Guaranteeing Stability and Delay in Dynamic Networks based on Infinite Games

Simon Tenbusch, Christof Löding, Frank Radmacher
Department of Computer Science
RWTH Aachen University, Germany
Email: simon.tenbusch@rwth-aachen.de

James Gross
School of Electrical Engineering
KTH Royal Institute of Technology, Sweden
Email: james.gross@ee.kth.se

***Abstract*—We study stability and delay in dynamic networks under adversarial conditions. Adversarial conditions are mandatory in establishing deterministic performance guarantees in networks. Under this framework, we concentrate on the general stability region for a network, i.e. without specifying the routing algorithm. This is in contrast to related work for adversarial network conditions, where usually the backpressure routing algorithm is considered. Our work consists of four novel contributions: (1) We present a novel analysis model which is based on the theory of infinite two-player games; (2) Using this approach, we can characterize the stability region of networks under adversarial conditions for arbitrary routing schemes; (3) We determine conditions under which a delay bound for packet forwarding under adversarial conditions exists; (4) We provide a backtracking algorithm which determines in a model-checking fashion network stability. The backtracking algorithm is furthermore shown to reduce the computational effort significantly for practical scenarios.**

I. Introduction

Among the more fundamental research problems addressed recently by the networking community are the investigations with respect to *stability* of networks. Stability is usually defined as the circumstance that the node queues within a network are bounded under certain variations of the traffic and/or the link capacities between the nodes. Over the last two decades a rich set of results has been achieved here with respect to various different models and conditions. These results have almost all been established regarding the backpressure routing algorithm as (for example) investigated in an early contribution by Tassiulas [1].

This seminal work of Tassiulas studied network stability under random stochastic arrival processes but with fixed link capacities and showed that under certain conditions on the arrival rate of the flows the network stays stable if backpressure routing is the algorithm for forwarding data between the nodes. This initial result with respect to stochastic variations of the network arrivals was significantly extended by Neely et al. [2] where the *network stability region* was established for networks with stochastic link capacity variations. Subsequently, further extensions of the stochastic model were presented. However, while these stochastic variations are well justified from a practical point of view, these models also come with disadvantages. Most strikingly, from a practical point of view an application or a network provider might be interested in guarantees with respect to stability or the end-to-end delay, which cannot be studied based on stochastic models.

This perspective motivates to consider adversarial scenarios, where malicious agents actively degrade network performance. One of the first contributions in this line was provided by Borodin et al. in [3] where network stability of backpressure routing was investigated for adversarial conditions. Since then, several contributions have extended the original models and results [4], [5], [6]. Most recently, two contributions from Andrews et al. [7], [8] addressed the stability of backpressure routing for dynamic networks (i.e., networks with varying link capacities) where an adversary injects multi-commodity traffic into the network. The papers show that as long as the adversary injects packets such that the network load is subcritical, backpressure routing is stable. Subcritical load is defined in this context by an $A(w,\varepsilon)$ adversary that operates as follows [7] : For any time t let $I^{[t,t+w-1]}$ be the set of packets injected during the w time steps from t to $t+w-1$. Then the adversary can associate with each packet $p \in I^{[t,t+w-1]}$ a simple path Γ_p from s_p to d_p such that for all $e \in E$ we have $\sum_{p \in I^{[t,t+w-1]}, e \in \Gamma_p} l_p \leq (1-\varepsilon) \sum_{i=t}^{t+w-1} c_e(i)$. E is here the set of edges of a directed graph which models the network, l_p is the size of packet p and $c_e(t)$ is the capacity of edge e at (slotted) time t (such that any packet with a smaller size than the capacity can be transmitted over the edge).

These insights with respect to network stability under adversarial conditions come with some limitations. First of all, they are restricted to backpressure routing. While the analysis techniques (namely arguing about the maximum drift of backlogged queues) are well established, we lack a more fundamental insight into network stability under any routing scheme in adversarial conditions. Moreover, the usage of the $A(w,\varepsilon)$ adversary is not easily transferred to practical scenarios, i.e. if we are given a set of traffic streams and some possible configurations for the capacity variations, one needs to simulate the operation of the network in order to determine if at each time slot t routes can be found with enough capacity - or not. Instead, a more assessable criterium for network stability would be of interest. What is missing is a more fundamental insight into the stability region under adversarial conditions, possibly providing as much insight into the case of adversarial settings as achieved by Neely et al. [2] for the case of stochastic variations of network conditions.

In this paper, we address this open issue. We provide four novel contributions:

- We present an infinite games based model which is used to analyze adversarial dynamic networks under critical and subcritical load. This modeling and analysis method is completely different to the usual analysis technique for network stability, where the drift of backlogged queues is considered.
- Based on this approach we obtain conditions for network stability, as well as a bound on the number of packets in dynamic networks for critical load. These conditions essentially define the network stability region under adversarial conditions regardless of the routing scheme.
- Furthermore, we provide a delay bound for dynamic networks under subcritical load.
- Finally, we present an algorithm that checks given instances of dynamic networks for stability. While our theoretical results indicate that exponentially many network conditions need to be considered in the worst case to determine network stability, we present a backtracking approach which considers only a tiny fraction of these conditions for rather practical instances. This leads in a numerical example to quite acceptable run times.

The remaining paper is structured as follows. In Section II we present our system model and problem statement, summarize some background information on infinite games and provide a mapping of the system model and problem statement to a play in infinite games. Section III contains our main results as well as all of the proofs. We continue with the presentation of our model-checking algorithm for network stability in Section IV and demonstrate its application for a simple example. Finally, we conclude the paper in Section V.

II. Preliminaries and Background

A. Basic Model

We consider a communication network represented by a directed graph $G = (V, E)$ with a set of nodes V and $E \subseteq V \times V$ being the set of mutual direction communication links, i.e., $(u, v) \in E \Leftrightarrow (v, u) \in E$. Time is split into slots with index t. Each node has up to $N \in \mathbb{N}$ independent channels available for communication. However, the number of available channels per node can vary from slot to slot. We model this varying capacity by defining a set $\mathcal{B} := \{B_1, B_2, ..., B_b\}$ of *blocking functions* $B_i : V \to \{0, ..., N\}$. Hence, $N - B(u)$ channels are available for communication at the current time slot at node u. For each available channel a node can forward one packet (error-free) to some neighboring node during one slot[1].

Furthermore, we consider a set $\mathcal{D} = \{(s_1, d_1, \lambda_1), ..., (s_m, d_m, \lambda_m)\}$ of traffic flows to be conveyed by the network. Each flow is specified by a source and destination $s, d \in V$ as well as an intensity λ which denotes the (maximum) number of newly generated packets per slot for this flow. We refer to packets belonging to a particular source-destination pair (s_j, d_j) as *commodity* j.

In the following, we are interested in conditions on the blocking functions B under which the network is *guaranteed* to be stable given the set of traffic flows $\mathcal{D}$. If stability can be reached, we are furthermore interested in delay guarantees for the flows. Note that we do not restrict ourselves to any routing algorithm. Instead, we are interested in general conditions for stability and finite delay.

B. Infinite Two-Player Games

The interest in guaranteed stability and delay requires us to turn to an adversarial approach. We utilize in the following infinite two-player games as novel tool for the adversarial analysis. Infinite two-player games have been studied first by set theorists [9], [10]. Their algorithmic theory, however, has its origin in the synthesis of digital circuits [11]. The basic idea is straightforward: A system (called controller in the following), which has to produce an infinite output sequence Y, plays in a turn-based game against an environment, which provides an infinite input sequence X. This interaction produces a sequence $\rho = X_1 Y_1 X_2 Y_2 ...$ where the goal of the controller is to assure a requirement which is checkable on ρ. The question arises if for such a two-player game we can decide at all if the requirement can be met by the controller, and if so, if we can synthesize a strategy that the controller should apply. It has been shown that if the requirement is ω-regular [12], one can determine algorithmically whether the controller can satisfy the requirement. In this case, one can furthermore compute a strategy for the controller in the form of a finite automaton. In order to do so, one transforms the described game to a game on a finite, directed graph. Each vertex of this game graph represents a position of the game; it corresponds either to a chosen input or a chosen output. Hence, the vertex set Q of the game graph is partitioned into a set Q_0 of the controller and a set Q_1 of the environment. At each vertex of Q_1 the environment chooses an outgoing edge to a successor vertex of Q_0, which represents the chosen input vector; and at each vertex of Q_0 the controller chooses an edge to a successor vertex of Q_1, which represents the chosen output vector. Likewise, the requirement is transferred to a so-called *winning condition* for the controller on the game graph. For ω-regular requirements, the winning condition can be described by sets of vertices (on a bigger graph) which eventually occur or which occur infinitely often. Reformulating the game and the winning condition in form of the game graph allows now the application of standard analysis techniques to determine conditions for winning the game, as well as for synthesizing a winning strategy as there exist various algorithms that solve games. For a more detailed introduction to the theory of infinite games we refer to the book [12] and the tutorial [13].

C. Routing Game Formulation

We now introduce the detailed game formulation. We refer to the controller in the following as routing agent and to the environment as demand agent. The *routing game* $\mathcal{G}$ is given as

[1] We consider in the following a so called node-centric model. Nevertheless, all our results can easily be transferred to a edge-centric model.

the tuple $\mathcal{G} = (G, \mathcal{D}, \mathcal{B})$, where G is the connectivity graph, $\mathcal{D}$ the set of traffic flows, and $\mathcal{B}$ the set of blocking functions. For the game to be played, we need to define the game graph $H = (Q, T)$. In the game graph, the positions Q describe the current network state and the transitions $T \subseteq Q \times Q$ are given by all possible choices of routing agent and demand agent. Q is furthermore partitioned into the positions Q_0 and Q_1 of routing and demand agents next turn. A network state $q \in Q_i$ for $0 \leq i \leq 1$ is given as a tuple $(i, B, \mathcal{P})$. i indicates which player's turn comes next, 0 (1) indicating routing (demand) agent's turn. B is the currently active blocking function and $\mathcal{P}$ is the set of packets currently in the network. Each packet is represented as a tuple $(id, u, j) \in \mathbb{N} \times V \times \{1, ..., m\}$ where id is a unique identifier, u is the node where the packet is stored, and j is the commodity.

The game starts in the position $q_1 = (1, -, \emptyset)$ and evolves then by routing and demand agent turns. A turn of routing agent consists of three steps: a) λ_j new packets for every commodity j are inserted at the corresponding source nodes; b) Routing agent transfers packets from each node to neighboring nodes respecting the capacity limit at node u being $N - B(u)$; c) Packets arriving at their destination are removed. These steps lead then to a new position in the game graph $(1, B, \mathcal{P}')$. It is then demand agent's turn with the sole step of choosing a new blocking function $B' \in \mathcal{B}$ which leads to the position $(0, B', \mathcal{P}')$. Overall, the game results in a play $\rho = q_1, q_2, ...$ where $q_i \in Q_0$ if i is even and $q_i \in Q_1$ otherwise.

Initially, we consider the stability of the network as winning condition for routing agent: *Routing agent wins if there exists a bound $k \in \mathbb{N}$ such that no position q_i in the play exists with $|\mathcal{P}| > k$.* We are primarily interested in the conditions under which routing agent is guaranteed to win the game. However, for this we also need to construct winning strategies, i.e. routing schemes that are applied by routing agent. Formally, a *winning strategy* for routing agent defines for any history of the play $q_1, q_2, ..., q_{2i}$ for $i \in \mathbb{N}$ which position q_{2i+1} to choose next, such that the play is assured to fulfill the winning condition. However, the type of winning condition that we use is rather simple: routing agent just has to stay inside the "safe" area of the game graph. Such conditions are referred to as safety conditions. For the resulting class of safety games it is known [12] that the player who has a winning strategy also has a winning strategy which is memoryless, i.e. the choice of the next position depends only on the current position in the game graph. The existence of memoryless strategies makes it much easier to argue about the behavior of the players.

III. Main Contributions

A. Stability Results

We characterize the games $\mathcal{G} = (G, \mathcal{D}, \mathcal{B})$ which can be won by routing agent, i.e., a winning strategy for her exists. For this, we establish a connection to multicommodity flows for which we rehearse here for convenience the definition:

Definition 1. Given a connectivity graph G, a blocking function B, and demand set $\mathcal{D}$, a multicommodity flow (MCF) f with throughput λ_j for each commodity j is a tuple of flows $f = (f_1, ..., f_m)$ such that for each flow $f_j : E \to \mathbb{Q}^+$ with $1 \leq j \leq m$ the flow equations hold:

$$\sum_{v:(v,u)\in E} f_j(v,u) = \sum_{w:(u,w)\in E} f_j(u,w) \quad \forall u \in V \setminus \{s_j, d_j\} \tag{1}$$

$$\sum_{u:(u,d_j)\in E} f_j(u,d_j) = \lambda_j \tag{2}$$

$$\sum_{u:(d_j,u)\in E} f_j(d_j,u) = 0 \tag{3}$$

$$f_j(u,v) \geq 0 \ \forall (u,v) \in E \tag{4}$$

$$\sum_{j=1}^{m} \left(\sum_{v:(u,v)\in E} f_j(u,v) \right) \leq N - B(u) \ \forall u \in V \tag{5}$$

Note that the last condition assures that the sum of flows respects the number of free channels on each node.

We initially consider the special case where the capacities in the network are static, i.e., $\mathcal{B} = \{B\}$. Thus, demand agent cannot influence the play, and the outcome of the game depends solely on the choices of routing agent.

Lemma 1. *In the game $\mathcal{G} = (G, \mathcal{D}, \mathcal{B})$ with $\mathcal{B} = \{B\}$ routing agent has a winning strategy if and only if there exists an MCF $f = (f_1, ..., f_m)$ with throughput λ_j for each commodity j under blocking function B.*

Proof: "⇒": Assume routing agent plays according to a winning strategy. By definition, the number of packets in the network then never exceeds k. Thus, all positions in the game graph where $|\mathcal{P}| > k$ can be replaced with a 'sink state' position q_{sink}. As k is a bound on the number of packets, all packet identifiers are from the range $\{1, ..., k\}$ (if new packets enter the network, some free identifiers are reused). Overall, the modified game graph has only finitely many positions. The winning condition for routing agent is to avoid ever visiting q_{sink}. Assume routing agent to play according to a memoryless strategy. Due to the static blocking function and the memoryless strategy, the play will enter a loop as soon as a position is seen twice. This must happen at some point as the game graph is finite. Thus, there are sequences of positions r, s such that the play ρ has the form $\rho = r \cdot s^\omega$ [2], where $s = s_1, ..., s_l$. At $r \cdot s^i$, the game is at the same position s_l as at $r \cdot s^{i+1}$. Let $n_j^h(u,v)$ be the number of packets of commodity j that routing agent sends at position s_h from node u to node v according to the winning strategy. Consider the following tuple $f = (f_1, ..., f_m)$ with $f_j : E \to \mathbb{Q}^+$ where $f_j(u,v) := \frac{1}{l} \sum_{h=1}^{l} n_j^h(u,v) \ \forall (u,v) \in E, j = 1 ... m$. We show that f is in fact an MCF with throughput λ_j for each commodity j, i.e., it satisfies the constraints (1) to (5). The play at $\rho = r \cdot s^i$ and $\rho = r \cdot s^{i+1}$ is at position s_l which means the number of packets of commodity j received at each node u during $s_1, ..., s_l$ is equal to the number of packets sent, i.e.,

[2] s^ω denotes the infinite repetition of s, similarly s^i denotes the repetition of s exactly i times.

$\sum_{h=1}^{l}\sum_{v:(v,u)\in E} n_j^h(v,u) = \sum_{h=1}^{l}\sum_{w:(u,w)\in E} n_j^h(u,w)$. Hence,

$$\begin{aligned}\sum_{v:(v,u)\in E} f_j(v,u) &= \frac{1}{l}\sum_{h=1}^{l}\left(\sum_{v:(v,u)\in E} n_j^h(v,u)\right)\\ &= \sum_{w:(u,w)\in E} f_j(u,w).\end{aligned}$$

During one iteration of s, exactly $l\lambda_j$ packets of commodity j are generated. Hence, $l\lambda_j$ packets have been delivered to d_j, i.e., $\sum_{h=1}^{l}\sum_{u:(u,d_j)\in E} n_j^h(u,d_j) = l\lambda_j$ and

$$\begin{aligned}\sum_{u:(u,d_j)\in E} f_j(u,d_j) &= \frac{1}{l}\sum_{h=1}^{l}\left(\sum_{u:(u,d_j)\in E} n_j^h(u,d_j)\right)\\ &= \lambda_j.\end{aligned}$$

Packets that have reached their destination are immediately removed. Therefore, we have $\sum_{u:(d_j,u)\in E} f_j(d_j,u) = 0$. Because $n_j^h(u,v) \geq 0$ we obtain $f_j(u,v) \geq 0$. Finally, the strategy of routing agent clearly obeys the number of free channels. Thus, overall f is an MCF with throughput λ_j for each commodity j.

"$\Leftarrow$": Assume $f = (f_1, \dots, f_m)$ is an MCF with throughput λ_j for each commodity j on G with blocking function B. We construct a strategy σ_B and show that it is a winning strategy for routing agent. For the construction of the strategy, we define a list L_u for each node u. The strategy σ_B uses these lists to make the routing decisions.

For $u \in V$ we denote the neighbors of u by $v_1, \dots, v_p$ by fixing an arbitrary order. Let K be the lowest common multiple of the denominators of all flow values appearing in f. For each node $u \in V$, the list L_u consists of tuples $(j,v) \in \{1,\dots,m\}\times V$. L_u contains $K \cdot f_j(u,v)$ copies of the tuple (j,v) for each outgoing edge $(u,v) \in E$ and each commodity j. The total number of these entries is $\sum_{j=1}^{m}(K \cdot f_j(u)) \leq K\cdot(N - B(u))$. The list is filled up with additional *null*-entries so that it has size exactly $K \cdot (N - B(u))$. Hence, L_u has the following form:

$$\begin{aligned}&[\underbrace{(1,v_1),\dots,(1,v_1)}_{K\cdot f_1(u,v_1)\text{ times}},\dots\underbrace{(1,v_p),\dots,(1,v_p)}_{K\cdot f_1(u,v_p)\text{ times}},\dots\\ &\underbrace{(m,v_1),\dots,(m,v_1)}_{K\cdot f_m(u,v_1)\text{ times}},\dots\underbrace{(m,v_p),\dots,(m,v_p)}_{K\cdot f_m(u,v_p)\text{ times}},\\ &(\textit{null}),\dots,(\textit{null})]\end{aligned}$$

The strategy σ_B works now in rounds of K time steps: For each node u, the list L_u is traversed, starting at the beginning. For each time step the $N - B(u)$ next entries are selected and for each entry (j,v) the oldest packet of commodity j is forwarded to v. If no such packet is stored at u, nothing is sent. In the next time step, the next $N - B(u)$ entries of the list are selected, and packets are sent correspondingly. Within K time steps, the end of the list is reached, as $|L_u| = K\cdot(N - B(u))$. After a round of K time steps is completed, the next round of K time steps starts and again, the list is traversed from the beginning. We show that the number of packets in u remains bounded. After a round of K time steps, at most $K \cdot f_j(u)$ packets of commodity j have arrived from incoming transmissions at u. In the next round of K time steps, $K\cdot f_j(u)$ packets of commodity j are forwarded by u (or less if less packets are available at u). Again within this round, at most $K \cdot f_j(u)$ new packets have arrived at u which are forwarded in the next round. Hence, at any point in time, there are at most $2K \cdot f_j(u)$ packets of commodity j at u, which means the total number of packets in the network remains bounded by $2K\sum_{j=1}^{m}\sum_{v\in V} f_j(v)$. Hence, σ_B is a winning strategy. ∎

Next, we expand the characterization to the general case with non-static capacities. Hence, demand agent may choose from a set of blocking functions.

Theorem 2. *In the game $\mathcal{G} = (G, \mathcal{D}, \mathcal{B})$ routing agent has a winning strategy if and only if there exists an MCF $f^B = (f_1^B, \dots, f_m^B)$ with throughput λ_j for each commodity j on G for each blocking function $B \in \mathcal{B}$.*

Proof: "$\Rightarrow$": Given that routing agent has a winning strategy, assume there exists a blocking function $B \in \mathcal{B}$ for which no MCF exists. Then demand agent can play B^ω, i.e., he can always choose this blocking function. Lemma 1 states, that in this case, demand agent wins. This is a contradiction to the winning strategy of routing agent.

"$\Leftarrow$": Given there is an MCF for each $B \in \mathcal{B}$, we construct a winning strategy σ for routing agent: For each $B \in \mathcal{B}$ consider the strategy σ_B as defined in the proof of Lemma 1. σ behaves according to σ_B whenever blocking function B is active. If demand agent switches to some other blocking function and later returns to B, σ continues to behave like σ_B, memorizing the current list positions for each σ_B. When blocking function B is active, all packets that have been generated while other blocking functions were active are ignored and never forwarded, even if they are the oldest packets of that commodity. As shown in the proof of Lemma 1, the number of packets in the network for each σ_B is bounded by[3] $R^B = 2K^B\sum_{j=1}^{m}\sum_{v\in V} f_j^B(v)$. Hence, the total number of packets is bounded by $R = \sum_{B\in\mathcal{B}} R^B$, when playing according to σ. This means, σ is a winning strategy. ∎

The contribution of Theorem 2 and its proof is not so much the routing strategy, which requires global knowledge at all nodes and is therefore infeasible for practical applications. However, with Theorem 2 we have obtained a characterizing property of routing agent winning the stability game: It suffices if there exists a suitable MCF for each blocking function. If we are given the set $\mathcal{B}$ (or if we can summarize it in a reasonable way), we can check for a given traffic specification $\mathcal{D}$ if stability can be achieved *at all*, independent of any specified routing algorithm. Hence, Theorem 2 provides a tight and checkable characterization of the capacity region of the network under adversarial conditions. We comment in

[3]We denote by K^B the smallest common multiple of the denominators of the flow values occurring in the flow for blocking function B, f^B is the corresponding MCF to blocking function B.

Section IV-B on the run-time of checking stability of larger networks.

B. Delay Results

The stability characterization above does not in general yield worst-case end-to-end delays, as packets may be held back in the network for arbitrary time spans, e.g. when a node is completely blocked for an extended period of time. Nevertheless, it is possible to specify a delay bound that is guaranteed to hold for at least some fraction of the traffic as Lemma 3 states.

Lemma 3. *If routing agent plays according to σ from Theorem 2 then at any point in the play and for each $\alpha \in \mathbb{N}^0$, a fraction of at most $\frac{1}{\alpha+1}$ of the packets have a delay greater than $\alpha \cdot \frac{S_j}{\lambda_j}$ for each commodity j, with $S_j := 2\sum_{B\in\mathcal{B}}\left(\sum_{v\in V} K^B \cdot f_j^B(v)\right)$.*

Proof: We consider the sum of delays over all packets of commodity j: A packet can only get delayed, as long as it has not been delivered. When routing agent plays according to σ, the number of packets of commodity j in the network is bounded by $S_j = 2\sum_{B\in\mathcal{B}}\left(\sum_{v\in V} K^B \cdot f_j^B(v)\right)$. Since packets can get delayed only if they are not delivered yet, the sum of delays of those packets is increased in each time step by at most S_j. Hence, for any time step t, the total delay for these packets is at most $t \cdot S_j$. Furthermore, at time step t the total number of packets (in the network or already delivered) for commodity j is $t\lambda_j$. Therefore for the (worst-case) average packet delay $\bar{d}$ it holds: $\bar{d} \leq t \cdot S_j \cdot \frac{1}{t\lambda_j} = \frac{S_j}{\lambda_j}$ independently of the time step t. Consider $\alpha \in \mathbb{N}^0$ and assume there is a packet with delay $d > \alpha \cdot \frac{S_j}{\lambda_j}$. As the minimum delay of a packet is 0, there have to be at least α packets with a delay $< \bar{d}$. Therefore the fraction of packets exceeding a delay bound of $\alpha \cdot \frac{S_j}{\lambda_j}$ is at most $\frac{1}{1+\alpha}$. ∎

Next, we consider conditions under which all packets can be guaranteed an eventual delivery and therefore allow for upper-bounding the worst-case end-to-end delay. However, for this we require somewhat more reliable network conditions than for pure stability: For each blocking function, there has to be an MCF with throughput slightly larger than λ_j for each commodity j. Furthermore, some nonzero capacity is needed for each node and each blocking function. Without such an assumption on the reliability of capacity no delay guarantees can be given in the adversarial setting as the demand agent could trap packets in some part of the network indefinitely. Formally, this leads to:

Theorem 4. *Let $\varepsilon > 0$ and let $\mathcal{G} = (G, \mathcal{D}, \mathcal{B})$ be a game with demand $\mathcal{D}$ where an MCF $f^B = (f_1^B, \dots, f_m^B)$ exists with throughput $\lambda_j + \varepsilon$ for each commodity j and each blocking function B. Furthermore, let $N - B(u) \geq 1$ for all $u \in V$ and $B \in \mathcal{B}$. Then routing agent has a winning strategy σ' such that all packets of commodity j are delivered within at most D_j time steps.*

Proof: Notice that we only have one direction to show. We construct in the following a routing strategy which forwards packets based on two different principles. Initially, *regular* packets are forwarded on routes equal to the static routing strategy of Theorem 2. However, under certain conditions packets are converted into *stuck* packets. For the forwarding of these we utilize the spare capacity ε. We distinguish the forwarding of regular and stuck packets by different flows and start with discussing properties of the flows for the forwarding of the stuck packets. We assume without loss of generality, that the network consists of a single connected component. For each commodity j and all nodes $v_1 \in V$ with $v_1 \neq d_j$ we fix an arbitrary shortest path $P_{(v_1,d_j)} = \{(v_1, v_2), (v_2, v_3) \dots (v_k, d_j)\}$ from v_1 to d_j. Let

$$p_{(v_1,d_j)}(u,w) := \begin{cases} 1 & \text{if } u \neq d_j \text{ and } (u,w) \in P_{(v_1,d_j)} \\ 0 & \text{otherwise.} \end{cases}$$

Given some $\delta > 0$ (to be specified below), we define a tuple of functions $f' = (f'_1, \dots, f'_m)$ on the network by setting for each commodity j and $(u,w) \in E$:

$$f'_j(u,w) := \delta \sum_{v_1 \in V} p_{(v_1,d_j)}(u,w).$$

Note that f' does not define a flow because the flow conservation property is not given, as $\sum_{v:(u,v)\in E} f'_j(u,v) = \sum_{v:(v,u)\in E} f'_j(v,u) + \delta$ for each $u \in V, 1 \leq j \leq m$ with $u \neq d_j$. This equality holds because each node $u \neq d_j$ has an outgoing edge for the path $P_{(u,d_j)}$ for which there is no incoming edge.

Denote the minimal capacity in the network by $c_{\min} := \min_{B\in\mathcal{B}, u\in V} N - B(u)$ and note that as long as $\delta \leq \frac{c_{\min}}{m|V|}$ the flow f' respects the minimum capacity $c_{\min}$ as $\sum_{v_1\in V} p_{(v_1,d_j)}(u,w)$ is upper bounded by $m|V|$. Finally, note that f' spans over the entire network.

Now we turn to the MCFs in Theorem 4. Fix initially $B \in \mathcal{B}$. By assumption f^B has throughput $\lambda_j + \varepsilon$ for each commodity j. Let $\bar{f}^B$ be the normalized MCF, i.e., $\bar{f}_j^B(u,v) := \frac{\lambda_j}{\lambda_j+\varepsilon} f_j^B(u,v)$ for all $(u,v) \in E, 1 \leq j \leq m$. Then $\bar{f}^B$ has throughput λ_j for each commodity j. Furthermore under $\bar{f}^B$, there is free capacity of at least ε left on each node. Hence, the combined function F^B, defined by $F_j^B(u,v) := \bar{f}_j^B(u,v) + f'_j(u,v)$ for all $u, v, \in V, 1 \leq j \leq m$ respects the capacity restrictions of the nodes in case that $\delta \leq \frac{\varepsilon}{m|V|}$. Based on F^B we derive now the following routing strategy in analogy to Lemma 1: Define a list L_u^B for each node u. Let $\delta := \min\left(\frac{\varepsilon}{m|V|}, \frac{c_{\min}}{m|V|}\right)$. For $u \in V$ we denote the neighbors of u by $v_1, \dots, v_p$ by fixing an arbitrary order. Let K^B be the lowest common multiple of the denominators of δ and all values appearing in $\bar{f}^B$ and f'. The list L_u^B contains tuples of the form $(j, v, B/1) \in \{1, \dots, m\} \times V \times \{B, 1\}$ as follows: For each outgoing edge $(u,v) \in E$ and each commodity j, the list contains $K^B \cdot \bar{f}_j^B(u,v)$ copies of the tuple (j, v, B). Additionally, for each outgoing edge $(u,v) \in E$ and each commodity j, the list contains $K^B \cdot f'_j(u,v)$ copies of the tuple $(j, v, 1)$. Finally, fill up the list with *null*-entries so that the length of the list is exactly $K^B \cdot (N - B(u))$. Overall,

L_u^B is an extended version of the list in Lemma 1 with the following structure:

$$[\underbrace{(1,v_1,B),\dots,(1,v_1,B)}_{K^B\cdot\bar{f}_1^B(u,v_1)\text{ times}},\dots\underbrace{(1,v_p,B),\dots,(1,v_p,B)}_{K^B\cdot\bar{f}_1^B(u,v_p)\text{ times}},\dots$$
$$\underbrace{(m,v_1,B),\dots,(m,v_1,B)}_{K^B\cdot\bar{f}_m^B(u,v_1)\text{ times}},\dots\underbrace{(m,v_p,B),\dots,(m,v_p,B)}_{K^B\cdot\bar{f}_m^B(u,v_p)\text{ times}},$$
$$\underbrace{(1,v_1,1),\dots,(1,v_1,1)}_{K^B\cdot f_1'(u,v_1)\text{ times}},\dots\underbrace{(1,v_p,1),\dots,(1,v_p,1)}_{K^B\cdot f_1'(u,v_p)\text{ times}},\dots$$
$$\underbrace{(m,v_1,1),\dots,(m,v_1,1)}_{K^B\cdot f_m'(u,v_1)\text{ times}},\dots\underbrace{(m,v_p,1),\dots,(m,v_p,1)}_{K^B\cdot f_m'(u,v_p)\text{ times}},$$
$$(\textit{null}),\dots,(\textit{null})]$$

With the construction of this list, we obtain the strategy σ'_B which is played while B is active as follows: Each newly inserted packet is a *regular* packet and is marked with a B. Regular packets are forwarded via the $\bar{f}^B$ flow. The stuck packets (those labeled 1) are forwarded via the f' functions. Overall, under strategy σ'_B at each node u the list L_u^B is traversed and in each time step the $N - B(u)$ next entries in the list are selected. For each entry of the form (j, v, B) the oldest packet of commodity j that has mark B is forwarded to node v. If no such packet exits, no packet is forwarded. For each entry $(j, v, 1)$ the oldest packet of commodity j with marking 1 is forwarded to v. If no such packet exists, no packet is forwarded. For *null*-entries, no packets are sent. After K^B time steps, the end of the list is reached and it is again traversed from the beginning. Before starting a new round on the list at node u, the oldest regular packet of each commodity at u is turned into a stuck packet (by changing its mark B' to 1). As in the argument for Lemma 1, at any point in time, there are at most $2K^B \cdot \bar{f}_j^B(u)$ packets of commodity j with marking B at u. During one traversal of the list, at most $1+\sum_{v:(v,u)\in E} f'_j(v,u)$ new stuck packets arrive at u (from incoming edges or by converting a regular packet). The number of packets that can be forwarded during one traversal of the list is $\sum_{v:(u,v)\in E} K^B f'_j(u,v) = K^B(\sum_{v:(v,u)\in E} f'_j(v,u) + \delta) \geq 1 + \sum_{v:(v,u)\in E} f'_j(v,u)$. Therefore, there are at most $2K^B \cdot \sum_{v:(v,u)\in E} f'_j(v,u)$ packets of commodity j with marking 1 at u.

Now consider arbitrarily varying blocking functions $B \in \mathcal{B}$. We define in analogy to the proof of Theorem 2 the strategy σ' as follows: σ' plays according to σ'_B whenever the blocking function $B \in \mathcal{B}$ is active. If demand agent switches to some other blocking function and later switches back to B, the list is traversed, continuing at the point where it was when the blocking function was switched. Overall, at any point in time there will never be more than

$$R_j(u) := \sum_{B\in\mathcal{B}} \left(2K^B \bar{f}_j^B(u) + 2K^B \sum_{v:(v,u)\in E} f'_j(v,u) \right)$$

packets of commodity j at u. Therefore, the total number of packets of commodity j in the network is at most $R_j := \sum_{u\in V} R_j(u)$

Next we show that σ' provides a fixed delay bound on all packets. Note that each packet can be forwarded at most $|V|-1$ times as regular packet before reaching its destination, and similarly at most $|V| - 1$ times as stuck packet. Now consider the oldest packet of commodity j in the network. Within the next $T := \sum_{B\in\mathcal{B}} K^B$ time steps this packet is either forwarded (as regular or as stuck packet), or it is turned into a stuck packet because then at least one blocking function was active long enough to complete a round of K^B time steps. Hence, the oldest packet of commodity j will reach its destination and is removed after at most $(2|V|-1)\cdot T$ time steps. Since at any point in time there are at most R_j many packets of commodity j in the network, after $R_j(2|V|-1)\cdot T$ rounds any packet of commodity j has reached its destination. ∎

Delay bounds in networks with adversarial packet injection have been considered by Aiello et al. [4]. In contrast to our result, their delay bound does not apply to dynamic network capacities as they utilize an additional static routing scheme that is executed periodically.

IV. Model-checking Tool

Theorem 2 gives a characterizing condition for which a network is stable under adversarial conditions. In this section, we demonstrate how to put this result into more practical use. We present an implemented algorithm that operates in the fashion of a model-checker. The algorithm determines whether in a given network the conditions for worst-case stability hold. In case no stability can be guaranteed, the algorithm provides an example blocking function which prevents stability. The algorithm can be easily extended to determine the existence of a delay bound as well (by taking the conditions of Theorem 4 into account). We evaluate the performance of our algorithm, and discuss resulting applications.

The biggest problem to overcome in order to check stability is the computational effort involved. On the one hand, checking the condition for worst-case stability involves the computation of a sufficient MCF for each possible blocking function $B \in \mathcal{B}$. Indeed, calculating a single MCF is less an obstacle, as this problem is algorithmically well understood. Even though the integer version of the MCF problem is NP-complete [14], polynomial time algorithms are known for calculating fractional valued MCFs [15]. However, checking the existence of MCFs for each blocking function separately is tedious because in the general case the set $\mathcal{B}$ can be large (exponential in the network size). To still allow for realistic network sizes, we argue that under certain (simple) restrictions on the structure of $\mathcal{B}$, one can skip the MCF computation for a large fraction of the occurring blocking functions. In particular we consider the following straightforward restriction: We specify a fixed bound $O_{\max} \in \mathbb{N}$ and assume that the sum of blocked channels never surpasses $O_{\max}$. We call a blocking function that fulfills this constraint $O_{\max}$-*compliant* and denote by

$$\mathcal{B}(O_{\max}) = \{B \mid B \text{ is } O_{\max}\text{-}\textit{compliant}\}$$

Algorithm 1 The backtracking algorithm determines for a given problem instance whether a blocking function without suitable MCF exists.

Require: Network G, demand $\mathcal{D}$ and $O_{\max}$-compliant $\mathcal{B}$
Ensure: Returns a blocking assignment B for that no MCF with throughput λ_j for each commodity j exists or `null` otherwise.

```
1: function BACKTRACK(O, B)
2:     F ← maximal throughput MCF on G with blocking function B
3:     T ← excess throughput of F
4:     if T < 0 then return B
5:     else if O ≤ T then return null
6:     else
7:         F ← normalize(F)
8:         for all u ∈ V do
9:             r ← ⌊N − B(u) − F(u) + 1⌋
10:            if r ≤ O ∧ N − B(u) − r ≥ c_min then
11:                B' ← backtrack(O − r, newB(B, u, r))
12:                if B' ≠ null then return B'
13:                end if
14:            end if
15:        end for
16:    end if
17:    return null
18: end function
```

the set of all $O_{\max}$-*compliant* blocking functions. In general, the set $\mathcal{B}(O_{\max})$ with $O_{\max} \geq N$ contains blocking functions that will not support stability: A source node can be blocked out completely if the adversarial occupies all N channels at that node. Hence, checking these sets is trivial. Nevertheless, we can add further constraints to $\mathcal{B}$ that allow the considered traffic to have a certain minimum capacity as considered in Theorem 4: For all $u \in V$ it holds that $N - B(u) \geq c_{\min}$.

In the following, we assume the blocking functions contained in $\mathcal{B}$ to be both $O_{\max}$-*compliant* and respecting a given minimal channel capacity $c_{\min}$. Note that these restrictions for the blocking functions have some quite practical implications: Assume a wireless network provider to own spectral resources which are chopped up into N channels per base station. Given a certain load (equaling the arrivals in our model), we may ask how many of the N channels can be rented out to secondary users without further implications on the exact set of channels such that the stability of the providers network is not harmed. Assuming that a minimum capacity per node is held back for the providers traffic, this problem matches exactly the conditions specified previously with respect to the restrictions on the set of blocking functions B. In an equal manner, questions of survivability of fixed networks can be mapped into the same framework.

A. Algorithm Implementation

Our algorithm works in a backtracking fashion to find a blocking function for which no MCF with sufficient throughput exists (see also the pseudo-code in Algorithm 1): `backtrack`(O, B) takes two parameters. B is the blocking function that is currently considered and O is the number of channels that can be blocked additionally. Let B_0 be the blocking function that blocks no channels, i.e., $B_0(u) = 0$ for all $u \in V$. We have that B_0 is $O_{\max}$-*compliant*. Initially, the algorithm is called by `backtrack`$(O_{\max}, B_0)$.

At first the algorithm computes an MCF for the current blocking function B. This computation is explained in detail further below. The MCF is computed such that it provides a maximal additional throughput (we call it *excess throughput*) for each commodity j: Its throughput is $(\lambda_1 + T, \dots, \lambda_m + T)$ for T being maximal, instead of $(\lambda_1, \dots, \lambda_m)$. There are two breaking conditions, depending on T: a) If the excess throughput T is negative, no feasible MCF exists and we can return B as a counterexample[4]. b) Up to O channels may be blocked by the adversarial in addition to the current blocking assignment B. If $O \leq T$ then clearly, the MCF cannot be made unfeasible by O additional blockings and hence `null` is returned (i.e., we ascend in the backtracking tree). In case that neither condition is fulfilled, we have to descend in the backtracking tree. For this, the current blocking function has to be modified (by occupying more channels), so that the current MCF becomes infeasible. Therefore, the algorithm normalizes the MCF to throughput $(\lambda_1, \dots, \lambda_m)$, i.e., it sets $F_j(u,v) \leftarrow \frac{\lambda_j}{\lambda_j + T} F_j(u,v)$ for all $1 \leq j \leq m, (u,v) \in E$. In order to make the MCF unfeasible, the adversarial has to occupy enough channels so that for some u there is not enough capacity left to support $\sum_{j=1}^{m} \sum_{v:(u,v)\in E} F_j(u,v)$. In case this can be achieved by blocking at most O additional channels while respecting $c_{\min}$, the backtracking algorithm is called recursively. Here, `newB`(B, u, r) is the blocking function that blocks $B(u)+r$ channels on u and $B(v)$ channels for $v \in V, v \neq u$.

Overall, the runtime of this algorithm is still exponential in the size of the network. However, as we demonstrate in the evaluation, the algorithm typically considers only a small fraction of the possible blocking functions so that only relatively few MCFs have to be computed.

In line 2 of Algorithm 1, a feasible MCF for the current blocking function is computed. For simplicity and more flexibility during the development, we use a formulation as linear program (LP) in our implementation to compute the MCFs. The LP instances are processed by the well-known LP-solver CPLEX.

The input data is the connectivity graph $G = (V, E)$, the demand $\mathcal{D}$, a blocking function B and the number of channels N. The required variables are $T \in \mathbb{Q}$, which represents the excess throughput for all commodities, and $f_{u,v,j} \in \mathbb{Q}^+$ for $(u,v) \in E, 1 \leq j \leq m$ representing the fraction of the flow of commodity j on edge (u,v). We formulate the LP as follows:

[4]Note that in order to check for the existence of a delay bound according to Theorem 4, the algorithm must be modified to check whether $T \leq 0$. For stability according to Theorem 2, $T < 0$ is enough.

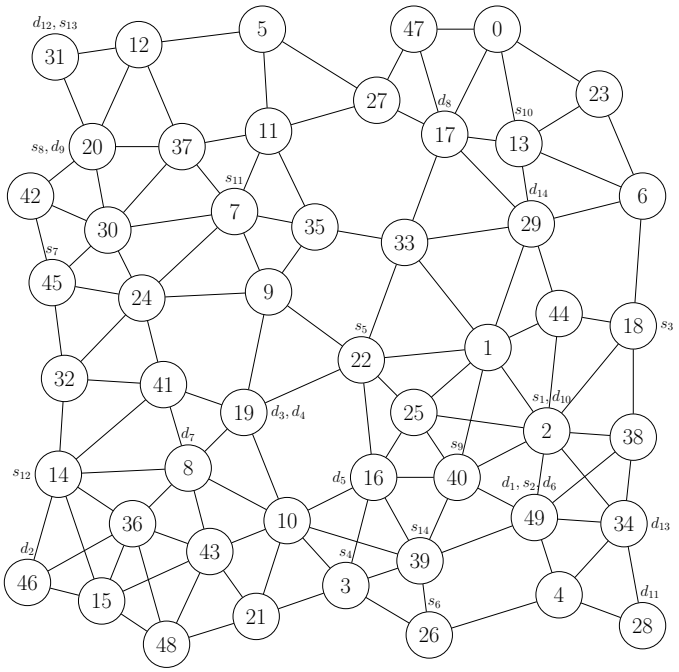

Fig. 1. The evaluation scenario consists of 50 nodes. We assume a maximal capacity of $N = 3$ and $c_{\min} = 1$. The source and destination nodes of demand $\mathcal{D}_4$ are annotated.

$O_{\max}$	$\lvert\mathcal{B}\rvert$
1	51
2	1.326
3	23.376
4	313.701
5	3.412.461
6	31.298.361
7	248.635.761
8	1.744.483.611
9	10.970.926.711
10	62.561.143.641

TABLE I
CARDINALITY OF $\mathcal{B}$ IN A NETWORK OF 50 NODES WITH $N = 3$ AND MINIMAL CAPACITY OF $c_{\min} = 1$ FOR DIFFERENT VALUES OF $O_{\max}$

	$O_{\max}$									
	1	2	3	4	5	6	7	8	9	10
$\mathcal{D}_1$	•	•	•	•	•	•	•	•	•	•
$\mathcal{D}_2$	•	•	•	•	•	•	•	•	•	○
$\mathcal{D}_3$	•	•	•	•	•	•	•	○	○	○
$\mathcal{D}_4$	•	•	•	○	○	○	○	○	○	○

TABLE II
THE DEMANDS LABELED WITH • ARE SUPPORTED IN THE NETWORK WITH BLOCKING FUNCTIONS INDUCED BY $O_{\max}$. FOR THE ○ ENTRIES, AT LEAST ONE BLOCKING FUNCTION WITHOUT SUITABLE MCF EXISTS.

Maximize T, subject to

$$\sum_{v:(v,u)\in E} f_{u,v,j} = \sum_{w:(u,w)\in E} f_{u,w,j} \quad \forall j\ \forall u \neq s_j, d_j \quad (6)$$

$$\sum_{v\in V} f_{v,d_j,j} \geq \lambda_j + T \quad \forall j \quad (7)$$

$$\sum_{v\in V} f_{d_j,v,j} = 0 \quad \forall j \quad (8)$$

$$\sum_{v\in V}\sum_{j=1}^{m} f_{u,v,j} \leq N - B(u) \quad \forall u \in V \quad (9)$$

Constraints 6 to 9 together with the variable definition correspond to the constraints 1 to 5 in Definition 1. Hence, they ensure that a feasible MCF is computed. The variable T is maximized, so that each commodity has T excess throughput in addition to the required throughput λ_j. Because $T \in \mathbb{Q}$, a feasible solution always exists, however if $T < 0$ then no feasible MCF exists.

B. Numerical Evaluation

1) Evaluation Scenario: We measure the performance of our algorithm on a randomly generated network depicted in Figure 1. There are in total 50 nodes, the maximal number of channels is set to $N = 3$ and in each blocking function, there is a minimal capacity of $c_{\min} = 1$. We consider the 10 $O_{\max}$-*compliant* sets that are induced by the choice of $O_{\max} \in \{1, \dots, 10\}$. For our network, this results in cardinalities of $\mathcal{B}$ as shown in Table I.

We consider four different traffic scenarios. In the first scenario there are $m = 8$ random source-destination pairs namely $\mathcal{D}_1 = \{(2, 49, 1), (49, 46, 1), (18, 19, 1), (3, 19, 1), (22, 16, 1), (26, 49, 1), (45, 8, 1), (20, 17, 1)\}$. In the second scenario we add two further pairs, so that we have $\mathcal{D}_2 = \mathcal{D}_1 \cup \{(40, 20, 1), (13, 2, 1)\}$. Subsequently, we add further commodities in the third and fourth scenario, resulting in $\mathcal{D}_3 = \mathcal{D}_2 \cup \{(7, 28, 1), (14, 31, 1)\}$ and $\mathcal{D}_4 = \mathcal{D}_3 \cup \{(31, 34, 1), (39, 29, 1)\}$.

2) Stability Results: We have assessed our algorithm for 40 different combinations of demands and sets of blocking functions in the above scenario. Table II shows the outcome of the computation. The demands $\mathcal{D}_1$ are supported for all tested choices of $O_{\max}$. For the other demands, it depends on the choice of $O_{\max}$, whether worst-case stability is given. Demand $\mathcal{D}_2$ with 10 commodities is guaranteed to work when at most 9 channels are blocked. For 10 or more blocked channels, there is a blocking function that will not support all demands. When even more commodities are added, the flexibility of the network in terms of blocked channels is diminished further. For $\mathcal{D}_3$ no more than 7 blocked channels and for $\mathcal{D}_4$ no more than 3 concurrently blocked channels are supported.

3) Runtime Behavior: In Figure 2 the total number of MCFs computed by our algorithm is shown for each setting of $O_{\max}$. Comparing this graph with the amount of potential computations, as shown by Table I, it is clear that only a tiny fraction of MCFs actually needs to be considered. Still, it is also important to consider the run time of the computation of a single MCF. The used LP formulation to compute the MCF grows for each additional commodity by $|E|$ many new variables and $|V|$ additional constraints. Still, when considering the average runtime for the computation of a single MCF as shown in Table III, the individual computation is reasonably low. Overall, this results in run-times which are below one hour in the worst case. Note that a more efficient

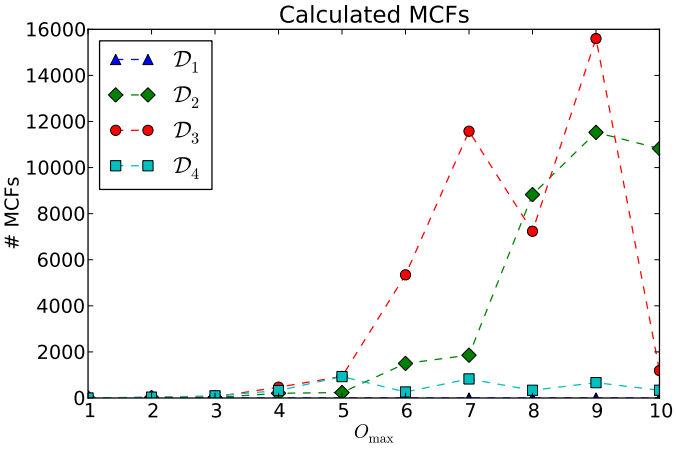

Fig. 2. The total number of calculated MCFs in each run. This number corresponds to the total number of checked blocking assignments.

$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$
-	47 ms	81 ms	120 ms

TABLE III
AVERAGE RUNTIME PER MCF ON AN *AMD Phenom II* X4 945 QUAD CORE MACHINE WITH 3.0GHZ AND 8GB OF RAM; 64BIT *Ubuntu* 10.04 AND *CPLEX* 12.4. FOR $\mathcal{D}_1$ NO MCF WAS COMPUTED BY THE ALGORITHM, AS THIS DEMAND IS TRIVIALLY SUPPORTED.

implementation of the MCF calculation could further reduce the runtime.

4) Remarks: Our backtracking algorithm enables an effective checking of stability conditions in reasonably sized networks. It accomplishes this by cutting large parts of the backtracking tree, enabling to skip most blocking assignments. The algorithm provides an example blocking function, whenever the demand is not supported due to bottlenecks in capacity. For example, for $\mathcal{D}_4$ and $O_{\max} = 4$, the algorithm identifies a bottleneck around node 31. If in total more than 3 channels are blocked out on the nodes 20 and 12, demand $\mathcal{D}_4$ cannot be supported anymore. This information can be used iteratively to improve the stability: Either certain blocking functions are excluded or more capacity. A further solution would reduce the traffic or could introduce priorities among the traffic flows. When the bottleneck has been taken care of, the algorithm is run again to either identify further problematic areas or verify stability.

Furthermore, we note that the packet and delay bounds obtained in our main results can be significantly improved by our model-checking algorithm. The bounds are obtained by summation over all possible blocking functions. However, with our model-checking algorithm we calculate a set of MCFs that work for all possible blocking functions. Typically, this set of MCFs is significantly smaller than $\mathcal{B}$. For the bounds, it is enough to summarize over this much smaller set.

V. CONCLUSION

In this work, we present a novel approach for network stability analysis under adversarial worst-case conditions. We first introduce a network model that considers three key network parameters: topology, traffic and capacities. Next, we formulate an infinite game upon our network model with which we model adversarial capacity changes in the network. This allows us first to derive the stability region of a dynamic network under adversarial conditions without specifying the routing algorithm. Furthermore, we can provide conditions under which a delay bound exists for these networks. A further benefit of our infinite game formulation, not present in related work, is that it enables us to construct a model-checking algorithm which determines for a practical instance whether the network stays stable. If network stability can not be provided, the tool identifies the traffic pairs that cause the network to become unstable as well as the nodes for which this happens.

Finally, our routing game framework allows for future research. For example, augmenting the model with a notion of maximum *burstiness* similar to Andrews et al. [7] or assuming *channel sensitive transmission* where each packet transmission blocks a dedicated channel on source and destination, yields the question how to characterize the games winnable by routing agent. How does the adversarial traffic relate to MCFs?

REFERENCES

[1] L. Tassiulas and A. Ephremides, "Stability Properties of Constrained Queueing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, 1992.

[2] M. Neely, E. Modiano, and C. Rohrs, "Dynamic Power Allocation and Routing for Time Varying Wireless Networks," in *Proc. of IEEE INFOCOM*, vol. 1, 2003, pp. 745–755.

[3] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson, "Adversarial Queueing Theory," in *Proc. of ACM STOC*, 1996, pp. 376–385.

[4] W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosén, "Adaptive Packet Routing for Bursty Adversarial Traffic," in *Proc. of ACM STOC*, 1998, pp. 359–368.

[5] M. Andrews, B. Awerbuch, A. Fernández, T. Leighton, Z. Liu, and J. Kleinberg, "Universal-Stability Results and Performance Bounds for Greedy Contention-Resolution Protocols," *Journal of the ACM*, vol. 48, no. 1, pp. 39–69, 2001.

[6] E. Anshelevich, D. Kempe, and J. Kleinberg, "Stability of Load Balancing Algorithms in Dynamic Adversarial Systems," in *Proc. of ACM STOC*, 2002, pp. 399–406.

[7] M. Andrews, K. Jung, and A. Stolyar, "Stability of the Max-Weight Routing and Scheduling Protocol in Dynamic Networks and at Critical Loads," in *Proc. of ACM STOC*, 2007, pp. 145–154.

[8] S. Lim, K. Jung, and M. Andrews, "Stability of the Max-Weight Protocol in Adversarial Wireless Networks," in *Proc. of IEEE INFOCOM*, 2012, pp. 1251–1259.

[9] D. Gale and F. M. Stewart, "Infinite Games with Perfect Information," in *Contributions to the Theory of Games 2*, ser. Annals of Mathematics Studies. Princeton University Press, 1953, no. 28, pp. 245–266.

[10] D. A. Martin, "Borel Determinacy," *Annals of Mathematics*, vol. 102, no. 2, pp. 363–371, 1975.

[11] A. Church, "Applications of Recursive Arithmetic to the Problem of Circuit Synthesis," in *Summaries of the Summer Institute of Symbolic Logic*, 1957, pp. 3–50.

[12] E. Grädel, W. Thomas, and T. Wilke, *Automata, Logics, and Infinite Games: A Guide to Current Research*. Springer, 2002, vol. 2500.

[13] W. Thomas, "Solution of Church's problem: A Tutorial," in *New Perspectives on Games and Interaction*, ser. Texts in Logic and Games. Amsterdam University Press, 2008, vol. 4, pp. 211–236.

[14] S. Even, A. Itai, and A. Shamir, "On the Complexity of Time Table and Multi-Commodity Flow Problems," in *Proc. of IEEE FOCS*, 1975, pp. 184–193.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.