovers decidability of the safety problem.

In the concrete (FIFO) network semantics channels are ordered. In an ordered channel, if a packet p_1 precedes a packet p_2 in an ingress channel of some middlebox, then the middlebox will receive packet p_1 before it receives packet p_2. We abstract this semantics by an *unordered network semantics*, where the channels are unordered, i.e., there is no restriction on the order in which a middlebox receives packets from its ingress channel. In this case, the sequence of pending packets in a channel can be abstracted by a multiset of packets. Namely, the only relevant information is how many occurrences each packet has in the channel. The definitions of configurations and runs w.r.t. the unordered semantics are adapted accordingly.

Remark 1. Unordered channels over-approximate ordered channels, and hence every run w.r.t. the FIFO network semantics is also a run w.r.t. the unordered semantics. Therefore, if safety holds w.r.t. the unordered semantics, then it also holds for the FIFO semantics. In this sense, the unordered semantics is a sound abstraction of the FIFO semantics w.r.t. safety. *Lossy channel semantics* is another overapproimation of the FIFO network semantics considered in the literature. We note that the unordered semantics also over-approximates the lossy semantics w.r.t. safety, as any violating run w.r.t. the lossy semantics can be simulated by a run w.r.t. the unordered semantics where "lost" packets are starved until the violation occurs.

The abstraction using unordered channels can introduce false alarms, as demonstrated by the next example which presents a network that violates isolation with respect to the unordered semantics, but satisfies isolation with respect to the FIFO semantics. Still, in many cases, the abstraction is precise enough to enable verification. In particular, in Lemma 9 we show that for an important class of networks, the two semantics coincide w.r.t. safety.

Example 3. Consider a network with two hosts (h_1 and h_2), each connected to an authentication middlebox (m_1 and m_2 respectively). The authentication middleboxes are connected to each other as well. Each authentication middlebox forwards all packets from a host only if the first packet seen from that host is an authentication key (k_1 and k_2 for m_1 and m_2 respectively), otherwise it drops all packets from that host. We would like to verify isolation between h_1 and h_2.

A possible scenario for unrodered reachability is: (i) h_1 sends k_1 and then sends k_2; (ii) m_1 receives first k_1 and then k_2 (and forwards both packets). (iii) m_2 receives k_2 before it receives k_1 (i.e., the order on the channel between m_1 and m_2 was not maintained). m_2 forwards k_2 to h_2 and isolation is violated.

On the other hand, if all channels are FIFO, then if h_1 first sends k_i, then m_{3-i} will never forward any packet from h_1 (for $i = 1, 2$). Hence, isolation always holds under the FIFO semantic.

4.1 Decidability of Safety w.r.t. the Unordered Semantics

In this section, we give some intuition on the decidability of safety in unordered networks (formal arguments with complexity bounds are provided by Theorem 4).

Monotone transition systems and decidability of unordered networks. In the unordered semantics, the network forms a special case of *monotone transition systems*. We define a partial order $\leq$ between network configurations such that $c_1 \leq c_2$ if the middlebox states in c_1 and c_2 are the same and c_2 has at least the same packets (for every packet type) in every channel. The network is a monotone in the sense that for every run from c_1 there is a corresponding run from any bigger configuration c_2. Indeed, more packets over a channel only increases the number of possible scenarios. The classical results of Abdulla et al. [2] and Finkel et al. [11] prove that in monotone transition systems a backward reachability algorithm always terminates and thus, the safety problem is decidable.

5 Classification of Stateful Middleboxes

Encouraged by the decidability of safety w.r.t. the unordered semantics, we are now interested in investigating its complexity. As a first step, in this section, we identify three special classes of forwarding behaviors of middleboxes within the class of arbitrary middleboxes. Namely, stateless, increasing, and progressing middleboxes. We show that these classes capture the behaviors of real world middleboxes. The classes naturally extend to classes of networks: a network is stateless (respectively, increasing, progressing or arbitrary) if all of its middleboxes are. As we show in Sec. 6, each of these classes results in a different complexity of the safety problem.

Stateless middlebox. A middlebox m is *stateless* if it can be implemented as a transducer with a single state, i.e., its forwarding behavior does not depend on its history.

Increasing middlebox. A middlebox m is *increasing* if its forwarding relation is monotonically increasing w.r.t. its history, where histories are ordered by the *subsequence* relation, denoted by $\sqsubseteq$. Formally, a middlebox m is increasing if for every two histories $h_1, h_2 \in (P \times \mathsf{Pr})^*$: if $h_1 \sqsubseteq h_2$, then for every packet p and port pr, if $(h_1 \cdot (p, pr), \gamma_1 \cdot o_1) \in L_m$ then either m aborts on $h_2 \cdot (p, pr)$ or there is $\gamma_2 \cdot o_2$ s.t. $(h_2 \cdot (p, pr), \gamma_2 \cdot o_2) \in L_m$ and $o_1 \subseteq o_2$, where L_m is the language of m's transducer.

Progressing middlebox. In order to define progressing middleboxes, we define an equivalence relation between middlebox states based on their forwarding behavior. States q, q' are equivalent, denoted $q_1 \approx q_2$, if $L(q_1) = L(q_2)$. A middlebox m is *progressing* if it can be implemented by a transducer in which whenever the state is changed into a non-equivalent state, it will never return to an equivalent state. Formally, if $(o', q') \in \delta_m(q, (p, pr))$ and $q' \not\approx q$ (where q, q' are reachable states of m) then for any history $h \in (P \times \mathsf{Pr})^*$, if $(\gamma'', q'') \in \delta_m(q', h)$ then $q'' \not\approx q$.

Finite-state progressing middleboxes have the following useful property:

Lemma 1. *Every finite-state progressing middlebox has an implementation as a finite-state transducer whose underlying state graph has a tree structure, except for, possibly, self-loops.*

The next lemma summarizes the hierarchy of the classes (as illustrated by Figure 1).

Lemma 2.
- *Any stateless middlebox is also increasing.*
- *Any increasing middlebox is also progressing.*

Syntactic characterization of middlebox classes. The classes of middleboxes defined above can be characterized via syntactic restrictions on their symbolic representation.

A middlebox representation is *syntactically stateless* if its representation does not use any insert or remove command on any relation. A middlebox representation is *syntactically increasing* if its representation does not use the remove command on any relation, and does not include any insert command under guards that include negated membership predicates. A middlebox representation is *syntactically progressing* if its representation does not use the remove command on any relation.

Lemma 3. *A middlebox is stateless (respectively increasing, progressing) if and only if it has a stateless (respectively increasing, progressing) representation.*

5.1 Examples

In this subsection, we introduce several middleboxes, each of which resides in one of the classes of the hierarchy presented above.

ACL switches. An *ACL switch* has a fixed access control list (ACL) that indicates which packets it should forward and which packets it should discard. Typically the rules in the list refer to the port number or to hosts that are allowed to use a certain service. As such, the forwarding policy of an ACL switch is based only on the source host and/or ingress port of the current packet, and does not depend on previous packets. Hence, an ACL switch can be implemented by a stateless middlebox.

Hole-punching firewalls. A *hole-punching firewall* is described in Example 1. As the set of trusted hosts depends on the history of the middlebox, a hole punching firewall cannot be captured by a stateless middlebox. (Formally, the same packet is handled differently when it follows different histories.) On the other hand, it is increasing. If for a certain history a host is trusted, then any additional packets (in the past or in the future) will not make it untrusted.

Learning switch. A *learning switch* dynamically learns the topology of the network and constructs a routing table accordingly. Initially, the routing table of the switch is empty. For every host h the switch remembers the first port from which a packet with source h has arrived. When a packet arrives, if the port of the destination host is known, then the packet is forwarded to that port; otherwise, the packet is forwarded to all connected ports excluding the input-port.

A learning switch is a progressing middlebox. Intuitively, after the middlebox's forwarding function has changed to incorporate the destination port for a certain host h, it will never revert to a state in which it has to flood a packet destined for h. A learning switch is however, not an increasing middlebox, as packets destined for a host whose location is not known are initially flooded, but after location of the host is learned, a single copy of all subsequent packets are sent.

Proxy server. The *Proxy server* as described in Example 1 is an increasing middlebox. After it had stored a response, it nondeterministically replies with the stored response, or sends the request to the server again. However, in a concrete network model that does not abstract away the packet payload, a proxy is a progressing middlebox. Once a

new request is responded by a proxy the forwarding behavior changes as it takes into account the new response, and it never returns to the previous forwarding behavior (as it does not "forget" the response). However, succh a proxy is *not* an increasing middlebox: while it behaves in a monotonically increasing manner over its request port, it behaves in a monotonically decreasing manner over the response port.

Round-robin load balancer. A *load balancer* is a device that distributes network traffic across a number of servers. In its simplest implementation, a round-robing balancer with n out-ports (each connected to a server) forwards the i-th packet it receives to out-port $i \pmod n$. Round-robin load balancers are not progressing middlebxes, as the same forwarding function repeats after every cycle of n packets.

Remark 2. In practice, middleboxes behavior can also be affected by timeouts and session termination. For example, in a firewall, a trusted host may become untrusted when a session terminates (which makes the firewall behavior no longer increasing). Similarly, a cached content of a cache server expires after a certain period of time (which violates progress). In this work, we do not model timeouts and session termination. In many practical cases, such as firewalls, resets can only prevent packets from being forwarded and therefore restrict reachability, thus not causing safety violations.

6 Complexity Lower Bounds of Safety w.r.t. the Unordered Semantics

When considering the unordered network semantics, the safety problem becomes decidable for networks with finite-state middleboxes. In the following two sections, we analyze its complexity. In this section we present complexity lower bounds for the safety problem. In Sec. 7 we present upper corresponding upper bounds, resulting in tight bounds. We also present algorithms with matching complexity.

6.1 Unordered Safety in progressing networks is coNP-hard.

Lemma 4. *The isolation problem w.r.t. the unordered network semantics for a progressing network is coNP-hard.*

Proof. (sketch.) We show a reduction from the Hamiltonian Path problem to the reachability problem, which is the complement of the isolation problem. Recall that the isolation middlebox is stateless, hence it does not change the class of the input network. We can therefore deduce that the same lower bounds also hold for the more general safety problem.

We use *flood-once* middleboxes that upon receiving a packet increment its tag and flood the new packet. All following packets that arrive at the middlebox are discarded. These flood-once middleboxes are finite-state progressing middleboxes.

We introduce a single flood-once middlebox for every vertex in the graph and connect them in accordance with the edges in the graph. In addition, we create two hosts h_{source} and h_{target} and connect them to the middleboxes representing the source and target in the graph. The packet tags 'count' the length of the path. Thus, a Hamiltonian Path corresponds to a packet with the tag n arriving at the destination host.

The following lemma shows that a similar result can be obtained using more "standard" middleboxes, namely, stateless middleboxes and learning switches.

Lemma 5. *The isolation problem w.r.t. the unordered network semantics for a network where each middlebox is either stateless or a learning switch is coNP-hard.*

Proof. (sketch.) The proof follows from Lemma 4 and a construction of a network gadget that simulates the "flood once" middlebox. Recall that the first time a learning switch receives a packet with a fresh destination host, then it floods the packet to all of its neighbors. We design a protocol with three stateless middleboxes A, B and C and one learning switch, such that after the flood, A responds with a packet that marks it as the destination of all future learning switch forwarding. In addition, only one of the middleboxes B or C (the middlebox that responds first) forwards the packet to the next "flood once" gadget (of a neighbor vertex). In any future packets that will arrive to the gadget, the learning switch will direct the packets only to A, and A will discard them. Hence, the specification of the "flood once" middlebox follows. The protocol has four phases and the stateless middleboxes keep track over the phases according to the packet header. That is, the protocol phase is encoded in the source and destination host of the packet header.

6.2 Unordered Safety in arbitrary networks is EXPSPACE-hard.

The lower bound is obtained by a reduction from the *VASS control state reachability problem*. We first present the problem and its known complexity results. A *vector addition system with states (VASS)* is a weighted directed graph $(V, E, v_0, w : E \to \mathbb{Z}^k)$, where V is a finite set of vertices, $E \subseteq V \times V$ is a set of directed edges, v_0 is the initial vertex, and w is a weight function that assigns a k-dimensional weight vector to every edge. A (finite) path π in the directed graph is *valid* if it begins in v_0 and every prefix of π has a non-negative sum of weights in every dimension.

The *VASS control state reachability problem* problem gets as input a VASS and a *reachability set* $R \subseteq V$, and checks whether there exists a valid path in the VASS to (at least) one vertex in R.

Lemma 6 ([9]). *The VASS control state reachability problem is EXPSPACE-complete. Moreover, it is EXPSPACE-hard even when the coefficients of every vector in the image of the weight function are bounded by* ± 1. *and even when every vector has at most one non-zero dimension.*

Lemma 7. *The safety problem in stateful networks w.r.t. the unordered semantics is EXPSPACE-hard.*

Proof. (sketch.) We rely on Lemma 6 and show how to simulate a VASS with vectors of the format that is mentioned in the lemma. For the simulation we use one host and one middlebox, and the packet tag set is $\{1, \ldots, k\}$, where k is the dimension of the VASS. The middlebox has two ports: one connected to the host and the other is a loopback port. A packet of type i that arrives from the host corresponds to a vector with value $+1$ in dimension i, and a packet of type i that arrives from the loopback port corresponds

to a vector with value -1 in dimension i. The states of the middlebox correspond to the states of the VASS. If at a certain state the middlebox receives a packet that corresponds to an invalid transition vector (i.e., the transition is not allowed in this state), then the middlebox goes to a (safe) sink state (and stays there forever). Otherwise, if the packet arrives from the host then it forwards it to the loopback port, and if the packet arrived from the loopback port then it discards the packet. Intuitively, the number of packets of type i over the unbounded loopback channel corresponds to the value of dimension i in the VASS. The forbidden state is the target reachability state in the VASS. Hence, safety is violated if and only if the VASS target state is reachable.

7 Complexity Upper Bounds of Safety w.r.t. the Unordered Semantics

In this section we provides complexity upper bounds for the safety problem of stateful networks w.r.t. the unordered semantics of networks. Our complexity analysis considers symbolic representations of middleboxes (which might be exponentially more succint than explicit-state representations). The obtained upper bounds match the lower bounds from Sec. 6 (hence, the bounds are tight).

Remark 3. The complexity upper bounds we present are under the assumption that all relations used to define middlebox states may have at most polynomial number of elements (polynomial in the size of the network and the size of the middlebox representation). To enforce this limitation we assume that the arity of relations is constant. In all of our examples we use relations with arity at most three, and since abstract packets have only three attributes, we believe that most applications will use relations with small arity.

7.1 Equivalence of FIFO and Unoredered Semantics w.r.t Safety in Increasing Networks

In this section we show that for increasing networks, safety w.r.t. the unordered semantics and the FIFO semantics coincide. As such, the polynomial upper bound shown in Sec. 7.2 applies to both.

Recall that in general, safety w.r.t. the FIFO semantics and the unordered semantics do not coincide. However, the following lemmas show that for increasing networks (with either finite-state or infinite-state middleboxes) they must coincide, making the abstraction precise for such networks. Intuitively, this is because in increasing networks if a packet p reaches a middlebox m once, then it can reach m again, thus enabling the simulation of unordered channles with ordered ones.

Lemma 8. *Let* N *be an increasing network. For every middlebox* m*, packet* p *and port* pr*, if there exists a run* r *of* N *from the initial configuration in the FIFO semantics such that in the last step* m *receives* p *from* pr*, then from* any *configuration there exists a run, in the FIFO semantics, that ends in a step in which* m *receives* p *from* pr *(or in abort).*

$StateData := \{m \mapsto InitialRelationValues(m) \mid m \in M\}$
$PacketData := \{m \mapsto NeighborHostPackets(m) \mid m \in M\}$
while fixed-point not reached
  foreach $m \in M$, $(p, pr) \in PacketData(m)$
    let $q = GetState(StateData(m))$
    if $\delta_m(q, (p, pr)) = \emptyset$ **then return** violation // abort state reached
    let $(q', o) \in \delta_m(q, (p, pr))$
    $StateData := AddData(m, q')$
    $PacketData := AddPacketsToNeighbors(m, o)$
return safe

Fig. 4: Safety checking of increasing networks.

Proof. (sketch.) The formal proof is by induction on the length of the run from the initial configuration. The idea is that for every middlebox m', if m' sends a packet in this run due to history h, then we can inductively construct a run, from any configuration, which results in a history h' for m' such that $h \sqsubseteq h'$, hence m' can send the same packet.

Lemma 9. *Let* N *be an increasing network. Then the output of the safety problem in* N *w.r.t. the FIFO semantics and the unordered semantics is identical.*

Proof. (sketch.) It suffices to prove that for every violating run w.r.t. the unordered semantics there is a violating run w.r.t. the FIFO semantics (the converse is clear). The idea is to simulate an unordered run r by a FIFO run as follows. Whenever a middlebox handles a packet out of order in r, in the FIFO run it will first process all preceding packets. The problem is that while doing so, it might consume a packet that is needed later in the simulation of r. If this happens, we will use Lemma 8 to construct another unordered run in which the packet is produced again.

7.2 Unordered Safety of Increasing Networks is in PTIME

In this section, we show that safety of syntactically increasing networks is in PTIME.

Fig. 4 presents a polynomial algorithm for determining safety of a syntactically increasing network. The algorithm performs a fixed-point computation of the set of all tuples present in midddlebox relations in reachable middlebox states, as well as the set of all different packets transmitted in the network. For every middelbox $m \in M$, the algorithm maintains the following sets:

- $StateData(m)$: a set of pairs of the form $(R, \overline{d})$ where R is a relation of m, and $\overline{d}$ is a tuple in the domain of R, indicating that there is a run in which $\overline{d} \in R$.
- $PacketData(m)$: a set of pairs of the form (p, pr), where p is a packet and pr is a port of m, indicating that p can reach m from port pr.

$StateData(m)$ is initialized to reflect the initial values of all middlebox relations. $PacketData(m)$ is initialized to include the packets h_P that can be sent from neighbor hosts $h \in H$. As long as a fixed-point is not reached, the algorithm iterates over all middleboxes and the packets in their packet data. For each middlebox m

and $(p, pr) \in PacketData(m)$, m is run over (p, pr) from the state q in which every relation R contains all the tuples $\overline{d}$ such that $(R, \overline{d}) \in StateData(m)$. The sets $StateData(m)$ and $PacketData(m')$ for every neighbor m' of m, are updated to reflect the discovery of more elements in the relations (more reachable states), and more packets that can be transmitted in the network.

As the algorithm only adds reachable states and packets, its running time is polynomial and bounded by $|M|(|P||\mathsf{Pr}|\sum|R_i|)^2$.

The correctness of the algorithm relies on the a variation of Lemma 8 for the unordered semantics, which ensures that in increasing unordered networks, if a packet is sent in some run, then a run where it is sent exists from any other configuration. The same goes for elements that are added to relations. Based on this, we prove that in every iteration the data structure of the algorithm under-approximates $StateData$ and $PacketData$, and that when fixed-point occurs the data structure over-approximate $StateData$ and $PacketData$:

Lemma 10.
- *For every iteration of the algorithm, there is a run r, such that if $(p, pr) \in PacketData(m)$, then in r there is a step in which p arrived to m from port pr, and if $(R, \overline{d}) \in StateData(m)$, then there is a step in which $\overline{d}$ was added to R.*
- *When the algorithm reaches a fixed-point, if $(p, pr) \in PacketData$ (respectively., $(R, \overline{d} \in StateData)$, then there is no run in which m receives p from port pr (resp., $\overline{d}$ is added to R).*

Proof. (sketch.) The first part is shown by induction on the number of iterations, using the property of runs described above. The second part is immediate from the fixed-point definition.

Lemma 10 implies correctness of the algorithm, hence:

Theorem 2. *The safety problem of syntactically increasing networks w.r.t. the unordered semantics is in PTIME.*

Remark 4. If n-tag packet headers are allowed, i.e. $P = H \times H \times T_1 \ldots \times T_n$, then $|P|$ is no longer polynomial in the network representation, damaging the complexity analysis of the algorithm. In fact, in this case the safety problem w.r.t. the unordered semantics becomes PSPACE-hard even for stateless middleboxes: intuitively, n-tags allow a middlebox to maintain the state of n automata in the packet header, supporting a reduction from the emptiness problem of the intersection of n automata, which is PSPACE-hard [18].

Proof. The proof is by reduction from the emptiness problem of the intersection of n automata, which is PSPACE-hard [18]. Intuitively, n-tags allow the middlebox to maintain the state of every automaton in the packet header. An additional tag corresponds to an input symbol. When the middlebox receives a packet it simulates one step of every automaton, and forwards to a loopback port two packets with the same states tag, each with different input symbol (w.l.o.g the input alphabet is of size two). The middlebox aborts if all automata are in accepting states. Note that the middlebox is stateless: all the states of the automata are maintained in the packet.

7.3 Unordered Safety of Progressing Networks is in coNP

We prove coNP-membership of the safety problem in syntactically progressing networks by proving that there exists a witness run for safety violation if and only if there exists a "short" witness run, where a witness run for safety violation is a run from the initial configuration in which at least one middlebox reaches an abort state.

The proof considers the *network states* that arise in a run. A *network state* captures the states of all middleboxes (not to be confused with a network configuration, which also includes the content of every channel). Formally, let N be a network whose middleboxes are defined symbolically via (in total) n relations, namely $R_1, \ldots, R_n$. Then the *network state* consists of the values of $(R_1, \ldots, R_n)$.

In order to construct a shorter run, we wish to bound both the number of different network states in a run and the number of steps in which a run stays in the same state. The former is bounded due to the progress of the network: once the state of some middlebox changes along a run, it will not change back to the previous state. The latter is more tricky. To provide a bound, we wish to analyze the packets that "affect" the run. We define the notion of *active packets*. The active packets are a superset of the packets that actually affect the run.

Active packets. Let r be a finite run of a network. We say that a packet p is *active* in step i of r, if it resides in the ingress channel of some middlebox m and it is processed (i.e., received by m) in some future step of r. A packet is *inactive*, if it is pending in the ingress channel of m until the end of the run. The next lemmas show that only a few active packets are needed to reach a certain state in the network.

Lemma 11. *Let r be a run in which the network state changes exactly k times, and the different states are $s_1, s_2, \ldots, s_k$ (in this order). Then for every prefix r_{s_i} of r that ends in a state s_i, there is an extension e_{s_i} to r_{s_i} such that: (i) e_{s_i} visits the network states $s_i, \ldots, s_k$; (ii) e_{s_i} has at most $k - i$ active packets in every step; and (iii) the number of active packets in e_{s_i} may decrease only after a change in the network state.*

Proof. (sketch.) The proof is by induction over $|r| - |r_{s_i}|$. The base case ($r = r_{s_i}$) is trivial. Suppose that the claim is true for a prefix r' of length $\ell + 1$, then for a shorter prefix r'' we observe the packet that is processed in step $\ell + 1$. If it does not contribute to the extension of r', then it might as well be inactive, and we can skip its processing. If it does contribute, then either it changes the state and thus we can mark it as active (but the extension will have only $k - i - 1$ active packets, so in total we have $k - i$ active packets), or it generates active packets for the extension of r'. In the last case, we mark the packet as active, in the next step the number of active packets does not increase (hence, it is still at most $k - i$).

Lemma 12. *Let r be a run in which the network state changes exactly k times, the different states are $s_1, s_2, \ldots, s_k$ (in this order). Then there exists a run r' such that: (i) r' visits the network states $s_1, s_2, \ldots, s_k$; and (ii) r' stays in state s_i at most $(k - i)^2|P||M|$ steps.*

Proof. (sketch.) We assign a unique id to every active packet (which is independent of the mutable packet header). Intuitively, in the shortest run an active packet will not visit

the same middlebox twice unless (i) the network state was changed; or (ii) it generates additional active packet. Since at most $k-i$ packets are active, at most additional $k-i$ active packets are generated, there are at most $|P|$ different headers, and there are $|M|$ middleboxes, then we obtain $(k-i)^2|P||M|$ bound on the number of steps.

The next lemma shows that there is a short witness for reachability of a state in progressing networks.

Lemma 13. *Let* N *be a network with syntactically progressing middleboxes that have the relations* $R_1, \ldots, R_n$*. Then there is a run in which relation* R_n *is in state* s *if and only if there is such a run whose length is at most* $(\sum_{i=1}^{n} |R_i|)^3|P||M|$*.*

Proof. The proof is an immediate corollary of Lemma 12. If there is a run r that leads to a certain state of R_n, then since all middleboxes are progressing we get that the number of intermediate network states k is at most $(\sum_{i=1}^{n} |R_i|)$. We denote the intermediate states by $s_1, \ldots, s_k$. By Lemma 12, there is also a run r' that visits the same k states and stays in state s_i at most $(k-i)^2|P||M| \leq k^2|P||M|$ steps. Therefore $|r'| \leq k^3|P||M|$.

Since the size of each relation is polynomial in the size of the network, we conclude:

Theorem 3. *The safety problem w.r.t. the unordered semantics for progressing networks is coNP-complete.*

7.4 Unordered Safety of Arbitrary Networks is in EXPSPACE

In this section we show how to solve the non-safety problem of symbolic networks by a reduction to the *coverability problem* of *vector addition systems* (VAS), a.k.a. petri-nets, which is EXPSPACE-complete [26].

A VAS is a pair $(x_0 \in \mathbb{N}^k, X \subset \mathbb{Z}^k)$, where x_0 is the *initial value vector* and X is a set of transition vectors, each with k *dimensions*. A finite run in the VAS is a sequence of transitions $x_1, x_2, \ldots, x_\ell$, such that for every $i \in \{1, \ldots, \ell\}$ the sum $x_0 + x_1 + \cdots + x_i$ is non-negative in all dimensions. The coverability problem asks whether a VAS has a run $x_1, x_2, \ldots, x_\ell$ with $\sum_{i=0}^{\ell} x_i \geq y$, where y is an input vector.

VAS construction We sketch a polynomial encoding of a network as a VAS. Roughly speaking, the transitions of the VAS are used to simulate the processing of packets in the network. Their non-deterministic nature captures the non-deterministic order of network events. We first introduce the VAS dimensions and their roles in the simulation.

Channel simulation: To keep track of the packets over the unbounded channels, we assign a *packet* dimension to every packet $p \in P$ and every channel. The initial value of each packet dimension is 0, it is incremented whenever a packet is added to a channel, and decremented whenever a packet is processed.

Relation simulation: To keep track of relation values, we assign two dimensions, *active* and *inactive*, to every relation R and every tuple $\overline{d}$ in the domain of R. The active dimension indicates whether $\overline{d} \in R$ and the inactive one indicates whether $\overline{d} \notin R$. Both dimensions will have only values of 0 or 1. We need two dimensions since the VAS semantics does not allow to encode negative (e.g., non-memebership) conditions.

Single step simulation: To make sure that no two packets are simultaneously processed, we introduce a *scheduler* dimension. The scheduler dimension has initial value 1, it is decremented whenever a packet processing starts, and incremented when it ends. In addition, to keep track of which command needs to be executed, we assign a *command* dimension to every guard and command, including an *abort* dimension (if an abort command exists). The guard/command dimension has value 1 when the command needs to be executed. Finally, to keep track of values of variables (e.g., src, dst, tag, prt), we assign a dimension for every possible value d of variable e_i. The dimension of (e_i, d) has value 1 if and only if e_i has value d.

Specifically, for each ingress channel of a middlebox m, every packet p, and every guarded command of m, we introduce a *start* transition that captures the beginning of a packet processing event. In addition, we define transitions for each guard and command.

A *start* transition decrements the corresponding packet dimension and the scheduler dimension and increments the dimensions of the corresponding packet source, destination, tag and port variables. In addition, it increments the dimension of the corresponding guarded command, indicating that this is the command to be executed.

We demonstrate the construction of guard and command transitions on a guard of the form e_i **in** R. For every possible value d of e_i we introduce a transition that decrements (i) the dimension of (e_i, d); (ii) the active dimension of (R, d); and (iii) the **if** instruction dimension, and increments: (i) the dimension of (e_i, d); (ii) the active dimension of (R, d); and (iii) the command dimension of the next command. We note that decrementing and incrementing the same dimension allows us to make the transition firable only if the value of the dimension is positive. If the guard is negated, the inactive dimension of (R, d) is used. Transitions of other guards and commands are constructed simlarly.

Non-safety of the network then amounts to a run in the VAS where an *abort* dimension gets a positive value. The reduction, combined with the lower bound implies:

Theorem 4. *The safety problem of arbitrary stateful networks w.r.t. the unordered semantics is EXPSPACE-complete.*

8 Implementation and Case Studies

In this section, we describe a prototype implementation of a tool for verification of stateful networks, and describe our initial experience while running the tool on the networks listed in Example 2 and illustrated in Fig. 3. For the experiments we used quad core Intel Core i7-4790 CPU with 32GB memory.

Increasing Middleboxes Increasing networks are verified using LogicBlox, a Datalog based database system [4]. The Multi-Tenant Datacenter example is an increasing network. Our tool produced a datalog program with 35 predicates, 153 rules and 29 facts. LogicBlox successfully reached a fixed point in 3s, and proved all required properties.

Arbitrary Middleboxes Progressing and Arbitrary networks are verified using LOLA, a Petri-Net model checker [28]. In the Load Balancer and Rate Limiter example our tool created a P/T net with 243 places and 663 transitions; it was successfully verified in 30ms. In the Firewall and Proxy example our tool produced a P/T net with 530 places and 4447 transitions. LOLA successfully verified the resulting petri-net in 0.2s.

9 Conclusion and Related Work

In this paper, we investigated the complexity of reasoning about stateful networks. We developed three algorithms and several lower bounds. In the future we hope to develop practical verification methods utilizing the results in this paper. Below we survey some of the most closely related work.

Topology-independent verification The earliest use of formal verification in networking focused on proving correctness and checking security properties for protocols [10,27]. Recent works such FlowLog [21] and VeriCon [5] also aim to verify the correctness of a given middlebox implementation w.r.t any possible network topology and configuration, e.g., flow table entries only contain forwarding rules from trusted hosts.

Immutable topology-dependent verification Recent efforts in network verification [20,8,15,16,32,30,3,13] have focused on verifying network properties by analyzing forwarding tables. Some of these tools including HSA [14], Libra [34] and VeriFlow [16]. These tools perform near real-time verification of simple properties, but they cannot handle dynamic (mutable) datapaths.

Mutable topology-dependent verification SymNet [33] has suggested the need to extend these mechanisms to handle mutable datapath elements. In their mechanism the mutable middlebox states are encoded in the packet header. This technique is only applicable when state is not shared across a flow (*i.e.,* the middlebox can punch holes, but do no more), and will not work for cache servers or learning switches.

The work in [24] is the most similar to our model. Their work considers Python-like syntax enriched with uninterpreted functions that model complicated functionality. However [24] do not define formal network semantic (e.g., FIFO vs ordered channels) and do not give any formal claim on the complexity of the solution.

Channel systems Channel systems, also called Finite State Communicating Machines, are systems of finite state automata that communicate via asynchronous unbounded FIFO channels [6,7]. They are a natural model for asynchronous communication protocolsand, indeed, they form the semantical basis of protocol specification languages such as SDL and Estelle. Unbounded FIFO channels can simulate unbounded Turing machine tape and therefore all verification problems are undecidable. Abdulla and Jonsson [1] introduced *lossy channel systems* where messages can be lost in transit. In their model the reachability problem is decidable but has a non-primitive lower bound [29].

In this work we use unordered (non-lossy) channels as a different relaxation for channel systems. The unordered semantics over-approximates the lossy semantics w.r.t. safety, as any violating run w.r.t. the lossy semantics can be simulated by a run w.r.t. the unordered semantics where "lost" packets are starved until the violation occurs.

However, the unordered semantics is sufficiently precise for many network protocols in which order is not guaranteed and hence not relied on, and in addition it allows us to have verification procedures with elementary complexity.

References

1. P. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Logic in Computer Science, 1993. LICS'93., Proceedings of Eighth Annual IEEE Symposium on*, pages 160–170. IEEE, 1993.
2. P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Logic in Computer Science, 1996. LICS'96. Proceedings., Eleventh Annual IEEE Symposium on*, pages 313–321. IEEE, 1996.
3. C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. NetKAT: Semantic foundations for networks. In *POPL*, 2014.
4. M. Aref, B. ten Cate, T. J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T. L. Veldhuizen, and G. Washburn. Design and implementation of the logicblox system. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1371–1382. ACM, 2015.
5. T. Ball, N. Bjørner, A. Gember, S. Itzhaky, A. Karbyshev, M. Sagiv, M. Schapira, and A. Valadarsky. Vericon: towards verifying controller programs in software-defined networks. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, page 31, 2014.
6. G. V. Bochmann. Finite state description of communication protocols. *Computer Networks (1976)*, 2(4):361–372, 1978.
7. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM (JACM)*, 30(2):323–342, 1983.
8. M. Canini, D. Venzano, P. Peres, D. Kostic, and J. Rexford. A nice way to test openflow applications. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, 2012.
9. E. Cardoza, R. Lipton, and A. R. Meyer. Exponential space complete problems for petri nets and commutative semigroups (preliminary report). In *Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 50–54. ACM, 1976.
10. E. M. Clarke, S. Jha, and W. R. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Programming Concepts and Methods, IFIP TC2/WG2.2,2.3 International Conference on Programming Concepts and Methods (PROCOMET '98) 8-12 June 1998, Shelter Island, New York, USA*, pages 87–106, 1998.
11. A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1):63–92, 2001.
12. A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. D. Millstein. A general approach to network configuration analysis. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015*, pages 469–483, 2015.
13. N. Foster, D. Kozen, M. Milano, A. Silva, and L. Thompson. A coalgebraic decision procedure for netkat. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 343–355, 2015.
14. P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte. Real time network policy checking using header space analysis. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13)*, 2013.
15. P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)*, 2012.

16. A. Khurshid, W. Zhou, M. Caesar, and B. Godfrey. Veriflow: verifying network-wide invariants in real time. *Computer Communication Review*, 42(4):467–472, 2012.
17. M. Kuzniar, P. Peresini, M. Canini, D. Venzano, and D. Kostic. A soft way for openflow switch interoperability testing. In *CoNEXT*, pages 265–276, 2012.
18. K.-J. Lange and P. Rossmanith. The emptiness problem for intersections of regular languages. *Mathematical Foundations of Computer Science 1992*, pages 346–354, 1992.
19. N. P. Lopes, N. Bjørner, P. Godefroid, K. Jayaraman, and G. Varghese. Checking beliefs in dynamic networks. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015*, pages 499–512, 2015.
20. H. Mai, A. Khurshid, R. Agarwal, M. Caesar, B. Godfrey, and S. T. King. Debugging the Data Plane with Anteater. In *SIGCOMM*, 2011.
21. T. Nelson, A. D. Ferguson, M. J. G. Scheer, and S. Krishnamurthi. Tierless programming and reasoning for software-defined networks. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014, Seattle, WA, USA, April 2-4, 2014*, pages 519–531, 2014.
22. OpenStack. LogicBlox. `http://www.logicblox.com/` retrieved 07/07/2015.
23. A. Panda, K. J. Argyraki, M. Sagiv, M. Schapira, and S. Shenker. New directions for network verification. In *1st Summit on Advances in Programming Languages, SNAPL 2015, May 3-6, 2015, Asilomar, California, USA*, pages 209–220, 2015.
24. A. Panda, O. Lahav, K. Argyraki, M. Sagiv, and S. Shenker. Verifying isolation properties in the presence of middleboxes. *arXiv preprint arXiv:1409.7687*, 2014.
25. R. Potharaju and N. Jain. Demystifying the dark side of the middle: a field study of middlebox failures in datacenters. In *Proceedings of the 2013 Internet Measurement Conference, IMC 2013, Barcelona, Spain, October 23-25, 2013*, pages 9–22, 2013.
26. C. Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978.
27. R. W. Ritchey and P. Ammann. Using model checking to analyze network vulnerabilities. In *Security and Privacy*, 2000.
28. K. Schmidt. Lola a low level analyser. In *Application and Theory of Petri Nets 2000*, pages 465–474. Springer, 2000.
29. P. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251–261, 2002.
30. D. Sethi, S. Narayana, and S. Malik. Abstractions for model checking sdn controllers. In *FMCAD*, 2013.
31. J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. In *SIGCOMM*, 2012.
32. R. Skowyra, A. Lapets, A. Bestavros, and A. Kfoury. A verification platform for sdn-enabled applications. In *HiCoNS*, 2013.
33. R. Stoenescu, M. Popovici, L. Negreanu, and C. Raiciu. Symnet: static checking for stateful networks. In *Proceedings of the 2013 workshop on Hot topics in middleboxes and network function virtualization*, pages 31–36. ACM, 2013.
34. H. Zeng, S. Zhang, F. Ye, V. Jeyakumar, M. Ju, J. Liu, N. McKeown, and A. Vahdat. Libra: Divide and conquer to verify forwarding tables in huge networks. In *NSDI*, 2014.