Using Routers to Build Logic Circuits: How Powerful is BGP?

Marco Chiesa* Luca Cittadini* Giuseppe Di Battista* Laurent Vanbever* Stefano Vissicchio†
*Roma Tre University *Princeton University †Université catholique de Louvain
*{chiesa,ratm,gdb}@dia.uniroma3.it *vanbever@cs.princeton.edu †stefano.vissicchio@uclouvain.be

***Abstract*—Because of its practical relevance, the Border Gateway Protocol (BGP) has been the target of a huge research effort since more than a decade. In particular, many contributions aimed at characterizing the computational complexity of BGP-related problems. In this paper, we answer computational complexity questions by unveiling a fundamental mapping between BGP configurations and logic circuits. Namely, we describe simple networks containing routers with elementary BGP configurations that simulate logic gates, clocks, and flip-flops, and we show how to interconnect them to simulate arbitrary logic circuits. We then investigate the implications of such a mapping on the feasibility of solving BGP fundamental problems, and prove that, under realistic assumptions, *BGP has the same computing power as a Turing Machine*. We also investigate the impact of restrictions on the expressiveness of BGP policies and route propagation (e.g., route propagation rules in iBGP and Local Transit Policies in eBGP) and the impact of different message timing models. Finally, we show that the mapping is not limited to BGP and can be applied to generic routing protocols that use several metrics.**

I. Introduction and Related Work

The Border Gateway Protocol (BGP) [1] is the de-facto routing protocol that regulates inter-domain routing. BGP comes in two flavors: external BGP (eBGP) and internal BGP (iBGP). eBGP is used to exchange reachability information between neighboring networks or Autonomous Systems (AS), while iBGP is used to distribute externally-learned routes within an AS.

BGP enables each AS to apply routing policies in complete autonomy, i.e., enabling each AS to fully control the routes that it accepts, prefers, and propagates to its neighboring ASes. While such a rich policy expressiveness can support complex business relationships, it can also cause routing and forwarding anomalies both in eBGP [2] and iBGP [3] configurations.

Because of its practical relevance for Internet operation and its lack of correctness guarantees, BGP has been the focus of many research and industrial efforts in the last 15 years. Results of such an effort encompass formal analyses of the protocol (*e.g.*, [2], [3]), experimental measurements of disruptions due to BGP (*e.g.*, [4], [5]), proposal of configuration guidelines (*e.g.*, [6]) and of protocol modifications (*e.g.*, [7]), and practical approaches to check a given configuration for correctness (*e.g.*, [8], [9]). However, all previous studies missed a fundamental analogy: Basic BGP configurations can encode elementary logic gates but also, memory and clock components. As such, BGP is powerful enough to encode logic circuits of arbitrary complexity, as we show in Section II.

978-1-4799-1270-4/13/$31.00

We build this mapping assuming a simplified model for BGP routing policies which does not include advanced BGP features like MED or conditional advertisement.

In this paper, we investigate the theoretical consequences of the existence of such a mapping between BGP configurations and logic circuits. We make the following four contributions.

First, we leverage the mapping to characterize the computational complexity of several routing problems in a "bounded" asynchronous model. Contrary to previous works on BGP complexity, in this model each network link is associated with a network delay bounded between finite minimum and maximum values. This effectively imposes a partial order on the exchange of BGP updates. Previous lower bounds for BGP related problems have been proved in models that allow BGP messages to be arbitrarily (even if not indefinitely) delayed [2], [3], [10], [11], [12], [13], [14]. Past work also suggested to study BGP with game-theoretic approaches, where ASes act as players and BGP policies as players' strategies. Messages between players are still allowed to be arbitrarily (even if not indefinitely) delayed. However, these approaches fail to capture specific BGP features either in the game (*e.g.*, by assuming that routers can directly receive routes from non-neighbors [15]) or in the strategies (*e.g.*, by considering impossible strategies in BGP [16], [17]). In Section III, we show that *BGP configurations can simulate arbitrary Turing Machines* in the considered bounded asynchronous model. Two implications derive from this observation. First, policy-based protocols like BGP intrinsically have the same computational power of Turing Machines, even when simple policies are considered. Second, it enables us to assess the computational intractability of BGP routing problems, like routing convergence and correct route propagation.

Second, in Section IV, we use the mapping to investigate the impact of policy restrictions on the complexity of BGP problems. We analyze both iBGP networks and eBGP policy configuration paradigms like the well-known Gao-Rexford conditions [6] and the widely used Local Transit Policies [18]. Also, we discuss the extent to which the mapping holds when other message timings are considered.

Third, in Section V, we show that our methodology can be applied in a routing framework that is different from BGP, and we investigate how difficult the analysis of a generic routing protocol using several metrics is.

Finally, we prove that our approach can be used in several message timing models in Section VI. In particular, we show

that the complexity of unstudied routing problems can be assessed in the bounded asynchronous models [2] by relying on the mapping between BGP configurations and logic gates.

II. BGP Configurations as Logic Circuits

The most prominent feature of BGP is the support for routing policies that can be independently defined on each BGP router. Routing policies are used to specify which routes should be accepted from (or announced to) which neighbors, and to assign different degrees of preference to different routes.

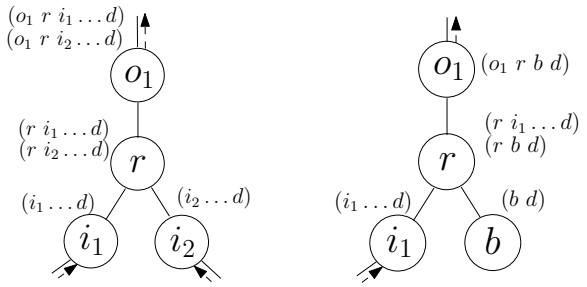
In this paper we rely on the well-known SPP formalism [2] to model eBGP configurations (in Section IV-A we use a similar model to represent iBGP configurations). In SPP, an eBGP configuration is represented as a graph where every node is an Autonomous System (AS) and every edge is an eBGP peering. Since BGP routers treat different destinations separately, we focus on one destination at the time. The destination is represented by a special node d, to which all other nodes try to establish a route. A route is a simple path on the graph. Each node can specify its own policy, which is modeled as a list of all the routes that the node accepts towards the destination. The order of the elements in the list corresponds to the preference of the node.

Despite the fact that SPP is a simplified model for BGP policies (it does not capture MEDs and conditional route announcements, just to name a few), in this section we show that SPP instances representing eBGP configurations can emulate any logic circuit. First, we show eBGP configurations that implement elementary logic gates. Second, we describe eBGP configurations for more advanced circuital components, namely flip-flops and clocks. Finally, we describe how to arbitrarily connect circuital components.

A. Elementary Logic Gates with eBGP

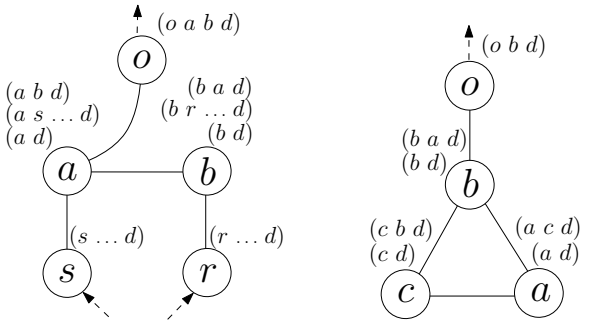
We now show how to build eBGP configurations that simulate the OR and NOT logic gates. We map the inputs (outputs, resp.) of a logic gate to a set of *input nodes* (*output nodes*, resp.) of the SPP instance. Also, we map the availability of a route to a 1 (true value) and the absence of a route to a 0 (false value). In particular, the availability (absence, resp.) of a route at an output node r at time t means that the output signal of r at time t is 1 (0, resp.).

The eBGP configurations simulating the OR and the NOT logic gates are shown in Fig. 1. The graphical convention we use in the figure is adopted throughout the paper, unless differently specified. ASes are represented by circles, and solid edges represent eBGP peerings. A list of paths is specified beside each AS. Each list contains the paths that the AS accepts (i.e., paths that are not filtered out by the routing policy) in a descending order of preference. All routes refer to the same destination d. We use dots inside a path when we do not specify the entire path, so $(a\ b \dots d)$ represents a path that start at a, traverses b and ends at d. Incoming and outgoing dashed arrows indicate input and output nodes, respectively. For the sake of brevity, whenever it is clear from the context, we omit node d and its peerings. For example, in Fig. 1(b), d should be considered directly attached to a, b and c.



(a) The OR gate. (b) The NOT gate.

Fig. 1. eBGP networks simulating basic logic gates.



(a) A DISAGREE with two extra nodes S and R behaves as a flip-flop.

(b) A BAD-GADGET plus output node o behaves as a clock.

Fig. 2. eBGP networks simulating memory and clock.

Fig. 1(a) represents an eBGP configuration corresponding to the OR gate. Nodes i_1 and i_2 correspond to the inputs and node o_1 corresponds to the output. Since node o_1 only accepts routes from r, it will have a route to d if and only if either i_1 or i_2 has a route to d. Similarly, Fig. 1(b) represents an eBGP configuration simulating the NOT gate, where i_1 is the input and o_1 is the output. In this configuration, o_1 has a route to d if and only if i_1 has no route to d. Indeed, if i_1 has a route to d, r receives and selects the route from i_1 instead of the route from b (that is always available at r) because of its preferences. Thus, o_1 will end up with no route, since o_1 does not accept path $(o_1\ r\ i_1 \dots d)$, as shown by the absence of the path in the list aside o_1 in Fig. 1(b)). On the contrary, if i_1 has no route to d, then r selects $(r\ b\ d)$ and, consequently, o_1 selects $(o_1\ r\ b\ d)$.

B. Memory and Clock with Popular eBGP Gadgets

Besides encoding elementary gates, eBGP is powerful enough to simulate more complex logic components, like flip-flops and clock generators.

Fig. 2(a) shows an eBGP configuration that simulates an SR flip-flop. This flip-flop has two inputs S (set bit) and R (reset bit) and one output Q. The flip-flop stores and outputs a 1 (0, resp.) whenever the set (resp., reset) bit is set to 1 (0, resp.). If both set and reset bits are set to 0, then output Q is the stored value. Setting both S and R to 1 is not allowed. The configuration in Fig. 2(a) simulates this behavior. It is based on the presence of a well-known BGP gadget, called

DISAGREE [2], that has two stable states. Indeed, nodes a and b form a DISAGREE. In one stable state, nodes a and b select paths $(a\ b\ d)$ and $(b\ d)$, respectively. In the other one, nodes a and b select $(a\ d)$ and $(b\ a\ d)$, respectively. Depending on whether nodes s and r receive a route, we have the following three cases. If s announces a route to a and r does not announce any route to b, then a never selects $(a\ d)$, since path $(a\ s \dots d)$ is available and more preferred than $(a\ d)$. Hence, b has to select $(b\ d)$ because none of $(b\ a\ d)$ and $(b\ r\ \dots\ d)$ is available. Since a receives $(b\ d)$, it can select its best path $(a\ b\ d)$. Symmetrically, if r announces a route to b and s does not announce any route to a, then a has to select $(a\ d)$. Finally, if neither s nor r receives a route, then the DISAGREE does not change its stable state. As a consequence, node o has an available path to d if and only if node a selects path $(a\ b\ d)$, hence mirroring the output of an SR flip-flop.

Further, the dynamics of eBGP configurations that admit no stable state are conceptually similar to those of clock generators. A *clock generator* is a logic circuit producing a signal that oscillates between 1 and 0. The BAD-GADGET [2], shown in Fig. 2(b), is a gadget that never converges to a stable state. It consists in a cycle of three nodes a, b, and c, in which each node prefers a route through its successor instead of a direct route to d. When the gadget oscillates, node a alternatively selects paths $(a\ c\ d)$ and $(a\ d)$. Since o does not accept path $(a\ c\ d)$ from a, o has a route only when node a selects $(a\ d)$. Therefore, the output node o will alternate forever between having a route and not having any route, as for a clock generator. Observe that the clock of Fig. 2(b) can be thought in terms of the circular interconnection of 3 NOT gates of Fig. 1(b).

C. Simulating Arbitrary Logic Circuits

Now that we have the elementary logic components, it would be tempting to simply interconnect them using eBGP peerings. Such an operation is needed for building: (i) the AND gate, using OR and NOT and applying the De Morgan's laws; (ii) arbitrary logic gates as a combination of AND, OR, and NOT; and (iii) arbitrary logic circuits starting from logic gates, flip-flops and clocks. Unfortunately, because of BGP peculiarities, arbitrary interconnections are not straightforward.

The first problem we face is that signal propagation in logic circuits has a direction, while routes may traverse an eBGP peering in both ways. We need to prevent routes from being propagated in unintended directions, e.g., "signals" traversing the gates from their output to their input. This can be accomplished by using eBGP policies to accept only routes in the intended direction.

A second and more subtle problem arises with loops. BGP has a built-in control plane loop prevention mechanism [1] which mandates an AS to discard routes containing its own identifier. Because of this mechanism, we need an additional building block to be able to simulate logic circuits where the signal is propagated through a loop. In particular, we interpose a special gadget, called HUB gadget, between any pair of interconnected logic components.

The HUB gadget is in Fig. 3. Intuitively, it takes a route at its input node i and generates a new, completely different

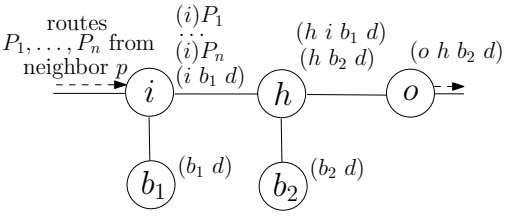


Fig. 3. The HUB gadget we use to interconnect logic components.

route at its output node o. It can be seen as the concatenation of two NOT gadgets, in which the first NOT gadget filters out the original route and the second NOT gadget generates the new one. No route is produced at the output if i receives no external route. In other words, the HUB gadget is able to correctly propagate both the presence of a route (a binary 1) and the absence thereof (a binary 0). Nodes h, b_1 and b_2 are different for each HUB gadget and therefore cannot appear in any external route received by i. This guarantees that the output route cannot share any node (besides d) with the input route, which in turn keeps BGP's loop prevention mechanism from being triggered. More precisely, if i receives no route from its neighboring node p, i selects route $(i\ b_1\ d)$. This allows h to select its preferred path $(h\ i\ b_1\ d)$, which in turn makes o unable to select any valid route to d. Otherwise, if p advertises a route $(p \dots d)$ to i, then i selects $(i\ p \dots d)$. As a consequence, h selects $(h\ b_2\ d)$, and o selects $(o\ h\ b_2\ d)$.

Observe that the output node of the HUB gadget can either announce no route, or it can announce a single route which does not depend on the route received by the input node. For this reason, when connecting the output node o of the HUB gadget to the input node i' of another gadget, i' can receive only one route. Given a logic circuit, w.l.o.g. composed by NOT and OR gates, we can replace each gate with the gadgets of Sect. II-A and replace each wire with a HUB gadget obtaining a BGP network computing the same function. Since the gadgets and the HUB have a constant number of routers, each requiring a constant size routing configuration, we have that the construction is done in polynomial time.

Restricting our attention to combinational logic circuits, we have a first interesting consequence of our constructions. Since a combinational logic circuit can encode any logic formula, the fact that eBGP can be used to construct combinational logic circuits gives new intuition of why most problems related to BGP are NP-hard. In fact, by encoding a logic formula in BGP, it is typically possible to obtain a polynomial reduction from SAT [19], a well-known NP-complete problem.

III. UNDERSTANDING THE COMPLEXITY OF BGP USING LOGIC GATES

The fact that eBGP configurations can simulate logic circuits has several implications in terms of the computational complexity of routing problems. To deep out investigation in this direction we have to model BGP dynamics. We consider a *bounded* asynchronous model, where messages traversing a link have a propagation delay between a minimum and a maximum value. These values depends on many factors (e.g., MRAI timers) and can be arbitrarily set by a network operator.

Even in the absence of a better guess, an operator can bind the minimum delay value to the physical delay of the connection and the maximum to "a couple of days". We recall that since BGP updates travel into TCP connections, packet loss and out-of-order packets are not an issue.

A. Building a Turing Machine with Logic Gates

In the bounded BGP model, each link l is associated with a propagation delay that can take any value within range (m_l, M_l), where m_l (M_l, resp.) is the minimum (maximum, resp.) delay value for l. Both m_l and M_l are finite values different from 0. Observe that, if $m_l = M_l$ all BGP message exchanges are completely synchronized. In general, however, we assume $m_l \neq M_l$. Analogously, we can define a bounded model for logic circuits associating a minimum and a maximum propagation delays on each wire.

A BGP network is *safe* [2] if, for each possible execution, the network converges to a stable state. SAFETY is defined as the problem of checking if a BGP network is safe. A BGP network *oscillates* if it is not safe. A logic circuit *halts* if, for each timing, there is a time instant when (i) for each link its endpoints have the same value and (ii) for each gate, its output value is the correct output with respect to the current gate inputs. A logic circuit *oscillates* if it does not halt.

Using standard circuit design methodologies for the bounded model (e.g., [20]) we can use logic gates in the bounded model to construct a Finite Turing Machine (FTM) [21], i.e., a Turing Machine where the size of the tape is finite. More details on such a construction are reported in [22]. An FTM is a simple device that reads and writes symbols on a finite tape according to a table of rules. This enables us to show that most BGP routing problems are at least as difficult as the "halting" problem for an FTM, which is known to be PSPACE-hard [23]. Hence, we have the following lemma.

Lemma 1: Given a Finite Turing Machine M, it is possible to construct in polynomial time a logic circuit C in the bounded model such that C halts iff M halts.

The discussion of Section II-C shows that using eBGP we can construct logic circuits. Since we are using a BGP model with bounded delays, then we can simply assign the desired delays to BGP peerings. Hence, exploiting Lemma 1, we have the following theorem.

Theorem 1: Given a Finite Turing Machine M, it is possible to construct in polynomial time an eBGP network N in the BGP bounded model such that N converges to a stable state iff M halts.

The ability to simulate FTMs with eBGP configurations enables us to prove PSPACE-hardness results for BGP problems. We reduce those problems from the LINEAR SPACE ACCEPTANCE problem, which is known to be PSPACE-complete [23]. An instance of LINEAR SPACE ACCEPTANCE consists of a FTM M and a finite string x, where the size of the tape of M is linear with respect to the size of x. The problem is to verify if M accepts x. We say that an FTM M accepts a string x if M halts on an acceptance state given that x is initially written on its tape.

In the following, we prove that both SAFETY [2] and REACHABILITY [10] are PSPACE-hard. The PSPACE class contain all problems that can be solved by a TM using a tape of polynomial length w.r.t the size of the input string [19]. A PSPACE-hard problem is a representative problem of the PSPACE class. It is known that, if a PSPACE-hard problem can be solved in polynomial time, then every problem in PSPACE can be solved in polynomial time and, since NP is believed to be a proper subset of problems of PSPACE, also every NP problem can be solved in polynomial time. For this reason, PSPACE-hard problems are considered to be harder than NP-hard problems. As a consequence, SAT solvers [24], which are a practical tool used to deal with NP-hard problems, cannot be used for PSPACE-hard problems.

Theorem 2: SAFETY is PSPACE-hard.

Proof: We reduce SAFETY from the LINEAR SPACE ACCEPTANCE problem. A similar construction with respect to that described above enables to build an eBGP configuration that simulates an arbitrary FTM M in such a way that the network converges to a stable state if and only if M reaches an acceptance state. This polynomial-time reduction directly yields the statement. ■

In [15] BGP SAFETY is proved to be PSPACE-complete in an unrealistic game-theoretical model in which BGP speakers are assumed to be omniscient and BGP messages are not passed router by router (i.e., router receives routing update as in a link-state protocol). We stress the fact that this unrealistic assumption fails to capture the communication model of BGP in which messages are exchanged as in a distance-vector protocol.

A very similar reduction from LINEAR SPACE ACCEPTANCE can be leveraged to show the complexity of the REACHABILITY problem [10], that is, deciding whether a BGP configuration admits a stable state in which a given node s has a route to a given destination d. Namely, it is sufficient to build an BGP configuration that simulates the FTM M as shown in the proof of Theorem 2 and modify it such that node s is guaranteed to have a route if and only if the BGP gadget simulating the clock of the FTM has stopped oscillating.

Observe that, theoretically, an infinite BGP network would be able to simulate a Turing Machine, where each cell of the infinite tape is modeled by a certain number of routers. In a sense this means that, despite the simplifications listed in Section II, unrestricted BGP policies have the same expressive power as Turing Machines. As a consequence, since the halting problem for a TM is undecidable [25], also SAFETY would be an undecidable problem for an infinite BGP network.

IV. THE IMPACT OF POLICY RESTRICTIONS

Intuitively, the fact that BGP configurations can encode arbitrary logic circuits suggests that the complexity of BGP related problems stem out of the intrinsic complexity of BGP semantics, which ultimately maps to the expressiveness of BGP policies. One might argue that it is not surprising that completely unrestricted policies yield complex semantics. It is therefore interesting to study whether restricting BGP policies significantly simplifies the analysis of a BGP configuration.

In this section, we consider iBGP configurations, where policies are dictated by the iBGP route propagation rules and IGP distances, Local Transit policies, where policies depend solely on the ingress and egress AS, and Gao-Rexford conditions, where policies are tied to commercial relationships among ASes.

A. Restricting to iBGP

As opposed to eBGP, in iBGP all routers belong to the same AS. For this reason, routing policies are typically not applied on iBGP messages [3]. However, the specification of iBGP with route reflection [26] imposes an implicit route ranking and an implicit route filtering. The ranking component is restricted in that it has to be consistent with the IGP graph. The filtering component is restricted in that it has to be consistent with the iBGP graph.

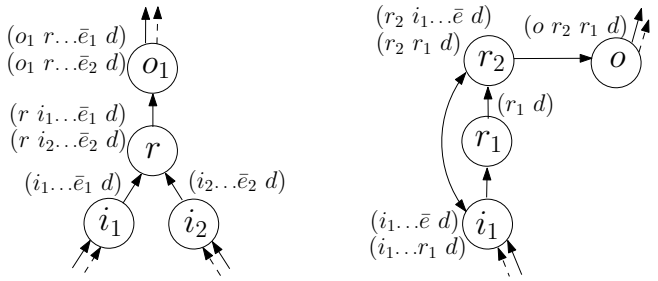
In particular, the BGP decision process has some tie breaking rules that only have local significance (e.g., IGP distance), therefore routing preferences are implicitly imposed by the BGP decision process itself. Further, when route reflection is used, the protocol requires certain routes to be filtered out at each iBGP router. More precisely, the iBGP neighbors of each router are split into three sets: *clients*, *peers* and *route-reflectors*. Best routes are always relayed to clients, but best routes learned from peers or route-reflectors are not propagated to other peers and route-reflectors.

We now show that, despite the restrictions above, the protocol still retains enough expressive power to encode arbitrary logic circuits. First of all, observe that we can consider just egress points preferences, disregarding the details of the IGP graph. In fact, in the following we show that for any given set of egress point preferences there exists an IGP graph which is consistent with those preferences. Consider a single destination d. Let $\mathcal{E}$ be the set of egress points to d. Let $\lambda^v : \mathcal{E} \to \mathbb{N}$ be the egress point ranking function of router v, such that $\lambda^v(e_j) = i$ if and only if e_j is the i-th most preferred egress point by v. Since the BGP process forces each router to deterministically select only one route, egress point preferences at each router are totally ordered, that is, $\forall e_i \neq e_j \ \lambda^v(e_i) \neq \lambda^v(e_j)$. Given the ranking functions of all the routers in the networks, the following algorithm, that we call IB, builds an IGP graph which is consistent with those ranking functions as follows:

1) Create an empty IGP graph where the nodes are the routers and the egress points of the network.
2) For each pair of a router r and an egress point e, add a link (r, e) in the IGP graph.
3) To each link (r, e), assign a weight $w(r, e) = \lambda^r(e) + |\mathcal{E}|$.

We now prove that the IB algorithm is correct, that is, it builds an IGP graph consistent with the given egress point preferences at each router. Let $dist(v, u)$ be the length of the shortest path from v to u. We say that an IGP topology *realizes* the given ranking functions if for each router $v \notin \mathcal{E}$ and each arbitrary pair of distinct egress points e_1 and e_2, $\lambda^v(e_1) < \lambda^v(e_2)$ implies that $dist(v, e_1) < dist(v, e_2)$.

Lemma 2: In the IGP topology built by the IB algorithm, the shortest path between any router r and any egress point e is $(r\ e)$.



(a) The OR gate in iBGP. (b) The NOT gate in iBGP.

Fig. 4. iBGP configurations which simulate logic gates in iBGP.

Proof: Let $G = (V, E)$ be the IGP topology built by the IB algorithm. Consider any router r and any egress point e. By construction, the weight of the path $(r\ e)$ is equal to $w(r, e) = \lambda^r(e) + |\mathcal{E}| \leq 2|\mathcal{E}|$. We now show that any path P from r to e, with $P \neq (r\ e)$, has a weight higher than $w(r, e)$. By definition of P, P contains at least two edges. By definition of the weight function adopted in the IB algorithm, the weight of P is equal or greater to $2 + 2|\mathcal{E}|$. Hence, the weight of any path $P \neq (r\ e)$ is higher with respect to $(r\ e)$, yielding the statement. ■

We are ready to prove the following theorem on the correctness of our construction.

Theorem 3: Given a set Λ of ranking functions, the IB algorithm builds an IGP topology that realizes Λ.

Proof: Let $G = (V, E)$ be the IGP topology built by the IB algorithm. Consider a router r and any pair of egress points e_1 and e_2, such that r prefers routes from e_1 over routes from e_2. By Lemma 2, $dist(r, e_1) = w(r, e_1)$ and $dist(r, e_2) = w(r, e_2)$. By definition of the weight function used in the algorithm, we have $w(r, e_1) < w(r, e_2)$, which proves the statement. ■

Exploiting the IB algorithm, we now show how to construct OR and NOT gates with iBGP configurations, as shown in Fig. 4. In these figures, one-headed solid arrows represent sessions from a client to its route reflector, while double-headed solid arrows represent sessions between two peers. Inbound and outbound dashed arrows indicate input and output nodes, respectively. Paths aside each router represent the iBGP path towards egress points. The rest of the notation is consistent with the graphical convention introduced in Section II.

The iBGP configuration in Fig. 4(a) simulates the behavior of an OR logic gate. The output router will receive a route to any of the egress points $\bar{e}_1$ and $\bar{e}_2$ if and only if a route is received by either i_1 or i_2 (or both). Similarly, Fig. 4(b) depicts an iBGP configuration corresponding to the NOT gate. If i_1 receives an eBGP route, it will propagate it to r_2. Because of egress point preferences, r_2 will select the route announced by i_1. Now, since this route was learned from a peer, iBGP route propagation rules require that r_2 do not relay the route to o, which therefore is unable to learn any feasible route. On the contrary, if i_1 receives no route, r_2 will select the route announced by r_1 and will propagate it to o. Note that the iBGP configuration corresponding to the NOT gate is based on the

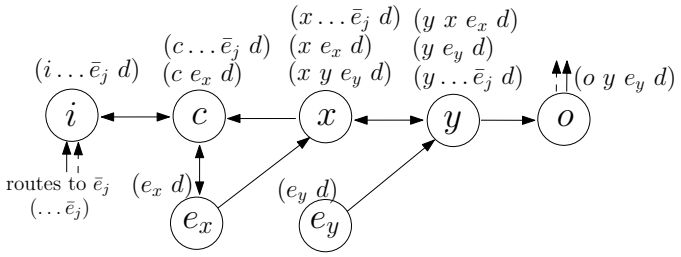
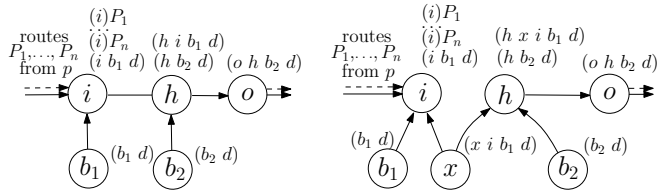


Fig. 5. The iBGP-HUB gadget.



(a) A HUB violating Condition i). (b) A HUB violating Condition iii).

Fig. 6. Variants of the HUB gadget obtained by violating one of the Gao-Rexford conditions. An oriented (unoriented) edge from a to b represents the fact that a is a customer (peer) of b.

OVER-RIDE gadget introduced in [27].

Reference [3] shows examples of iBGP configurations realizing DISAGREE and BAD-GADGET structures. This enables us to build the memory and the clock components as we did for eBGP in Section II.

Finally, to interconnect logic components, we use the iBGP-HUB gadget (see Fig. 5), which is the equivalent of the HUB gadget for iBGP. If i receives an iBGP path R towards any egress point $\bar{e}_j$, then i, c and x also select route R. In this case, y selects path $(y\ e_y)$ because of its ranking function, and propagates it to o. Hence, o receives, selects and propagates one route which has no router in common with the original route R. Otherwise, if i receives no path towards any $\bar{e}_j$, then x selects route $(x\ e_x)$, enabling y to select its most preferred route $(y\ x\ e_x)$. However, y cannot propagate $(y\ x\ e_x)$ to o because of iBGP propagation rules that deny propagation of a path learned from an iBGP peer to a route reflector.

Observe that the iBGP-HUB gadget outputs at most one route, and has at most two routes in input. Also, node i cannot receive paths from c, which implies that routes can only flow from the left to the right part of the gadget, hence preventing propagation of routes in undesired directions. In fact, either i) node i selects a path $R = (i \dots \bar{e}_j)$ which is learned over the client session itself; or ii) node i has no path to select as c's best route $(c\ e_x)$ is learned from an iBGP peer and cannot be propagated to another iBGP peer.

Having all the needed building blocks, an iBGP configuration that simulates a Finite Turing Machine can be done exactly as in Section III for eBGP configurations, again exploiting Lemma 1. As a consequence, we can derive new intractability results (PSPACE-hardness for the min-max model) of all correctness problems defined in iBGP, namely signaling, dissemination and forwarding correctness [3], [27].

B. Local Transit Policies and Gao-Rexford Conditions

We now consider eBGP policy restrictions that have been proposed in the literature.

A common policy configuration practice consists in applying the so-called Local Transit policies [18]. Local Transit policies consist in defining routing policies as functions of the AS that announces the route and of the AS to which the route is announced only. Observe that all the policies used to build the gadgets presented in Section II are compliant with the definition of Local Transit policies. The same holds for the DISAGREE gadget and the BAD-GADGET. As a consequence, BGP problems remain hard (in the unbounded asynchronous model) or very hard (in the bounded asynchronous model) even for BGP networks in which only Local Transit policies are applied. These results extend the findings in [13].

A further restriction with respect to Local Transit Policies consists in imposing that the eBGP configuration satisfies the Gao-Rexford conditions introduced in [6]. These conditions are the most famous way to trade policy expressiveness for correctness guarantees without the need for global coordination among ASes. Gao-Rexford conditions assume that each AS classifies its eBGP neighbors as either customers, peers, or providers, and that: i) routes learned from customers are preferred over those learned from peers and providers; ii) there is no cycle such that each AS in the cycle is a customer of the next AS in the cycle; iii) an AS does not export routes learned from a peer or provider to its peers or providers. It has been proved [6] that the Gao-Rexford conditions guarantee that BGP always converges to a unique stable state and a greedy algorithm proposed in [2] can be used to compute the stable state, and to solve BGP problems in polynomial time.

The BGP networks simulating the NOT and OR logic gates (Fig. 1) are compliant with Gao-Rexford conditions if the output node o_1 is set as provider of r, and r is a provider of all the other nodes. Similarly, the HUB gadget (Fig. 3) is compliant with the Gao-Rexford conditions if o is a provider of h, h is a provider of both i and b_2, and i is a provider of both p and b_1. This assignment of commercial relationships has the property that if a logic circuit does not contain cycles (as in combinational circuits) then it can be simulated by a BGP network that satisfies the Gao-Rexford conditions. Otherwise, cycles in the logic circuits translates to customer-provider cycles in the eBGP configuration, which violates the second Gao-Rexford condition. However, assuming Gao-Rexford conditions prevents us from building arbitrary logic circuits and configurations like a DISAGREE or a BAD-GADGET. This can be seen as an intuitive explanation of why most BGP problems turn out to be polynomial in such a setting.

However, violating any of the Gao-Rexford conditions enables us to build configurations that simulates arbitrary logic circuits, hence arbitrary Finite Turing Machines in the min-max model. We have already shown a customer-provider assignment such that a cycle in a logic circuit translates to a customer-provider cycle in the BGP network. Hence, if we violate condition ii) and customer-provider loops are allowed, then every interconnection between logic components is admitted. Otherwise, if conditions i) or iii) are violated,

we modify the HUB gadget as shown in Fig. 6. In each of these two cases, cycles in the logic circuits are guaranteed not to translate to customer-provider cycles, and only one of the Gao-Rexford conditions is violated at the time. Hence, for any violation of the Gao-Rexford conditions, arbitrary logic circuit can be simulated with BGP configurations. As a consequence, convergence and route propagation problems are still PSPACE-hard if any of the Gao-Rexford condition is violated.

V. COMBINING POPULAR METRICS CAN BE HARD TO ANALYZE

In Sect. II and IV-A, we built a mapping from BGP to logic gates. This mapping exploits two specific BGP constructs: per-neighbor filtering and a ranking function that is based on the presence of a specific vertex in the path (i.e., per-neighbor and per-egress-point ranking in eBGP and iBGP, respectively). One may wonder whether this mapping technique can be applied to protocols different from BGP. In this section, we answer this question. We consider distance-vector routing protocols where each routing message contains a vector of metrics (as in EIGRP [28], where messages contain between five and seven different metrics) and where route filtering is not allowed. We suppose (notice that this is different from EIGRP) that each router make routing decisions based on any available routing metrics (e.g., path length, bandwidth, reliability). Surprisingly, also in this setting, we found this mapping technique to be extremely powerful. We will show that, if a protocol handles more than two metrics, then it is possible to map router configurations to logic gates. As a consequence, we derive the computational intractability results as for BGP.

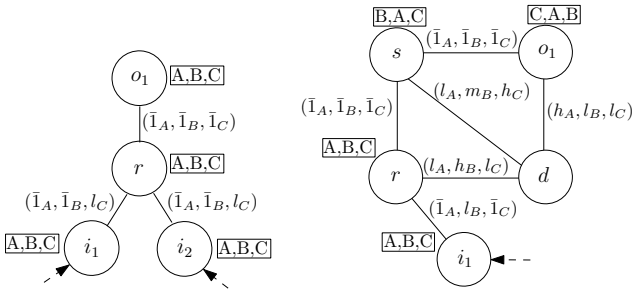
Consider an arbitrary distance-vector routing protocol, where messages associated to each route contain a vector of three metrics A, B, and C. We refer to this protocol as the METRIC-DV. To describe routing metrics, we use the standard terminology from routing algebras [29]. Each metric $S = A, B, C$ has a domain D_S and it is endowed with two binary operators $\oplus_S$ and $\otimes_S$. Given two paths, with metric values s_1 and s_2, from the same router, $s_1 \oplus_S s_2$ returns the value of the most preferred one. We assume that $\oplus$ is *transitive*, i.e., if s_1 is preferred over s_2 and s_2 is preferred over a value s_3, then s_1 is preferred over a third value s_3. Given a path p_n, with metric s_n, from a neighbor n of a router r, where the link joining r with n has metric value $s_{(r,n)}$, $s_{(r,n)} \otimes_S s_n$ returns the metric value of the path from r obtained by concatenating (r, n) with p_n. We denote by $\bar{1}_S$ the identity element of $\otimes_S$. For instance, a path length metric has domain $D_S = \mathbb{N}^\infty$ and operators $(\oplus, \otimes) = (\min, +)$. Each router r, for each path of length c learned from one of its neighbors n, adds ($\otimes$ is $+$) the cost of the link from r to n to c. Also, each router selects the shortest ($\oplus$ is $\min$) path among the available ones. The identity element of $+$ is 0. A bandwidth metric has domain $D_S = \mathbb{N}^\infty$ and it has operators $(\oplus, \otimes) = (\max, \min)$. In this case, the maximum-bandwidth available path is the most preferred and, since the capacity of a path is equal to the bottleneck capacity of that path (the smallest along the path), a minimum operator is used to compute the path bandwidth. The identity element of $\min$ is ∞. A most-reliable metric has domain $D_S = [0, 1]$, operators $(\oplus, \otimes) = (\max, \times)$, and the identity element of $\times$ is 1. When a router r receives a route R_n, with vector $\langle a, b, c\rangle$, from one of its neighbors n, r computes a new vector $\langle e_A \otimes a, e_B \otimes b, e_C \otimes c\rangle$ for route $(r\ n)R_n$, where $\langle e_A, e_B, e_C\rangle$ is the metric of the link between r and n.

In order to discuss the protocol, we define some generic metric values for metrics A, B, and C and a mapping from logic values to routes available at a router. For each metric $S = A, B, C$, let h_S be the most preferred value, m_S be the second-most preferred value, and l_S be the least preferred value. Also, we assume that each metric S is *monotonic*, i.e., for each pair of values x and y of S such that y is preferred over x, $(y \otimes x) \oplus x = x$. It basically means that the concatenation of two paths produces a path whose metric is at least as worse as the path with the lowest metric value. For example, this holds for metrics like path length, bandwidth, and reliability. Indeed, in the path length metric, for two paths of length 10 and 20, respectively, we have that $\min(10 + 20, 20) = 20$. As for the mapping, since no path filtering is allowed, we use a more sophisticated mapping between paths selected at a router and logic values. Namely, we map a router that selects a path with a vector $\langle h_A, l_B, \cdot\rangle$ ($\langle l_A, m_B, \cdot\rangle$) to a 1 (0) in the logic circuit, where $\cdot$ is an arbitrary value for the C value.

We now show that METRIC-DV configurations can simulate OR and NOT gates. To simplify their analysis, we allow identity elements to be used as link metric values (e.g., a link with bandwidth ∞). Even if this is not a realistic assumption, we stress that a more complex analysis would allow us to forbid identity elements as link metric values. In Fig. 7(a) and Fig. 7(b) we use the following graphical convention. Each edge has a label $\langle a, b, c\rangle$ that represents its values for metrics A, B, and C, respectively. Inside a box beside each router r, a label contains metrics in decreasing order of preferences for r. E.g., label $(A, B, C\rangle$ at router r means that r prefers metric A over B, which in turn is preferred over C.

Simulating an OR gate is easy (see Fig. 7(a)). It has two input vertices i_1 and i_2, an output vertex o_1, and a vertex r that prefers paths with higher values of A. Observe that, since each edge has identity elements for metrics A and B, metrics A and B remain unchanged when a path is propagated through the gadget. If both i_1 and i_2 select a path with vector $\langle l_A, m_B, \cdot\rangle$, then o_1 also selects a path with vector $\langle l_A, m_B, \cdot\rangle$. Otherwise, if at least one router among i_1 and i_2 selects a path with vector $\langle h_A, l_B, \cdot\rangle$, then o_1 also selects a path with vector $\langle h_A, l_B, \cdot\rangle$. We discuss backward propagation of paths after having introduced the NOT and HUB gadgets.

The NOT gadget is a more tricky (Fig. 7(b)). Let d be the unique destination vertex of the network. We make several necessary observations. First, each edge of path $(r\ s\ o_1)$ has identity elements as its metric values, hence, if a path is propagated through $(r\ s\ o_1)$, then its metric values remain unchanged. Second, vertices r, s, o_1 never select paths with metrics $\langle l_A, m_B, h_C\rangle$, $\langle h_A, l_B, \cdot\rangle$, and $\langle l_A, \cdot, l_C\rangle$, respectively, because of their metric preferences. In fact, paths with metrics $\langle l_A, h_B, l_C\rangle$, $\langle l_A, m_B, h_C\rangle$, and $\langle h_A, l_B, l_C\rangle$ are steadily available at r, s, and o_1, respectively. For this reason, r (s) selects a path with vector $\langle h_A, l_B, \cdot\rangle$ ($\langle l_A, h_B, l_C\rangle$) only if it receives it from i_1 (s). Now, we are ready to analyze the behavior of the gadget. If i_1 selects a path with vector $\langle h_A, l_B, \cdot\rangle$, which is mapped to a 1 in the logic circuit, then r selects it instead



(a) The OR gate in METRIC-DV. (b) The NOT gate in METRIC-DV.

Fig. 7. METRIC-DV gadgets that simulate (a) an OR and (b) a NOT gate.

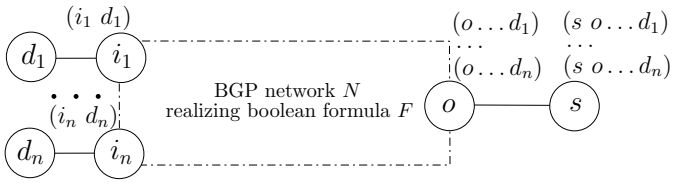


Fig. 8. Scheme of the reduction from SAT COMPLEMENT to MOAS REACHABILITY.

of the direct route to d with vector $\langle l_A, h_B, l_C \rangle$ because it prefers metric A. In turn, s selects its direct route to d with vector $\langle l_A, m_B, h_C \rangle$ instead of the one learned from r because it prefers metric B. Finally, o_1 selects the route via s, which is mapped to a 0 in the logic circuit, instead of the direct one vector $\langle h_A, l_B, l_C \rangle$ because it prefers metric C. Otherwise, if i_1 selects a path with vector $\langle l_A, m_B, \cdot \rangle$, which is mapped to a 0 in the logic circuit, then r selects the direct route to d with vector $\langle l_A, h_B, l_C \rangle$ (in fact, by monotonicity and transitivity, h_B is preferred over l_B, which, in turn, is preferred over $\langle l_B \otimes m_B \rangle$) and propagates it to s, which, in turn, selects this route over the direct one. Finally, o_1 selects its direct route to d with vector $\langle h_A, l_B, l_C \rangle$, which is mapped to a 1 in the logic circuit, instead of the one learned from s with vector $\langle l_A, h_B, l_C \rangle$. In fact, since both routes have the same value for C, the direct route has a better value for A. To construct an HUB gadget it suffices to connect two NOT gadgets in series, as we have already done in the BGP analysis.

We now discuss the issue of route backward propagations. We first analyze the NOT gadget. As we argued above, r selects a path with vector $\langle h_A, l_B, l_C \rangle$ only if it learns it from i_1. Hence, the only route that can be propagated back to i_1 is its direct path to d with vector $\langle l_A, h_B, l_C \rangle$. Two cases are possible. If i_1 selects a path with vector $\langle h_A, l_B, \cdot \rangle$, which is mapped to 1, then the direct path from r to d is not selected at i_1 and backward propagation is prevented. Otherwise, if i_1 selects a path with vector $\langle l_A, m_B, \cdot \rangle$, which is mapped to 0, then the direct path from r to d is not selected at i_1 because $l_B \otimes h_B$ is, by monotonicity, less preferred than l_B which, in turn, is less preferred than m_B. Hence, backward propagation is prevented in this case too. We now discuss the OR gadget. Since the output of an OR gadget is always connected to the input of an HUB gadget, which is the input of a NOT gadget, we are guaranteed, by the above discussion, that there is no backward propagation from the output router of the OR gadget to any of of its input routers. However, a route with vector $\langle l_A, m_B, h_C \rangle$ can be propagated from i_1 to i_2, which is connected to the output of an HUB gadget. When this route is propagated through path $(i_1\ r\ i_2)$, its metric value at o_1 becomes $\langle l_A, m_B, l_C \otimes (l_C \otimes h_C) \rangle$, which is less preferred than $\langle h_A, l_B, l_c \rangle$ (the metric of the directed path from o_1 to d). Hence, o_1 never selects a path from an input router of the OR gadget, which prevents any backward path propagation.

We draw several interesting considerations derived from this non-trivial construction of the NOT gadget. First, filtering is not a necessary construct for simulating logic circuits. This means that the SAFETY problem may be PSPACE-hard in the bounded model even if the analyzed protocol does not have filter capabilities. Second, per-neighbor and per-egress-point ranking functions, even in the absence of filters, are not necessary constructs for simulating logic circuits. In fact, our construction relies on simple popular metrics. Last, observe that if a protocol has exactly one "strict" monotonic metric, then it is guaranteed to converge to a stable state [30]. In this section, we proved that three metric are enough to make SAFETY PSPACE-hard. We think that studying the complexity of SAFETY in a protocol with two metrics is an interesting non-trivial open problem.

VI. EXTENDING THE APPROACH TO DIFFERENT DELAY MODELS

In this section, we show that our mapping technique can be used in a delay model different from the bounded one. Past work on eBGP and iBGP [3], [10], [11] assumed that messages are allowed to be delayed arbitrarily, even if not indefinitely. This means that every message must be eventually delivered, but no constraint is imposed on when the delivery occurs. We refer to this model as the *unbounded* model [31] and correspond to the bounded model where the minimum and maximum delays are set to 0 and ∞, respectively. Since Lemma 1 holds in the bounded model, all the proofs derived in Sect. III, Sect. IV, and Sect. V for building an FTM do not directly extend to this more theoretical model. In fact, as proved in [32], it is not possible to construct a circuit with memory elements using logic gates in an unbounded model. Hence, it is not possible to build a FTM. Anyway, this limitation does not prevent our mapping to be used in order to simply prove new complexity result on BGP (Thm. 4). Moreover, as a by-product of our study, we also discovered that many already known results from [3] and [10] can easily be proved with our technique (Appendix A).

For instance, consider the following problem, called MOAS REACHABILITY. Assume that a destination prefix is generated by multiple origin ASes in the eBGP network, as it happens when IP anycast is deployed in the Internet (e.g., for the DNS root name servers). The MOAS REACHABILITY problem consists in determining if the destination prefix is reachable from a given source AS for any nonempty subset of origin ASes announcing the prefix. Such a problem aims at verifying that reachability of a given MOAS (Multiple Origin AS) destination is guaranteed in presence of failures or planned maintenance. Leveraging the mapping between BGP networks

and logic gates, it is easy to prove the following theorem.

Theorem 4: MOAS REACHABILITY is coNP-hard.

Proof: The scheme of the reduction from SAT COMPLEMENT [19] is represented in Fig. 8, where nodes labeled as d_i, with $i = 1, \ldots, n$, represent the origin ASes announcing the destination prefix, and N, F, and s are defined as follows. Let F be the boolean formula in conjunctive normal form that represents an instance of SAT. Let s be a vertex that is the given source AS. We interpose between them a BGP network N which simulates the logic circuit corresponding to F. Now, considering the combination of origin ASes from which the prefix is announced corresponds to providing all possible inputs to the network N. By construction of N, this translates to considering all boolean assignments to variables in the original boolean formula F. Hence, s receives a route for each combination of origin ASes announcing the destination prefix if and only if the boolean formula F is satisfied by any boolean assignment. The statement of the theorem follows from the coNP-hardness of SAT COMPLEMENT and the fact that the reduction can be built in polynomial time with respect to the size of F. Indeed, given that the number of clauses in F is C, the number of origin ASes d_i is equal to C, and each gate in N has a constant number of nodes, each accepting at most $2 * C$ paths, because of the presence of HUB gadgets at each interconnection between gates. ■

Observe now that the above theorem holds also in the case Gao-Rexford conditions are enforced. In fact, as already discussed in Sect. IV-B, if there are no cycles in the logic circuit, it is possible to map it to a Gao-Rexford compliant BGP configuration. Hence, since the only circuit used in the proof of Thm. 4 is combinational, the MOAS REACHABILITY problem remains NP-hard even when Gao-Rexford conditions are enforced. The NP-hardness of MOAS REACHABILITY under Gao-Rexford conditions is especially interesting, because other BGP problems are polynomial in such a setting [6].

We stress that, by applying the same reduction technique, it is also straightforward to build reductions for problems like REACHABILITY, SOLVABILITY, TRAPPED, UNIQUE, and MULTIPLE [10]. Details are provided in Appendix A.

VII. CONCLUSIONS

Over the last 15 years, a routing theory has been developed to study problems on BGP convergence and route propagation. In this paper, we describe a mapping between BGP configurations and logic circuits that puts existing results in a new perspective. We show how to leverage the mapping to devise reduction techniques and define the computational complexity of several BGP routing problems under different assumptions. Most notably, by simulating Finite Turing Machines with BGP configurations, we prove the PSPACE-hardness of famous BGP problems like REACHABILITY and SAFETY in a model in which link delays are constrained into finite ranges. We also investigate the impact of restrictions to the expressiveness of the BGP policy language. We show that, under any restriction which does not guarantee convergence, BGP still retains enough expressive power to simulate arbitrary logic circuits, which in turn implies that several interesting BGP problems remain computationally intractable. Finally, we show that the mapping can be effectively used to investigate routing protocols that are very different from BGP. We found an interesting family of routing protocols, with no filter nor per-neighbor ranking constructs, for which the mapping to logic circuits is still possible.

We believe that this study raises many natural questions regarding the possibility of mapping routing configurations to logic circuits. In future work, we plan to investigate whether policy restrictions exist that do not guarantee convergence but allow efficient analysis of BGP configurations. We also plan to extend our analysis to other models and routing protocols.

REFERENCES

[1] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, 2006.

[2] T. Griffin, F. B. Shepherd, and G. Wilfong, "The stable paths problem and interdomain routing," *IEEE/ACM Trans. on Networking*, 2002.

[3] T. Griffin and G. T. Wilfong, "On the correctness of ibgp configuration," in *Proc. SIGCOMM*, 2002.

[4] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," in *Proc. SIGCOMM*, 2002.

[5] N. Kushman, S. Kandula, and D. Katabi, "Can you hear me now?! It must be BGP," in *Computer Communication Review*, 2007.

[6] L. Gao and J. Rexford, "Stable internet routing without global coordination," in *Proc. SIGMETRICS*, 2000.

[7] A. Flavel and M. Roughan, "Stable and Flexible iBGP," in *Proc. SIGCOMM*, 2009.

[8] L. Cittadini, M. Rimondini, S. Vissicchio, M. Corea, and G. D. Battista, "From theory to practice: Efficiently checking bgp configurations for guaranteed convergence," *IEEE Trans. Netw. and Serv. Man.*, 2011.

[9] A. Flavel, J. McMahon, A. Shaikh, M. Roughan, and N. Bean, "BGP Route Prediction within ISPs," *Comput. Commun.*, vol. 33, 2010.

[10] T. G. Griffin and G. Wilfong, "An Analysis of BGP Convergence Properties," in *Proc. SIGCOMM*, 1999.

[11] R. Sami, M. Schapira, and A. Zohar, "Searching for stability in interdomain routing," in *Proc. INFOCOM*, 2009.

[12] L. Cittadini, G. D. Battista, M. Rimondini, and S. Vissicchio, "Wheel + ring = reel: the impact of route filtering on the stability of policy routing," in *Proc. ICNP*, 2009.

[13] M. Chiesa, L. Cittadini, G. Di Battista, and S. Vissicchio, "Local Transit Policies and the Complexity of BGP Stability Testing," in *Proc. INFOCOM*, 2011.

[14] M. Suchara, A. Fabrikant, and J. Rexford, "Bgp safety with spurious updates," in *Proc. INFOCOM*, 2011, pp. 2966–2974.

[15] A. Fabrikant and C. Papadimitriou, "The complexity of game dynamics: Bgp oscillations, sink equilibria, and beyond," in *Proc. SODA*, 2008.

[16] A. D. Jaggard, M. Schapira, and R. N. Wright, "Distributed computing with rules of thumb," in *PODC*, vol. 8, no. 4, December 2011, pp. 387–400.

[17] R. Engelberg, A. Fabrikant, M. Schapira, and D. Wajc, "Best-response dynamics out of sync: complexity and characterization," in *EC*, 2013.

[18] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica, "Pathlet routing," in *Proc. SIGCOMM*, 2009.

[19] C. Papadimitriou, *Computational complexity*. Addison-Wesley, 1994.

[20] E. G. Friedman, "Clock distribution networks in synchronous digital integrated circuits," in *Proc. IEEE*, 2001, pp. 665–692.

[21] P. Linz, *An Introduction to Formal Language and Automata*. USA: Jones and Bartlett Publishers, Inc., 2006.

[22] M. Chiesa, L. Cittadini, G. Di Battista, L. Vanbever, and S. Vissicchio, "Computing with BGP: from Routing Configurations to Turing Machines," Université Catholique de Louvain, Tech. Rep., 2012.

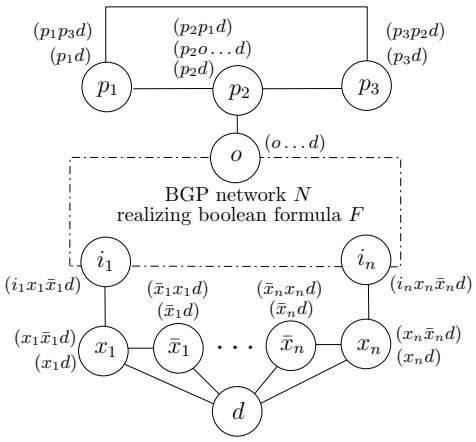


Fig. 9. Construction for REACHABILITY, SOLVABILITY, and TRAPPED.

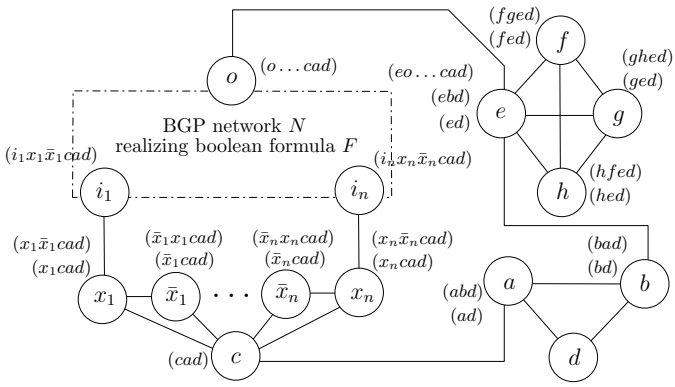


Fig. 10. Scheme of the reduction from SAT to UNIQUE and MULTIPLE.

[23] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[24] M. R. Prasad, A. Biere, and A. Gupta, "A survey of recent advances in sat-based formal verification," *STTT*, vol. 7, no. 2, pp. 156–173, 2005.

[25] S. Wolfram, *A new kind of science*. Wolfram Media Inc., 2002.

[26] T. Bates, E. Chen, and R. Chandra, "BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP)," RFC 4456, 2006.

[27] S. Vissicchio, L. Cittadini, L. Vanbever, and O. Bonaventure, "iBGP Deceptions: More Sessions, Fewer Routes," in *Proc. INFOCOM*, 2012.

[28] D. Savage, D. Slice, J. Ng, S. Moore, and R.White, "Enhanced Interior Gateway Routing Protocol," 2013.

[29] J. S. Baras and G. Theodorakopoulos, *Path Problems in Networks*. Morgan & Claypool Publishers, 2010.

[30] T. Griffin and J. L. Sobrinho, "Metarouting," in *Proc. SIGCOMM*, 2005.

[31] J. T. Udding, "A formal model for defining and classifying delay-insensitive circuits and systems," *Distributed Computing*, 1986.

[32] A. J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," ser. AUSCRYPT, 1990.

APPENDIX

A. Reductions for the Unbounded Delay BGP Model

In the following, we show how to leverage the mapping between eBGP configurations and logic gates to prove the complexity of the problems studied in [10] in the unbounded asynchronous model. All the following reductions can be built in polynomial time with respect to F (see Section II-C). Also note that a DISAGREE can be obtained as a loop of two NOT gadgets. As a consequence, the following complexity proofs remain valid whenever the OR and the NOT logic gates can be simulated and arbitrarily interconnected via the HUB gadget.

1) Reachability, Solvability, and Trapped: The REACHABILITY problem consists in deciding if a BGP network admits a stable state in which a given router s has a route to a given destination d. The problem has been already shown in [10] to be NP-hard. We now show an NP-hardness proof exploiting the mapping between BGP configurations and logic gates.

Theorem 5: REACHABILITY is NP-hard.

Proof: Let F be a boolean formula in conjunctive normal form that represents an instance of SAT. We build an instance of REACHABILITY as follows (see Fig. 9). Let o and d be the source and the destination vertices considered in REACHABILITY, respectively. We interpose between them a BGP network N which simulates the logic circuit corresponding to F. We also add a DISAGREE gadget between d and each input router i_j in N. Each DISAGREE gadget can converge to one of two distinct stable states. Each possible combination of these stable states is mapped to a boolean assignment M of the variables. For each of these combinations, by construction of N, node o will have a route to d if and only if F is satisfied by M. ■

We now consider the SOLVABILITY and TRAPPED problems that deal with convergence guarantees of a BGP configuration. Namely, SOLVABILITY consists of deciding if a given BGP configuration admits at least one stable solution, while TRAPPED is the problem of deciding if the network can be trapped in permanent routing oscillations, like those occurring in a BAD-GADGET. For both problems, we use the reduction shown in Fig. 9, where a BAD-GADGET between nodes p_1, p_2, and p_3 is added. The BAD-GADGET does not oscillate if and only if p_2 steadily receives path $(o\dots d)$ from o. However, since this requires to solve the REACHABILITY problem, both SOLVABILITY and TRAPPED are proved to be NP-Hard.

2) Unique and Multiple: UNIQUE (MULTIPLE) is the problem of deciding if a single (more than one) stable state exists for a given BGP configuration. In the reduced instance in Fig. 10 the presence of a stable state is guaranteed in one of the two stable states of the DISAGREE between a and b. Indeed, if a steadily selects $(a\ b\ d)$ and b steadily selects $(b\ d)$, then no route is provided to c, which implies o having no route, and e selecting $(e\ b\ d)$. This, in turn, implies nodes f, g, and h having no route to d. However, if a steadily selects $(a\ d)$ and b steadily selects $(b\ a\ d)$, then the presence of additional stable states depends on the formula F. Indeed, if F is not satisfiable then o has no steady route. In this case, e selects $(e\ d)$ activating the BAD-GADGET between f, g, and h. As a result, no other stable state exists. Otherwise, if F is satisfiable, then o steadily select a route for at least one combination of inputs to N. This implies that e selects the route it receives from o, hence preventing the BAD-GADGET between f, g, and h from oscillating, and forcing a second stable state. This proves UNIQUE is NP-Hard, which directly implies the NP-hardness of MULTIPLE, as already noted in [10].