Theoretical Computer Science 410 (2009) 670–687

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Adversarial queuing theory with setups

M. Kiwi [a,b,*], M. Soto [a], C. Thraves [a]

[a] Depto. Ing. Matemática, U. Chile, Correo 3, Santiago 170–3, Chile
[b] Ctr. Modelamiento Matemático (UMI 2807, CNRS), U. Chile, Correo 3, Santiago 170–3, Chile

ARTICLE INFO

Article history:
Received 17 August 2006
Received in revised form 22 September 2008
Accepted 23 September 2008
Communicated by J. Diaz

Keywords:
Adversarial queueing theory
Routing protocols
Scheduling protocols
Setups

ABSTRACT

We look at routing and scheduling problems on Kelly type networks where the injection process is under the control of an adversary. The novelty of the model we consider is that the adversary injects requests of distinct types. Resources are subject to switch-over delays or setups when they begin servicing a new request class. In this new setting, we study the behavior of sensible policies as introduced by Dai and Jennings [J. Dai, O. Jennings, Stabilizing queueing networks with setups, Math. Oper. Res. (2004) 891–922].

We first show that the model is robust in the sense that under some mild conditions universal stability of work conserving packet routing protocols is preserved for natural variants of the underlying model. Also, the model's equivalence to so called token networks is established.

We adapt to the multi-type request and setup setting, standard arguments for proving stability. Nevertheless, we provide counterexamples that show that for several reasonable adaptations of contention resolution protocols to the multi-type case, stability results do not carry over from the single-type scenario. This motivates us to explore fluid model based arguments that could be used for proving stability for a given network. Specifically we show analogues of results obtained by Gamarnik [D. Gamarnik, Stability of adversarial queues via fluid model, in: Proc. of the 39th Annual Symposium on Foundations of Computer Science, 1998, pp. 60–70] but in the multi-type request with setups scenario.

1. Introduction

Many processes in industry and elsewhere require completing a collection of tasks while satisfying temporal and resource constraints. Temporal constraints say that some tasks have to be finished before others can be started; resource constraints say that two tasks requiring the same resource cannot be done simultaneously. Typically, resources are either machines, shops, staff, etc. In many cases, it is natural to consider dynamic situations where jobs arise over time. This gives rise to an on-line version of the job-shop scheduling problem. In such scenarios, setup costs are typically associated with resources (e.g. a product might need painting — to change colors might require stopping, washing, loading new paint, and re-starting the machine). A resource is not available to process requests during setup. When many requests vie for the same resource, some type of contention resolution protocol is needed to choose which request to process while all others pile up in a queue associated with the resource. In the aforementioned context the central issue worth analyzing concerns *stability*, i.e. whether the number of unserviced requests remains bounded as the system runs for an arbitrarily long period of time.

Queuing theory studies the waiting times, lengths, and other properties associated with queues. In the context of network traffic and on-line job-shop scheduling (the areas of main concern to the ensuing discussion) some stochastic assumption is often made about the request generation process. However, typical premises as say exponentially and independently

* Corresponding author. Fax: +56 2 688 3821.
E-mail addresses: mkiwi@dim.uchile.cl (M. Kiwi), msoto@dim.uchile.cl (M. Soto), thraves@dim.uchile.cl (C. Thraves).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.09.064

distributed inter-arrival times are unrealistic in complex networks such as the Internet. In order to lift these hypotheses an alternative elegant framework was proposed by Borodin et al. [9] and is referred to as Adversarial Queuing Theory (AQT). In it, requests are generated by an adversary rather than by a stochastic process. Specifically, there is an underlying network graph where links are assumed to be of fixed capacity. Events happen at discrete unit time intervals. At each time step the adversary may inject requests, i.e. specify a route in the network that the request must traverse in order to be absorbed. Once absorbed, the request is considered to have been completed and thus "disappears" from the network. Per time step, each network link can process as many requests as its capacity. Packets waiting to cross an edge are said to be queued at the edge. If an edge's queue size is larger than the edge's service capacity, then a scheduling protocol chooses which packet to forward across the link.

In [9], a restriction on the adversary is proposed. Informally, the adversary is not allowed to inject (on average) more requests than $r \leq 1$ times the capacity of any edge (the parameter r is referred to as the adversaries rate). Thus, there are no trivially identifiable "hot-spots" in the network. The model's appeal and the great interest it has attracted is partly explained by the fine balance between the predictability of traditional injection processes (as in classical queuing theory) and on the other hand, unconstrained adversaries which can easily overload the system (as in competitive analysis).

The job-shop scenario mentioned at the start of this section encompasses the packet routing setting considered in [9]. Indeed, given a routing network one may view each network link as a distinct shop and each packet as a job. Link queues will thus correspond to shop queues. The converse holds, i.e. packet routing encompasses the job-shop scenario, but now jobs give rise to routes which might visit an edge multiple times. Say that a path is *simple* if it visits each network edge at most once. Traditionally, and quite naturally for packet routing networks, the adversarial queuing literature has presupposed packets visit each node at most once — hence, packets also follow simple paths. Nevertheless, for a given network G, any initial configuration, and a rate r adversary A which may inject packets along arbitrary paths, there is a network G', a starting configuration, and a rate r adversary A' whose injection paths are simple, and such that the total number of packets in G remains bounded independent of time if and only the total number of packets in G' can be bounded independent of time. (The crux of the proof argument required to establish the aforementioned folklore result is implicit in the proof of [3, Theorem 5.1].) Thus, in an adversarial setting, stability issues in online job-shop problems are equivalent to stability of packet routing networks where injection paths are simple. However, in the former class of problems, requirements typically belong to distinct classes and setup costs are usually involved. In the language of packet routing this translates to multiple packet types and latency penalties due to switching among servicing distinct packet types. We investigate stability issues in this scenario. But in fact, we do more. To explain this, we first precisely describe the model we consider.

1.1. Model specification

A *routing network* is a directed graph of Kelly type, i.e. with unit link capacities. Time proceeds in discrete *steps*. A *packet* is an atomic entity that resides at the queue of an edge at the end of any step. A packet travels along a path in the network from its injection node (*source*) to its absorption node (*destination*). Formally, a packet is a triple (ID, P, t) where ID is some unique identifier, P is the packet's traversal path, and t is the time that the packet was injected into the network. An *adversary* is an entity that at each time step generates a set of *packet injections*, i.e. source to destination paths traversed by each packet.

Our first departure from extant adversarial models is that the adversary will also specify, for each edge along a packet's injection path, a *type* (or *class*) chosen from a fix set of classes $\mathcal{I}$. Formally, when a packet p is injected the adversary specifies a traversal path $e_1 \dots e_\ell$ and a *traversal type function* $f_p : \{1, \dots, \ell\} \to \mathcal{I}$ (throughout, we abuse notation and view a path both as a multi-set and a sequence of edges). The traversal function should be interpreted as capturing the fact that as a packet progresses through the network it might require being serviced by different resources, say resource $f_p(i)$ for edge e_i. This formalization captures the case where a fixed type is associated to a packet at injection, and also encompasses situations where types might change as the packet moves through the network.

Remark 1. It is also natural to let f_p take values not in $\mathcal{I}$ but in its power set (note that this is drastically different from identifying a distinct type with each subset of $\mathcal{I}$). An underlying motivation for such a general framework is one where jobs (e.g. cars in an assembly line) move between shops some of which might handle indifferently certain collections of job classes (e.g. distinct color cars in the assembly line example). Throughout this work we shall be concerned solely with the case where traversal type functions take values in $\mathcal{I}$.

The cases where $\mathcal{I}$ contains one and more than one element will be referred to as the *single-class* (or *single-type*) and *multi-class* (or *multi-type*) case respectively. In the multi-type scenario, we also distinguish two situations depending on whether or not the traversal functions associated with packets are constant functions. The non-constant case will be called the *dynamic multi-type* case and the underlying adversary will be called a *dynamic multi-type adversary*. The non-dynamic case will be called *static*. The static multi-type setting captures situations where the type of a packet is fixed once and for all at its injection step. Even such a restricted scenario is of major practical relevance, e.g. in TCP/IP networks such as the Internet packets may be classified according to their data load (e.g. audio, video, etc.) or destination port — this classification remains fixed throughout the packet's lifespan.

In the AQT model, the adversary is constrained by a pair (r, b) where $b \geq 1$ is a natural number and $0 < r \leq 1$. The restriction on the adversary is [9,5] such that for every edge e and all $t_1 < t_2$,

$$A_e(t_1, t_2) \leq r(t_2 - t_1) + b, \tag{1}$$

where $A_e(t_1, t_2)$ denotes the number of packets injected by the adversary during time interval $[t_1, t_2)$ that need to traverse edge e in their source to destination routes (for an injection path crossing e several times each traversal is counted separately by $A_e(\cdot)$). The restriction specified in (1) will be referred to as *load constraint* — it is motivated by the fact that it inhibits the adversary from trivially congesting the network. Roughly, this constraint asserts that for any time period of length $t_2 - t_1$ the number of packets injected that require crossing a given edge is bounded by $r(t_2 - t_1) + b$. Specifically, except for at most b sporadic packets, no more than r such packets are injected on average during each time step. An adversary satisfying the load constraint will be referred to as an (r, b) adversary. Note that this adversarial model allows for injection patterns that are "bursty" — the burst parameter being represented by b. The class of *bounded rate* adversaries is the collection of (r, b) adversaries such that $r < 1$.

In a Kelly type routing network, if several packets try to cross an edge during the same step, then a *packet scheduling policy* decides which one to send across the link. The other packets wait in the edge's queue. Our second departure from the standard adversarial model is that no queued packet that belongs to class $i \in \mathcal{I}$ may be chosen by the queuing policy if a packet belonging to class $j \in \mathcal{I}, j \neq i$, was sent across the link during the previous $\Delta_{j,i}$ time steps — where $\Delta_{i,j}$'s are network parameters henceforth called *setup costs* or *latencies*. We henceforth let Δ denote the maximum of the $\Delta_{i,j}$'s taken over all $i, j \in \mathcal{I}$ (we always assume $\Delta_{k,k} = 0$). Formally, denote by $D_{e,i}(t_1, t_2)$ the number of type i packets crossing e during the interval $[t_1, t_2)$. The existence of setup costs is captured by the following *setup constraint*: for every edge e,

$$D_{e,i}(t, t+1) > 0 \Longrightarrow \forall j \neq i, D_{e,j}(t+1, t+1+\Delta_{i,j}) = 0. \tag{2}$$

We henceforth focus only in the case where $\Delta_{i,j} = \Delta$ for every $i \neq j$. We refer to Δ as the *network setup cost* or simply *setup cost*.

A *packet scheduling policy* specifies, for each network edge and each time step, which packet waiting at the edge's queue is to be moved (if possible). Examples of natural policies are considered in [9], among them:

- First-In-First-Out — packets are served in the same order they arrive.
- Nearest-To-Go — each packet has a prescribed path and priority is given to the packet which has the smallest number of edges still to be traversed.
- Farthest-To-Go — each packet has a prescribed path and priority is given to the packet which must still traverse the largest number of edges until absorption.
- Shortest-In-System — a packet originates at a specified time and priority is given to the packet that has spent the least amount of time in the network.
- Longest-In-System — a packet originates at a specified time and priority is given to the packet that has spent the largest amount of time in the network.

A *switching policy* specifies, for each network edge and each time step, whether to keep servicing packets of the type serviced during the preceding time step, or to start servicing (if possible) a new packet type. In other words, a packet scheduling policy specifies how priorities are assigned amongst packets of the same type while a switching policy specifies priorities among distinct packet types. Examples of switching policies are:

- Largest-Queue — switch to serving the type that has the largest number of packets waiting at the queue (ties are broken arbitrarily).
- Round-Robin — types are numbered $0, \ldots, |\mathcal{I}|-1$, if type i was last served, then switch to sending type $(i+1) \bmod |\mathcal{I}|$.
- First-In-First-Out — switch to sending the type for which a packet first arrived at the queue (ties are broken arbitrarily).
- Priority — types are numbered $0, \ldots, |\mathcal{I}|-1$, switch to serving the type whose value is smallest among all the packet types awaiting at the queue.

For the sake of preciseness, we point out that once the switching policy determines to start servicing a different packet type it cannot change its decision before Δ steps have elapsed. So for example, an edge that has been idle for less than Δ units of time will not be readily available for servicing an arbitrary request type.

We say that a switching policy is *exhaustive* if once it starts servicing packets of type i it must continue serving packets of type i until none remain in the queue. In this work we only consider exhaustive switching policies. (For results concerning exhaustive policies in the standard stochastic queueing theory setting the reader is referred to [28,21,23].)

Consider the permutation of $\mathcal{I}$ defined by $\pi(i) = (i+1) \bmod |\mathcal{I}|$. We say that a switching policy is *shift-oblivious* if on a given queue state it chooses to serve next type j packets, then when every packet of type i is replaced by one of type $\pi(i)$, the switching policy will choose to serve next type $\pi(j)$ packets. An example of a shift-oblivious switching policy is Round-Robin. Switching policy Priority is an example of a policy which is not shift-oblivious. We say that a switching policy is *type-oblivous* if the condition in the definition of shift-oblivious policy is satisfied for every permutation π of $\mathcal{I}$. Note that any type-oblivious switching policy is also shift-oblivious. Examples of type-oblivious switching policies are Largest-Queue and First-In-First-Out. Moreover, Round-Robin is an example of a switching policy that is shift-oblivious but not type-oblivious.

A *scheduling protocol* is one that specifies both a packet scheduling and a switching policy. A *greedy* or *work conserving* scheduling protocol is one that advances a packet through an edge provided the edge's queue is not-empty and the edge is not in a setup period. This notion coincides with that of *work conserving packet scheduling protocols* in the special case

where all packets are of the same type. It is also worth to stop and discuss the case where the setup cost Δ is 0. In particular, observe that even in this case the packet type is not irrelevant. Indeed, the exhaustiveness of the switching policy we focus on requires that all packets of a given type waiting to cross an edge be served before packets of a distinct type can start being serviced.

Our main goal is to study *stability* of arbitrary and particular greedy scheduling protocols on various networks and against restricted adversaries in the AQT with setup scenario described so far.

We define a *network system* as the tuple $(G, \Delta, (P, S); A)$ where G is a routing network, $\Delta \geq 0$ is a setup cost, P is a packet scheduling policy, S is a switching policy, and A is the adversary that specifies the packet injection pattern.

The *state of packet* p at time t, denoted $state_t(p)$, is the vector $(p, t_1, \ldots, t_s)$ where s is the number of time steps up to t in which P has traversed an edge in its path and t_j is the time of the jth such traversal. We define the *packet configuration* $PckCnf_t(\mathcal{N})$ of the network system $\mathcal{N} = (G, \Delta, (P, S); A)$ at time t as the collection of $state_t(p)$ for which p is a packet that is present in the network at time t. Note that the packet configuration implicitly specifies the location of packets in each (edge) queue of the network. Analogously, we say that the *state of an edge* e at time t, denoted also $state_t(e)$, is the vector $(e, i_0, \ldots, i_s, t_0, \ldots, t_s)$ where t_j's are the time steps up to t in which the switching policy specifies that edge e should start servicing a packet type i_j distinct than the one serviced during step $t_j - 1$ (we always assume $t_0 = 0$ and hence i_0 denotes the packet type serviced at $t = 0$). We define the *switching configuration* $SwtCnf_t(\mathcal{N})$ of the network system $\mathcal{N} = (G, \Delta, (P, S); A)$ at time t as the collection of $state_t(e)$ for all edges e of G. A packet and switching configuration pair at time step t for a network $\mathcal{N}$, say $(PckCnf_t(\mathcal{N}), SwtCnf_t(\mathcal{N}))$, will be called a *network configuration* and denoted by $Cnf_t(\mathcal{N})$ or simply Cnf_t when $\mathcal{N}$ is clear from context.

In the standard AQT model it was argued early on [5, Lemma 2.9] that systems with empty initial configurations (systems for which at time zero there are no packets in the system) and systems with nonempty initial configurations are equivalent, since any adversary in the latter can be transformed into an adversary in the former that behaves similarly. However, it is important to remember that the construction used to establish such a result changes the network topology and creates a set of packets with a specific age; therefore, as noted in [13], if the scheduling policy bases its queuing decision on its history (e.g. in LONGEST-IN-SYSTEM) it is not clear if the above mentioned result remains valid. In the multi-type setting which is the focus of this work, we will not dwell further on the empty versus non-empty initial configuration issue and simply work on the more general non-empty initial configuration setting.

Definition 1. We say that the network system $(G, \Delta, (P, S); A)$ is stable if for every initial network configuration Cnf_0 there is a constant M (which may depend on the setup cost, the size of the network, the size of the initial configuration and parameters of P, S and A) such that when the network system is executed with initial configuration Cnf_0, packets are injected according to A's strategy and routed according to the scheduling protocol (P, S), the number of packets in any queue remains bounded above by M.

If the network system $(G, \Delta, (P, S); A)$ is stable, we say that network G with setup cost Δ and scheduling protocol (P, S) is stable against adversary A. So for example, a network system is stable against an (r, b) adversary if the long-run input rate r of the system is matched by the long-run output rate. If the network system $(G, \Delta, (P, S); A)$ is stable for a class of networks $G \in \mathcal{G}$ and every Δ, then we say that for the class $\mathcal{G}$ the scheduling protocol (P, S) is stable against adversary A.

We are particularly interested in determining whether a greedy protocol (P, S) is stable for every network G and setup cost Δ against any bounded rate adversary. Also, given a family of scheduling protocols, we would like to characterize the networks that are stable against bounded rate adversaries. However, it is intuitively clear that for stability to hold, the switching policies involved should have some mechanism to avoid frequently changing between which class of packets to serve. In order not to unnaturally construe the collection of switching policies we consider, we study the aforementioned properties under the *sensible policy* notion introduced by Dai and Jennings [17] — given a switching policy S and a parameter θ let S_θ denote the policy that picks according to S which class to service next among all the ones that have at least θ packets in the queue. If no class has at least θ packets waiting in the queue, then the policy may choose whatever class it prefers. Switching policies of the form S_θ are called sensible policies.

Definition 2. We say that the network G is stable for $\mathcal{S}$ with respect to the adversary class $\mathcal{A}$, if for every $\Delta \geq 0$, all $(P, S) \in \mathcal{S}$ and any $A \in \mathcal{A}$, there exists some $\theta \geq 0$ (which may depend on the setup cost, the size of the network, the size of the initial configuration and parameters of P, S and A) for which the network system $(G, \Delta, (P, S_\theta); A)$ is stable.

Definition 3. We say that the network G is universally stable with respect to the adversary class $\mathcal{A}$, if G is stable for $\mathcal{S}$ with respect to $\mathcal{A}$ where $\mathcal{S}$ denotes the class of greedy scheduling protocols.

Definition 4. We say that a scheduling protocol (P, S) is universally stable for $\mathcal{G}$ with respect to the adversary class $\mathcal{A}$, if for every $G \in \mathcal{G}$, the network G is stable for (P, S) with respect to the adversary class $\mathcal{A}$.

When the adversary class $\mathcal{A}$ is not mentioned, it is to be understood that it corresponds to the collection of bounded rate adversaries.

As in [9] we view the packet routing problem as a game between an adversary and a scheduling protocol.

1.2. Main contributions

In this work, we introduce an adversarial model that captures the natural scenarios where distinct types of network resource requests are generated in a non-predictable (adversarial) way but without trivially overloading the network. Specifically, we consider the case where an adversary injects packets into a network and associates classes (types) to each packet. Two situations are considered depending on the size of the set of possible request types $\mathcal{I}$: the single-class (or single-type) and the multi-class (or multi-type) case. In the multi-type scenario, two situations are distinguished depending on whether or not traversal functions are restricted to constant functions: the dynamic multi-type and the static case.

Our first main result shows that although constrained, the static multi-type scenario is of key importance. Indeed, stability issues for the dynamic multi-type case may be addressed by focussing on the static model. Specifically, in Section 2 we show, under some general conditions, that dynamic multi-type network systems can be simulated by static ones in the adversarial with setups model.

It is interesting to point out that dynamic multi-type routing networks with setups are "equivalent" to the so called token routing networks introduced by Kiwi and Russell [25]. In Section 2 we also precisely state and prove this assertion.

The previous discussion justifies why we focus our study of AQT with setups to the static multi-type case. Indeed, from Section 3 onwards the exposition concerns solely the static multi-type model.

In Section 3, we adapt arguments of [9,5] and establish stability of networks whose scheduling protocol relies on sensible policies. First, we show an analogue in the multi-type setting of Borodin et al.'s [9, Theorem 1] result about universal stability of acyclic digraphs under work conserving policies. Then, we establish for the multi-type setting a result similar to that of Andrew et al.'s [5, Theorem 3.7] which says that the unidirectional ring is universally stable under work conserving policies. We conclude Section 3 with a characterization of universally stable networks in the multi-type adversarial setting.

One is tempted to conclude that if a network G under a packet scheduling policy P is stable against an adversary A, then for any switching policy S and any Δ there exists a θ for which $(G, \Delta, (P, S_\theta); A)$ is also stable (the converse is obviously true). In Section 4, we show that this is not the case for various natural switching policies. Among others, for sensible versions of Largest-Queue, Round-Robin, and First-In-First-Out.

The state of affairs described above motivates us to explore general techniques that could be useful for proving stability for given networks. Specifically, in Section 5 we show an analogue of Gamarnik's [20, Theorem 1] result for the multi-type with setups scenario. Through these results one can establish the stability of a given network by considering an associated fluid model.

1.3. Related work

Cruz [14,15] proposed the "leaky-bucket" model, a predecessor of the AQT model [9,5]. The motivation underlying all these proposals is to have good models of heterogeneous networks such as for example Asynchronous Transfer Mode (ATM) switching networks. The relevance of these networks partly explains the considerable amount of attention given to the AQT model. Just for the sake of illustration we name three active lines of research; (1) the problem of characterizing and efficiently deciding whether a given network is universally stable was considered in [5,2], (2) in [4,5,19,27,29,8] the instability of the FIFO protocol was analyzed, and (3) stability of networks whose topology is time dependent was studied in [1,6,7].

Stochastic queuing networks is another branch of research in queuing theory. Seminal works in this area were [31,30,28] which established that load conditions are not sufficient for stability. Bramson [10] showed that FIFO is unstable for Poisson arrivals and exponential service times. This result was complemented also by Bramson [11] who proved the instability of FIFO at an arbitrary small ratio of arrival to service rates. This background of negative results motivated the development of techniques through which stability results for specific networks could be ascertained. Specially noteworthy is the "fluid model abstraction" developed by Dai [16] and Dai and May [18]. This abstraction's analogue for adversarial networks was formulated by Gamarnik [20]. The stochastic queuing network problem in the multi-class setting was first addressed by Chen [12]. Dai and Jennings [17] generalized the results of [16] to the multi-class setting with setup costs. For classical texts in queuing theory we refer the reader to [24,26].

2. Equivalences between models

In this section we first show that in the adversarial setting dynamic multi-type routing networks can be simulated by static multi-type networks, and vice-versa. We also describe the notion of token networks introduced in [25] and show that under some mild hypothesis they can be simulated by dynamic multi-type routing networks, and viceversa.

Theorem 1. *Let $\mathcal{N} = (G, \Delta, (P, S); A)$ be an arbitrary network system where (P, S) is a greedy scheduling protocol such that S is shift-oblivious and A is an (r, b) dynamic multi-type adversary. Then, there is a $\mathcal{N}' = (G', \Delta, (P, S); A')$ multi-type routing network where A' is an (r, b) static adversary, and an injective mapping φ from configurations of $\mathcal{N}$ to configurations of $\mathcal{N}'$, such that for any time step t, the configuration of $\mathcal{N}$ is Cnf_t if and only if the configuration of $\mathcal{N}'$ is $\varphi(Cnf_t)$.*

Proof. Without loss of generality assume that the collection of types $\mathcal{I}$ is $\{0, \ldots, n-1\}$ where the type labels are chosen such that if on a given queue state and while serving type i packets S chooses to serve next type j packets, then if every

packet of type k is replaced by a type $k+l$ packet, we will have that when serving type $i+l$ packets S will choose to serve next type $j+l$ packets. The existence of such labels is guaranteed by the fact that S is shift-oblivious.

Let V and E denote G's node and edge set respectively. Network G' will have V as node set. The edge set of G' will consist of n copies, say $e^{(0)}, \ldots, e^{(n-1)}$ of each edge $e \in E$. The idea of the proof is to simulate the traversal in G of a packet p by n packets $p^{(0)}, \ldots, p^{(n-1)}$ going through G'. When p crosses edge e of G, each packet $p^{(i)}$ will traverse a distinct edge $e^{(i)}$ of G' (the exact edge will depend on the type the adversary A assigns to packet p when traversing e).

The packet scheduling policy associated to G' is the same as the one of G. The switching policy is also the same in G and G'. The initial configuration of G' specifies which type will be served by each edge at time step $t=0$. In our case, the packet type initially serviced by edge $e^{(j)}$ is $(l+j) \bmod n$ where l is the packet type initially serviced by edge e.

We now describe the adversary A'. To do so, consider a packet p that is injected in G at time t by adversary A along path $e_1 \ldots e_\ell$ with a traversal type function $f_p : \{1, \ldots, \ell\} \to \mathcal{I}$. For $k = 1, \ldots, \ell$, let i_k denote the type assigned to the packet when it wishes to traverse edge e_k, i.e. $i_k = f_p(k)$. In G' adversary A' will inject n packets at time t, say $p^{(0)}, \ldots, p^{(n-1)}$. The type of packet p_j is fixed at injection and equals j. Using $\bmod n$ arithmetic on superindices, the injection path of packet p_j is expressed as $e_1^{(j-i_1)} \ldots e_\ell^{(j-i_\ell)}$. Note that A' is an (r, b) static adversary in G'.

Let φ be the map that associates to configuration $(PckCnf_t(\mathcal{N}), SwtCnf_t(\mathcal{N}))$ of $\mathcal{N}$ at time step t the pair (C_t, S_t) such that:

- Let $\mathcal{P}_t$ be the collection of packets present in network $\mathcal{N}$ at time step t. Thus, $PckCnf_t(\mathcal{N}) = (state_t(p))_{p \in \mathcal{P}_t}$. Then, $C_t = (state_t(p^{(i)}))_{p \in \mathcal{P}_t, i \in \mathcal{I}}$, where $state_t(p^{(i)}) = (p^{(i)}, t_1, \ldots, t_s)$ provided $state_t(p) = (p, t_1, \ldots, t_s)$.
- If $SwtCnf_t(\mathcal{N}) = (state_t(e))_{e \in E}$, then $S_t = (state_t(e^{(i)}))_{e \in E, i \in \mathcal{I}}$, where

$$state_t(e^{(i)}) = (e^{(i)}, (j_0 + i) \bmod n, \ldots, (j_s + i) \bmod n, t_0, \ldots, t_s)$$

  provided $state_t(e) = (e, j_0, \ldots, j_s, t_0, \ldots, t_s)$.

Note that C_t as defined above corresponds to n "copies" of $PckCnf_t$, say $PckCnf_t^{(0)}, \ldots, PckCnf_t^{(n-1)}$, where $PckCnf_t^{(i)}$ specifies the collection of packets at the queue of edge $e^{(i)}$ for all $e \in E$ (here we say that $PckCnf_t^{(i)}$ is a copy of $PckCnf_t$ because there is packet of type l in $PckCnf_t$ at e's queue if and only if there is packet of type $(l+i) \bmod n$ in $PckCnf_t^{(i)}$ at $e^{(i)}$'s queue).

We claim that $PckCnf_t(\mathcal{N}') = C_t$ and $SwtCnf_t(\mathcal{N}') = S_t$. The proof is by induction on t. By construction, it is immediate that the claim holds for $t=0$. By induction hypothesis and the fact that S is a shift oblivious policy, it follows that at time step $t-1$ a packet of type l crosses edge e of G if and only if for all $i \in \mathcal{I}$ a packet of type $(l+i) \bmod n$ crosses edge $e^{(i)}$ of G'. Hence, the claim also holds at time step t. □

In token networks, at each time step edges may posses a *token*. Only edges that hold a token in a given time step may be traversed by a packet. Thus, tokens represent scarce resources (e.g. a computational resource) for which packets compete. Edges that might hold a given token are said to be *serviced* by the token.

In particular, we consider networks where tokens "move" around, i.e. a token that at time t is at a given edge e can be identified with a token that at time $t' > t$ is at some edge e'. The network is said to have *latency* Δ if $t' > t + \Delta$ whenever $e \neq e'$. In such networks, in order for a token to move from an edge e to another edge $e' \neq e$ it must be absent from the network for a time interval of length $\Delta \geq 0$.

Edges that hold a token every time step will be referred to as *permanent edges*. Non-permanent edges will be called *temporary edges*.

A *token switching policy* specifies, for each token and each time step, the subset of edges that hold tokens. For example, a FIFO token switching policy is one where priority is given to the edge (among those that can grab a token) that has not held a token for the largest amount of time. A *token scheduling protocol* is a (P, S) pair that specifies policies P and S for packet and token switching, respectively.

Conceptually, one might distinguish tokens according to the edges they might service. Call a token switching policy (and the associated scheduling protocol) *disjoint* if the collection of edges serviced by distinct tokens are disjoint.

For token networks, the adversarial model is exactly the one of AQT, but now, in order to prevent the adversary from trivially congesting the network, one imposes the following *load constraint* on the adversary: for any time period $[t_1, t_2)$ the number of packets injected into the network *that are destined to be serviced by a given token* is bounded by $r(t_2 - t_1) + b$. An adversary meeting such a constraint is also referred to as an (r, b) adversary. In analogy to network systems, we define a *disjoint token network system* as the tuple $(G, \Delta, (P, S); A)$ where G is a token network with latency Δ, P is a packet scheduling policy, S is a disjoint token scheduling protocol, and A is the adversary that specifies the packet injection pattern. Other standard notions, such as for example stability, are easily adapted to the token network scenario.

Our next result formally establishes our claim concerning the simulatability of token networks by dynamic multi-type routing networks. Intuitively, a token corresponds to a network "bottleneck". In our simulation of token networks by multi-type routing networks we map all edges serviced by a token to a single "bottleneck" edge. Packets that in the original network need to traverse distinct edges will be mapped to distinct types in the new routing network. The setup incurred by a token when going from servicing one edge to another one in the token network will end up associated to a change of packet type being serviced at the "bottleneck" edges of the multi-type routing network.

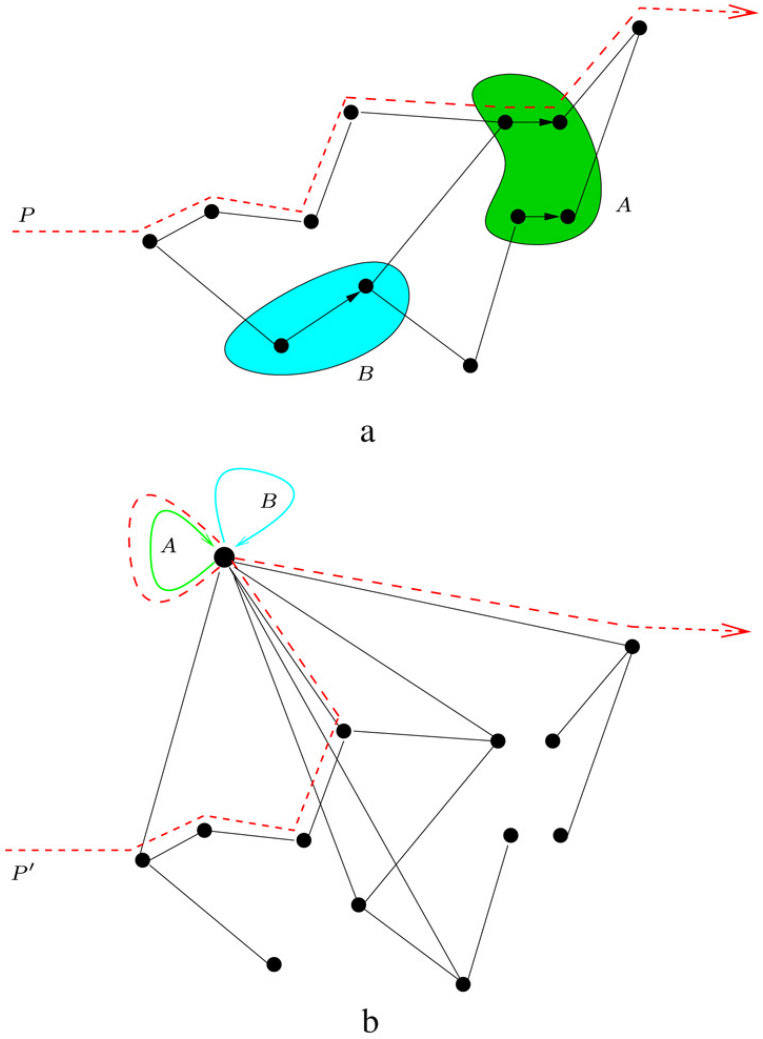

Fig. 1. (a) Token network and token switching policy pair (G, S). Directed edges are shown as arrows. Tokens are denoted by A and B. The edges they service are completely contained in the areas labeled A and B respectively. An injection path P is represented by a segmented line. (b) Multi-type routing network and switching policy pair (G', S'). The injection path to which P gets mapped is represented by a segmented line and denoted P'.

Before precisely stating our simulatability claim, we accurately describe the transformation on which it relies, and which maps a token network and token switching policy pair (G, S) into a multi-type routing network and switching policy pair (G', S'). The network G' is obtained from G by "collapsing" all edges serviced by a given token. The node set of G' will be the node set of G plus a special node v'. The edge set of G' will correspond to the collection of: (i) permanent edges of G, (ii) one loop on v' for each of the tokens of G, (iii) edges uv' for each edge uv of G such that v has a temporary edge emanating from it in G, and (iv) edges $v'v$ for each edge uv of G such that u has a temporary edge incident to it in G. See Fig. 1 for an example of a (G', S') obtained from a given pair (G, S). The collection of types of packets traversing G' will be in one-to-one correspondence with the edges of G. The switching policy S' of G' is obtained from the token switching policy S in the natural way (i.e. a token moving from edge e to e' in G is "simulated" by the edge associated to that token as going from servicing type e to servicing type e').

Note that a path P in G is mapped in a "natural" way to a path P' in G', specifically, to the path that visits the same sequence of nodes as P does, but where each head or tail of a temporary edge is replaced by the special node v'. Note that the path P' may not be a simple path even if P is simple. Also note that P and P' have the same length.

Theorem 2. *Let $\mathcal{N} = (G, \Delta, (P, S); A)$ be a disjoint token network where A is an (r, b) adversary. Then, there is a dynamic multi-type (r, b) adversary A' such that one can associate to the network system $\mathcal{N}' = (G', \Delta, (P, S'); A')$ an injective mapping φ from packet configurations of $\mathcal{N}$ to packet configurations of $\mathcal{N}'$, such that for any time step t the packet configuration of $\mathcal{N}$ is $PckCnf_t$ if and only if the packet configuration of $\mathcal{N}'$ is $\varphi(PckCnf_t)$. Moreover, if (P, S) is work conserving, so is (P, S').*

Proof. Associate to each edge of G a distinct type. In particular, denote by E the edge set of G, or equivalently the collection of types of packets traversing G'.

For each packet injected along path P in the original network by adversary A, a packet will be injected by A' in G' along path P'. Observe that two packets that wish to traverse (in G) distinct edges serviced by a given token will be in correspondence with a pair of packets of distinct types that wish to traverse the same edge of G'.

Assume packet p is injected along path $P = e_1 \dots e_\ell$ and that $P' = e'_1 \dots e'_\ell$. Consider the traversal type function associated to p's injection path in G' given by $f'_p : \{1, \dots, \ell\} \to E$ such that $f'_p(k) = e_k$. Now, let G''s packet scheduling policy be the one of G.

The desired conclusion follows since at any given time step t a packet of type $e \in E$ that wants to traverse edge e' of G' is in one to one correspondence with a packet that at the same step t wishes to traverse edge e in G. Moreover, two packets of distinct type $e_1, e_2 \in E$ that wish to traverse edge e' of G' during time step t, correspond to two packets that wish to traverse during the same step t distinct edges e_1 and e_2 of G which are serviced by the same token. □

The converse of Theorem 2 is trivially true; simply replicate each network edge, once for each possible type, and associate a single token to all copies of the same edge. The injection of a packet p in the original network can be simulated by an injection of a packet p' in the new network following the same sequence of nodes as p would travel. However, p' traverses the copy of an edge e associated to type i provided p is assigned type i when traversing e. The following result immediately follows.

Theorem 3. *Let $\mathcal{N} = (G, \Delta, (P, S); A)$ be a network system where A is a dynamic multi-type (r, b) adversary. Then, there is a multi-type (r, b) adversary A' to which one can associate a disjoint token network $\mathcal{N}' = (G', \Delta, (P, S'); A')$ and an injective mapping φ from packet configurations of $\mathcal{N}$ to packet configurations of $\mathcal{N}'$, such that for any time step t the packet configuration of $\mathcal{N}$ is $PckCnf_t$ if and only if the packet configuration of $\mathcal{N}'$ is $\varphi(PckCnf_t)$. Moreover, if (P, S) is work conserving, so is (P, S').*

3. Stability results

In this section we characterize stable networks. As in the standard AQT model, the characterization is based on the universal stability of directed acyclic graphs and directed rings.

To establish the universal stability of directed acyclic graphs and directed rings we rely on potential function type arguments similar to those used to prove [9, Theorem 1] and [5, Theorem 3.7]. As is usually the case for these types of proof arguments, we upper bound the potentials by a term which is independent of time, thence obtaining the desired conclusions. However, the bounds we derive need to take into account the setup cost Δ. Intuitively, if an edge is not highly congested it does not matter much what type of packet is being serviced. However, for congested edges, a switching policy should choose to service a packet type if it allows for efficient use of the edge (transmit 1 packet per time step) for a sufficiently long time interval. The length of the latter interval, say θ, will necessarily have to be a function of Δ. A moment of though also allows one to see that it must depend on the adversaries rate r and burst b parameters. Indeed, one expects that θ should be larger, the larger r, b and Δ are. A sensible switching policy S_θ, for sufficiently large θ, guarantees that the inefficiencies entailed by setup costs will be amortized by an edge working at full capacity over a sufficiently long interval after the setup cost is incurred. A sensible switching policy can thus avoid paying unnecessary setup costs and maintain network stability against (r, b) adversaries for rates arbitrarily close to 1. The potential functions introduced in the next two sections are adaptations of the potential functions defined in [9] and [5]. But, their derived upper bounds now also depend explicitly on the value of θ, and hence indirectly on r, b, and Δ.

3.1. Directed acyclic graphs

Borodin et al. [9] established that acyclic networks are universally stable in the adversarial setting. We generalize their result to the multi-type with setups scenario. As mentioned before the argument we will use is a potential function argument. In the context of acyclic networks it is natural to consider potential functions which take into account the network state upstream of any given network edge e. Specifically, we will consider the total number of packets present in the network that want to traverse a given edge e at a given time step t. We will show that such a quantity remains bounded over time. As one might expect, the bound will depend on the structure of the network upstream of e. Specifically, we have the following result:

Theorem 4. *For every acyclic digraph G, all $\Delta \geq 0$, every greedy scheduling protocol (P, S), and every (r, b) adversary A where r is* strictly *less than* 1, *there exists a $\theta > 0$ such that $(G, \Delta, (P, S_\theta); A)$ is stable.*

Proof. For each edge e of G let $Q_{e,i}(t)$ be the number of packets of type i at e's queue at time step t. Let $Q_e(t)$ denote the total number of packets at e's queue, i.e. $Q_e(t) = \sum_{i \in \mathcal{I}} Q_{e,i}(t)$. Furthermore, let θ be large enough so that $r(\theta + 2\Delta) + b \leq \theta$.

We inductively define the function $\psi(\cdot)$ as follows: for each edge e denote by $f_1, \dots, f_k$ the edges incident to e's tail node and define

$$\psi(e) = \max\{(|\mathcal{I}| + 1)\theta, Q_e(0)\} + \sum_{j=1}^{k} \psi(f_j).$$

Denote by $N_e(t)$ the number of packets present in the network that want to traverse edge e and were injected by time t. We claim that $N_e(t) \leq \psi(e)$ for all e and for all $t = l(\theta + 2\Delta)$ with $l \in \mathbf{N}$. The desired result will follow once we prove this claim. Indeed, since $N_e(\cdot)$ can grow only by injection of new packets, one has $N_e(t') \leq N_e(t) + r(\theta + 2\Delta) + b$ for $t \leq t' < t + (\theta + 2\Delta)$. Since $\sum_e \psi(e)$ is an upper bound on the total number of packets in the graph, the theorem follows.

First we show that when e is a source edge and $t = l(\theta + 2\Delta)$,

$$Q_e(t) \leq \max\{(|\mathcal{I}| + 1)\theta, Q_e(0)\}. \tag{3}$$

Indeed, consider the following two cases:

- $Q_{e,i}(t) \geq \theta$ for some $i \in \mathcal{I}$: In this case, the number of packets z that cross e during the interval $[t, t + \theta + 2\Delta)$ is at least $\theta \geq r(\theta + 2\Delta) + b$. We get that

$$Q_e(t + (\theta + 2\Delta)) \leq Q_e(t) + r(\theta + 2\Delta) + b - z \leq Q_e(t).$$

- $Q_{e,i}(t) < \theta$ for all $i \in \mathcal{I}$: Clearly, $Q_e(t) \leq |\mathcal{I}|\theta$, so by the choice of θ and the induction hypothesis

$$Q_e(t + (\theta + 2\Delta)) \leq Q_e(t) + r(\theta + 2\Delta) + b \leq |\mathcal{I}|\theta + \theta.$$

Now, to show that $N_e(t) \leq \psi(e)$ for all e and $t = l(\theta + 2\Delta)$ with $l \in \mathbf{N}$, we proceed by induction on the maximum distance of the tail of e to a source of G (i.e. the length of the longest directed path from any source to the tail of e). The base case of the induction is when the tail of e is a source. In this case $N_e(t) = Q_e(t)$ and $\psi(e) = \max\{(|\mathcal{I}| + 1)\theta, Q_e(0)\}$. Hence, the claim holds because of (3). Now, assume the maximum distance from a source of G to the tail of e is $\ell > 0$. Let again $f_1, \ldots, f_k$ denote the edges incident to e's tail node. Consider the following two cases:

- $Q_{e,i}(t) \geq \theta$ for some $i \in \mathcal{I}$: In this case, the number of packets z that cross e during the interval $[t, t + \theta + 2\Delta)$ is at least $\theta \geq r(\theta + 2\Delta) + b$. We get that

$$N_e(t + (\theta + 2\Delta)) \leq N_e(t) + r(\theta + 2\Delta) + b - z \leq N_e(t).$$

- $Q_{e,i}(t) < \theta$ for all $i \in \mathcal{I}$: Clearly, $Q_e(t) \leq |\mathcal{I}|\theta$, so by the choice of θ

$$\begin{aligned} N_e(t + (\theta + 2\Delta)) &\leq Q_e(t) + r(\theta + 2\Delta) + b + \sum_{j=1}^{k} N_{f_j}(t) \\ &\leq (|\mathcal{I}| + 1)\theta + \sum_{j=1}^{k} N_{f_j}(t). \end{aligned}$$

Since $N_e(0) = Q_e(0) + \sum_{j=1}^{k} N_{f_j}(t)$ and by induction hypothesis, we conclude that for $t = l(\theta + 2\Delta)$,

$$N_e(t) \leq \max\{(|\mathcal{I}| + 1)\theta, Q_e(0)\} + \sum_{j=1}^{k} N_{f_j}(t) \leq \psi(e). \quad \square$$

Corollary 1. *Acyclic digraphs are universally stable.*

3.2. Rings

Andrew et al. [5] established that the simplest non-acyclic networks (directed rings) are universally stable in the adversarial setting. We extend their result to the multi-type with setups scenario. Roughly, the potential function we now consider is the number of packets that needs to traverse an edge e and are on queues with at least θ packets of its same type. By definition, there are few packets that need to traverse edge e that are not taken into account by such a potential. The intuition is that the potential should not take arbitrary values given that all packets taken into account by it at a given time step t are at queues of edges that in an interval close to t transmit one packet per time step.

Henceforth, let $n > 1$ and R_n denote the n-node directed ring. Without loss of generality, assume that R_n's nodes are labeled $0, 1, \ldots, n-1$ and that its edges are directed from i to $i+1$ (all arithmetic on labels is modulo n). The edge emanating from node i will be referred to in what follows as edge i.

Theorem 5. *Let $\Delta \geq 0$. For every greedy scheduling protocol (P, S) and (r, b) adversary A where r is* strictly *less than* 1*, there exists a $\theta > 0$ such that $(R_n, \Delta, (P, S_\theta); A)$ is stable.*

Proof. Choose θ so that $(r + \varepsilon) = \theta/(\theta + 2\Delta)$ for some $\varepsilon > 0$. Let $P_{e,i}(t)$ be the collection of packets of a given type $i \in \mathcal{I}$ which need to traverse edge e and at time step t (before injections take place) are on a queue with at least θ packets of type i. Denote by $P_e(t)$ the union over $i \in \mathcal{I}$ of all $P_{e,i}(t)$'s.

Let Q' be a constant large enough whose value will be fixed appropriately later on. For the sake of contradiction assume that $(R_n, \Delta, (P, S_\theta); A)$ is not stable. Hence, there must exist a T' such that $|P_s(T')| > Q'$ for some s. Let T' be the smallest such value and

$$Q = \max\left\{|P_e(t)| : e \in \{0, \ldots, n-1\}, t \in [T_0, T')\right\}.$$

For $q \in P_s(t)$ denote by l_q the maximum positive integer j such that $q \in P_s(t-j)$ for all $j \in \{0, 1, \ldots, l_q\}$. Let p be a packet of $P_s(T')$ such that $l_p = \max\{l_q : q \in P_s(T')\}$. Let $T_0 = T' - l_p$ and assume without loss of generality that p is at the queue emanating from node 0 at time step T_0. In other words, assume $0, 1, \ldots, s$ is the sequence of nodes visited by p during the interval $[T_0, T']$. Denote by T_e the moment at which p arrives at e's queue. For the sake of convenience, let $T_{s+1} = T'$.

Note that for all $e \in \{0, 1, \ldots, s\}$ and $t \in [T_e, T_{e+1})$ it holds that $Q_{e,i}(t) \geq \theta$ for some type i. Hence for any subinterval of $[T_e, T_{e+1})$ of length $\theta + 2\Delta$ at least θ packets must cross e.

Consider now and edge e, a time step $t \in [T_e, T_{e+1})$, and let $t' \leq t$. A packet $p \in P_e(t)$ could either; (i) belong to $P_e(t')$, (ii) have been injected during the interval $[t', t)$, or (iii) not belong to $P_e(t')$ and by time t become part of a queue of size at least θ of packets of the same type (either by joining such a queue or by arrival to its queue of other equal type packets). The number of each of the latter group of packets is bounded by $|P_e(t')|$, $r(t-t') + b$, and $n|\mathcal{I}|\theta$ respectively. Hence,

$$|P_e(t)| \leq |P_e(t')| + r(t-t') + b + n|\mathcal{I}|\theta - z, \tag{4}$$

where z is the number of packets sent across edge e during the interval $[t', t)$.

For $e \in \{0, \ldots, n-1\}$ and $t \geq T_0$, define

$$f(e,t) = \begin{cases} Q + e(b + n|\mathcal{I}|\theta), & \text{if } t = T_0, \\ Q - \varepsilon(t - T_0) + (e+1)(b + n|\mathcal{I}|\theta), & \text{if } t > T_0. \end{cases}$$

Say that the pair (e, t) is *applicable* if either one of the following two conditions hold:

- $e \in \{0, 1, \ldots, s\}$ and $t \in [T_0, T_{e+1}]$, or
- $e > s$ and $t \in [T_0, T']$.

We claim that $|P_e(t)| \leq f(e,t)$ if the pair (e, t) is applicable.

Clearly, for all $t \geq t' > T_0$,

$$f(e,t) = f(e,T_0) - \varepsilon(t - T_0) + b + n|\mathcal{I}|\theta, \tag{5}$$

$$f(e,t) = f(e,t') - \varepsilon(t - t'). \tag{6}$$

Moreover, if in addition $e > 0$,

$$f(e,t) = f(e-1,t) + b + n|\mathcal{I}|\theta. \tag{7}$$

By definition of f and Q we have that $f(e,T_0) = Q + e(b + n|\mathcal{I}|\theta) \geq Q \geq |P_e(T_0)|$. Hence, the claim holds for (e,t) when $t = T_0$.

Say that an applicable pair (e,t) is good if

$$t' \in [T_0, t) \Longrightarrow \exists i \in \mathcal{I}, Q_{e,i}(t') \geq \theta.$$

Assume (e,t) is a good applicable pair. By definition, for every $t' \in [T_0, t)$ there is a type i such that e's queue contains at least θ packets of type i. In particular, during a subinterval of $[T_0, t)$ of length $\theta + 2\Delta$ and given that S_θ is sensible, at least θ packets (of the same type) must cross e. Thus, if z is the number of packets sent across edge e during the interval $[T_0, t)$, then $z \geq (t - T_0)\theta/(\theta + 2\Delta)$. It follows, by (4), because $f(e,T_0) \geq Q \geq |P_0(T_0)|$, and (5), that

$$\begin{aligned} |P_e(t)| &\leq |P_e(T_0)| + r(t - T_0) + b + n|\mathcal{I}|\theta - (r + \varepsilon)(t - T_0) \\ &\leq f(e,T_0) - \varepsilon(t - T_0) + b + n|\mathcal{I}|\theta \\ &= f(e,t). \end{aligned}$$

Hence, the claim holds for every good applicable pair.

Observe now that any $(0,t)$ applicable pair is good. Indeed, t must belong to $[T_0, T_1]$ for $(0,t)$ to be applicable. Moreover, since p is at 0's queue during the interval $[T_0, T_1)$, by the choice of p, it follows that for i equal to p's type, $Q_{0,i}(t') \geq \theta$ when $t' \in [T_0, t)$.

We still need to prove that the claim is true for non-good applicable pairs (e,t) where $e > 0$ and $t > T_0$. We proceed by induction on e. Assume that $|P_{e-1}(t)| \leq f(e-1,t)$ for all applicable $(e-1,t)$. Let $T_0 \leq t' \leq t$ be the largest time step such that $Q_{e,i}(t') < \theta$ for all $i \in \mathcal{I}$. Note that, $t' \neq T_0$ and $|P_e(t')| = |P_{e-1}(t')|$. By (4), we get that

$$\begin{aligned} |P_e(t)| &\leq |P_e(t')| + r(t - t') + b + n|\mathcal{I}|\theta - (r + \varepsilon)(t - t') \\ &= |P_{e-1}(t')| - \varepsilon(t - t') + b + n|\mathcal{I}|\theta. \end{aligned}$$

Now note that $(e-1, t')$ is applicable. Indeed, since (e,t) is applicable, then $t' \leq t \leq T'$. Thus, if $e > s$, then $(e-1, t')$ is applicable. So suppose $e \in \{1, \ldots, s\}$. In particular, $t' \leq t \leq T_{e+1}$ because (e,t) is applicable. By the choice of p and given that p is at e's queue during the interval $[T_e, T_{e+1})$, it follows that for i equal to p's type, $Q_{e,i}(t'') \geq \theta$ whenever $t'' \in [T_e, T_{e+1})$. But $Q_{e,i}(t') < \theta$ for all $i \in \mathcal{I}$, so it must hold that $t' < T_e$, i.e. $(e-1, t')$ is applicable. Hence,

$$\begin{aligned} |P_e(t)| &\leq f(e-1,t') - \varepsilon(t - t') + b + n|\mathcal{I}|\theta \\ &= f(e-1,t) + b + n|\mathcal{I}|\theta \\ &= f(e,t), \end{aligned}$$

where the first equality is by (6) and second one is by (7). The claim is thus proved.

By the choice of T', the fact that (s, T') is applicable and the definition of f,

$$Q' < |P_s(T')| \leq f(s, T') = Q - \varepsilon(T' - T_0) + (s+1)(b + n|\mathcal{A}|\theta).$$

By definition of T' we have that $Q \leq Q'$ so we get that $\varepsilon(T' - T_0) < (n+1)(b + n|\mathcal{A}|\theta)$. However, packets in $P_s(T')$ either were injected during the interval $[T_0, T']$ or were in the network before time step T_0. There can be at most $r(T' - T_0 + 1) + b$ of the former type of packets. To upper bound the number of packets of the latter type, note that each such packet must be in a queue of length smaller than θ during some time step $t \in [T_0, T')$. Hence, there are at most $n|\mathcal{A}|\theta(T' - T_0)$ such packets. We get that

$$Q' \leq |P_s(T')| \leq r(T' - T_0 + 1) + b + n|\mathcal{A}|\theta(T' - T_0).$$

Since $\varepsilon(T' - T_0) < (n+1)(b + n|\mathcal{A}|\theta)$, the sought-after contradiction follows taking

$$Q' = (1/\varepsilon)(r + n|\mathcal{A}|\theta)(n+1)(b + n|\mathcal{A}|\theta) + r + b. \quad \square$$

Corollary 2. *The n-node directed ring network is universally stable.*

3.3. Characterization of stable networks

Since the AQT with setups model is an extension of the classical AQT model, if G is not universally stable under the latter model, then it is not universally stable in the former. Hence, all instability results in [2] hold in the AQT with setup scenario.

We now characterize networks which are universally stable in the AQT with setups model. The proof argument is an adaptation to the AQT with setup scenario of an argument due to Goel [22]. First, one establishes that universal stability depends on whether or not each of the networks strongly connected components is universally stable.

Lemma 1. *Let G_1 and G_2 be digraphs and G' be also a digraph formed by joining G_1 and G_2 with directed edges going from nodes in G_1 to nodes in G_2. If G_1 and G_2 are universally stable in the AQT with setup model, then so is G'.*

Proof. Fix the setup value and the scheduling protocol. Let $r < 1$ and consider an (r, b) adversary A' injecting packets in G'. Since G_1 is universally stable against the adversary induced by A' when restricted to G_1, there is a constant C_1 (depending on (r, b), the setup cost, the size of G_1 and its initial configuration) which is an upper bound on the total number of packets in G_1 at any given step.

For convenience, we refer to the edges of G' going from G_1 to G_2 as bridges. Let $\widetilde{G}$ be the network G_1 plus all bridges. Now consider a time interval $[t_1, t_2)$ and an (r, b) multi-type adversary injecting packets into $\widetilde{G}$. Any packet that requires crossing a given bridge e during $[t_1, t_2)$ must have been injected in $\widetilde{G}$ during the interval, was present in G_1 at the start of the period, or was one of the C_e packets at e's queue initial configuration. The number of packets of the first type is upper bounded by $r(t_2 - t_1) + b$ and both of the latter types by $C_1 + C_e$. Hence, at most $r(t_2 - t_1) + (b + C_1 + C_e)$ packets require crossing any given bridge during the interval $[t_1, t_2)$. Since by Theorem 4 acyclic graphs are universally stable, we conclude that the queues of bridges are bounded independent of time by a constant (depending on (r, b), the setup cost, the scheduling protocol's parameters, the size of $\widetilde{G}$ and its initial configuration). It immediately follows that the queues of all edges in $\widetilde{G}$ are also bounded by a constant. In particular, there is a constant $\widetilde{C}$ that bounds the total number of packets in $\widetilde{G}$ independent of time.

Now, let A_2 be the adversary that injects into G_2 all packets with destination in G_2 that A' injects into G'. If the packet's source node is also in G_2 the injection time and traversal path specified by A_2 is the same as the one specified by A'. If the packet's source node is in G_1 the traversal path specified by A_2 is the restriction to G_2 of the traversal path specified by A' and the moment of injection is the time step where the given packet would have left $\widetilde{G}$. Note then, that all packets injected by A_2 traversing a given edge of G_2 during an interval $[t_1, t_2)$ had to be either injected by A' during the same interval or had to be in a queue of $\widetilde{G}$ at the start of the period. There are at most $r(t_2 - t_1) + b$ of the former packets and, by the preceding paragraph's conclusion, at most $\widetilde{C}$ of the latter. It follows that A_2 is an $(r, b + \widetilde{C})$ adversary. Since G_2 is universally stable, we conclude that there is a constant C_2 independent of time that bounds the total number of packets in G_2. It immediately follows that the number of packets in G' remains bounded by $\widetilde{C} + C_2$ when the injection pattern is under the control of A'. $\square$

An immediate consequence of the previous result is,

Theorem 6. *In the AQT with setup model, a digraph G is universally stable if and only if all its strongly connected components are universally stable.*

The previous result allows us to focus our study of stability exclusively on strongly connected networks. Theorem 5 tells us that that directed rings are universally stable. Note that a strongly connected network that is not a directed ring must contain as a subgraph (not necessarily induced) one of the networks shown in Figs. 2 and 3 where edges might be replaced by node disjoint directed paths. We will show that all these latter networks are unstable, and thus obtain a characterization of universally stable networks for the AQT model with setups.

Recall that a characterization of universally stable networks under the AQT model is given by Álvarez, Blesa and Serna [2]. In it, a collection of simple non-stable digraphs is first identified. Then, two subdivision operations are introduced; (1) edge subdivision, and (2) cycle subdivision. Subdivision of an edge consists on the addition of a new vertex w and the replacement

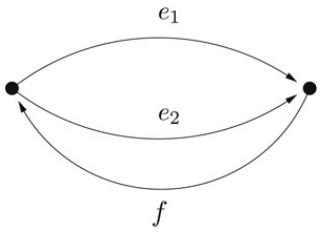

Fig. 2. U_1.

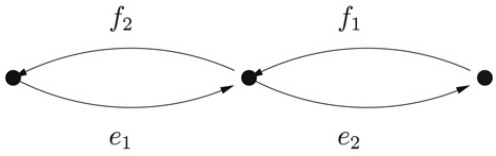

Fig. 3. U_2.

of the directed edge (u, v) by the two directed edges (u, w) and (w, v). Subdivision of a 2-cycle consists of the addition of a new vertex w and the replacement of the two directed edges $\{(u, v), (v, u)\}$ by the directed edges (u, w), (w, u), (v, w) and (w, v). Applying these operations to non-stable digraphs yields non-stable digraphs. Let $\varepsilon(G)$ denote the collection of digraphs obtained from G through zero or more subdivision operations and consider the graphs defined in Figs. 2 and 3.

Theorem 7. *In the AQT model with setups, a digraph is universally stable if and only if it does not contain as a subgraph a digraph in* $\varepsilon(U_1) \cup \varepsilon(U_2)$.

Proof. Suppose G is universally stable and contains as a subgraph $H \in \varepsilon(U_1) \cup \varepsilon(U_2)$. All subgraphs of a universally stable network must be universally stable. This contradicts the fact that digraphs in $\varepsilon(U_1) \cup \varepsilon(U_2)$ are unstable under the NEAREST-TO-GO/LONGEST-IN-SYSTEM packet scheduling protocol even in the classical AQT model without setups [2, Theorem 3].

If G does not contain as a subgraph a digraph in $\varepsilon(U_1) \cup \varepsilon(U_2)$, then all of its strongly connected components are simple cycles. Since we know by Theorem 4 and Theorem 5 that acyclic digraphs and directed cycles are universally stable, applying Theorem 6, we conclude that G is universally stable. □

4. Instability results

In this section we prove that stability does not trivially carry over from the single-class to the multi-class setting. Specifically, we establish the following result.

Theorem 8. *For all* $0 < r < 1$, $\Delta \geq 0$, $\theta > 0$, *and every greedy scheduling protocol* (P, S) *where* S *is shift-oblivious, there exists a network* G *(whose size depends on* r, Δ *and* θ*), and an* $(r, 1)$ *adversary* A, *such that* $(G, \Delta, (P, S_\theta); A)$ *is unstable.*

The network whose existence is guaranteed by Theorem 8 is built from a basic gadget which "slows" packets that need to traverse it. These delays allow an adversary to inject packets so as to make them arrive simultaneously at some other network spot. The network and the adversary are simplifications of those presented in [8] in order to show instability of FIFO at arbitrarily small rates. For the sake of clarity of exposition, we henceforth only consider the case $\Delta = 0$. Our arguments can be easily adapted to the case where $\Delta > 0$.

The types and the labeling of types: We require the collection of types $\mathcal{I}$ to be of cardinality at least 4. Since S is shift-oblivious, there exists an identification between $\mathcal{I}$ and $\{0, 1, 2, 3\}$ such that if on a given queue state and while serving type i packets S chooses to serve next type j packets, then when every packet of type i is replaced by one of type $i + 1$ (arithmetic on types is modulo 4), we will have that when serving type $i + 1$ packets S will choose to serve next type $j + 1$ packets. For the sake of convenience it will be better to refer to types 2 and 3 by $\widetilde{0}$ and $\widetilde{1}$ respectively.

The network: The basic structure of the network is called a *gadget*. A gadget has nodes v_0 and v_1, and its edges are of three distinct types; (1) pairs of input edges i_0, i_1, (2) pairs of output edges o_0, o_1, and (3) two load edges e_0 and e_1 with e_j going from v_j to $v_{j'}$, where throughout this section j' denotes 1 if $j = 0$ and 0 if $j = 1$. Note that load edges form a two node unidirectional ring with. (Two gadgets are depicted in Fig. 4.)

We say that a gadget G_2 is concatenated to gadget G_1 if a pair of output edges of G_1 corresponds to a pair of inputs edges of G_2 (see Fig. 4 for an illustration of two concatenated gadgets). One gadget can be concatenated to several gadgets and more than one gadget can be concatenated to a gadget. Either one of these situations can be realized through multiple pairs of input and/or output edges. We say that $C = \langle H_1, H_2, \ldots, H_n \rangle$ is a *chain of length* n if H_{i+1} is concatenated to H_i. A chain $B = \langle G_1, H_1, \ldots, H_m, G_2 \rangle$ will be called a *bridge* of length m between G_1 and G_2.

The main components of the network G we are going to construct are: columns, connectors and shortcuts. There are two columns C_0 and C_1, each one is a chain of length α, say

$$C_a = \langle C_{a,1}, C_{a,2}, \ldots, C_{a,\alpha} \rangle.$$

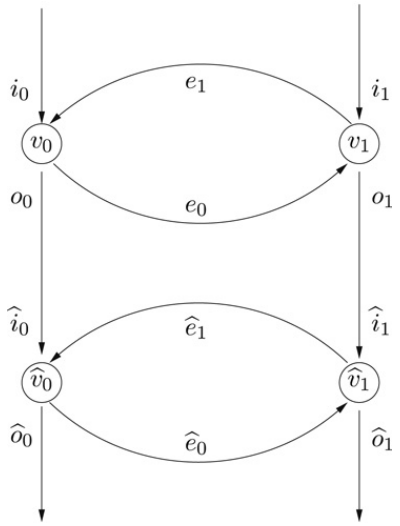

Fig. 4. Two concatenated gadgets.

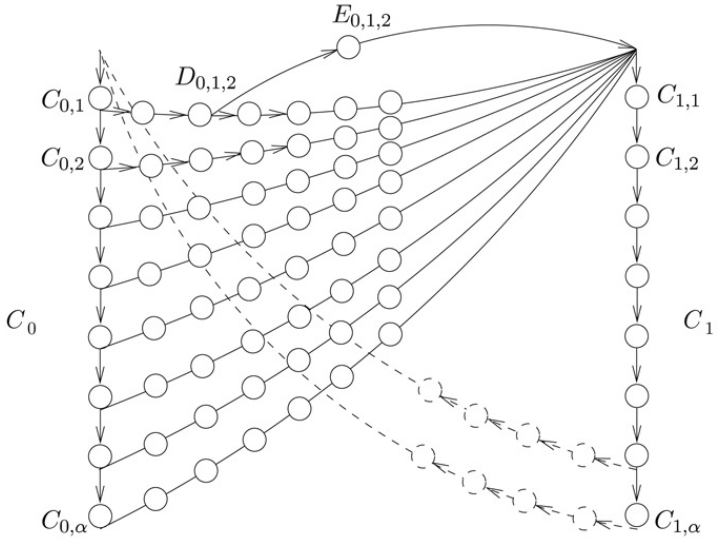

Fig. 5. Network G.

Connectors are bridges of length β between each gadget of a column and the other column's first gadget; say for each $C_{a,i}$ there is a bridge

$$\langle C_{a,i}, D_{a,i,1}, D_{a,i,2}, \ldots, D_{a,i,\beta}, C_{a',1}\rangle.$$

The values of α and β will be fixed later. Finally, shortcuts are bridges of length 1 from each connector gadget $D_{a,i,j}$ to $C_{a',1}$ and they are denoted by $E_{a,i,j}$. We stress that all components of type $C_{a,i}$, $D_{a,i,j}$, and $E_{a,i,j}$ are gadgets. An illustration of the network just described is schematically given in Fig. 5.

The adversary: We define an adversary A that roughly proceeds in the following way: first, we assume that there are l packets going down through a column, say C_0. While these packets are crossing a gadget $C_{0,i}$ of column C_0, the adversary injects packets that:

- First, use one of the gadget's $C_{0,i}$ load edges.
- Afterwards, want reach the first gadget $C_{1,1}$ of column C_1 partly traversing the connector $\langle C_{0,i}, D_{0,i,1}, D_{0,i,2}, \ldots, D_{0,i,\beta}, C_{1,1}\rangle$ concatenated to gadget $C_{0,i}$.
- Jumps through a shortcut $E_{0,i,j}$ (chosen in a way that will be described shortly).
- Finally, traverse down through column C_1.

The adversary uses shortcuts in order to arrange for $l' > l$ packets to arrive continuously at each of the input edges of the first gadget of column C_1. Replacing l by l' and repeating the injection pattern the adversary creates instability in G.

Let us now precisely specify how adversary A injects packets in G. A packet is called *gadget-traversing* if for every gadget that it crosses, it enters through an input edge i_k, sequentially traverses e_k and $e_{k'}$, and finally exits the gadget through one

of its o_k output edges. A packet is called *chain-traversing* for a chain $\langle H_1, H_2, \ldots, H_k\rangle$ if its route is gadget-traversing for each H_j.

The adversary injects only the 4 different packet types 0, $\widetilde{0}$, 1, and $\widetilde{1}$. Types 0 and 1 correspond to packets that are chain-traversing for column C_0, while packets of types $\widetilde{0}$ and $\widetilde{1}$ are chain-traversing for column C_1. Moreover, gadgets of type $D_{0,i,j}$ (respectively $D_{1,i,j}$) are traversed by packets of type $\widetilde{0}$ and $\widetilde{1}$ (respectively of type 0 and 1) only.

To describe the packet injection pattern we introduce the notion of *gadget activation* which encompasses three phases: precondition, injection and postcondition. We now describe each of these phases for column gadgets. Fix a column a and consider its i-th gadget $C_{a,i}$.

- In the precondition phase, we assume that for l consecutive time steps there will be a gadget-traversing packet at the queue of each of $C_{a,i}$'s input edges (here l is a parameter that might depend on time). Moreover, all such packets will be C_a traversing, in particular $\langle C_{a,i}, C_{a,i+1}, \ldots, C_{a,\alpha}\rangle$ chain-traversing.
- The injection phase for gadget $C_{a,i}$ lasts l time steps. It starts when the first packet of the precondition phase arrives at the queue of a load edge of $C_{a,i}$. In the first step of the injection phase the adversary does not inject any packet. During the following $l^- = l - 1$ injection phase steps the adversary injects exactly $lr/2$ packets at each node v_0 and v_1 of gadget $C_{a,i}$. A packet injected in node v_k follows an injection path that starts with edge e_k, is chain-traversing for up to some shortcut $E_{a,i,j}$ of the connector concatenated to $C_{a,i}$, is gadget-traversing for $E_{a,i,j}$, and is chain-traversing for column $C_{a'}$. Packets injected at node v_k are of type $\widetilde{k'}$ if v_k belongs to column C_0 and of type k' if v_k belongs to column C_1. In particular, note that packets injected in column C_0 are of type $\widetilde{0}$ and $\widetilde{1}$ while packets injected in column C_1 are of type 0 and 1. Also note that at the end of an injection phase all packets of the precondition phase have traversed at least one load edge of $C_{a,i}$.
- The postcondition phase of gadget $C_{a,i}$ will correspond to the precondition assumption for the activation of gadget $C_{a,i+1}$.

The notion of gadget activation for connector gadgets is defined analogously except that during the injection phase the adversary does not inject packets in the gadget.

The time interval since the first gadget traversing packet enters and leaves a given gadget will be referred to as *gadget activation period*.

Time steps will be grouped into stages. At the beginning of the s-th stage, s even, we will guarantee that for $l = l(s)$ consecutive time steps a chain-traversing (for column C_0) packet of type k will be at i_k's queue, where i_k is a k-th input edge of gadget $C_{0,1}$. We will arrange things so that for at least $2l$ consecutive times steps at the beginning of the next stage (the $(2s+1)$-th one), a chain-traversing (for column C_1) packet of type $\widetilde{k}$ will be at i_k's queue, where i_k is a k-th input edge of gadget $C_{1,1}$.

Each stage is divided into α sub-stages. During the i-th sub-stage the gadget $C_{0,i}$ is activated and for each $k \in \{0, 1\}$ a total of $lr/2$ packets of type $\widetilde{k'}$ are injected in node v_k. The routes followed by these packets were described above.

The adversary never injects packets in shortcuts or internal connector gadgets.

Instability: We now show that network G is unstable when subject to an injection pattern under the control of the adversary A described above. First, we need to establish some facts about the length of gadget-activation phases and injection patterns.

Lemma 2. *If $lr^2/4 > \theta$, then a column gadget activation period lasts either l or $l(1 + r/2)$ time steps, and a connector gadget activation period lasts $lr/2$ steps. Moreover, the injection phase of column gadgets lasts l steps and at any time step at most one gadget can be in its injection phase.*

Proof. First, consider the case of column gadgets. Without loss of generality, we assume the stage is an even one. Hence, gadgets that become activated are those of column C_0. The step just before the injection phase of gadget $C_{0,i}$ begins a packet of type k requiring load edge e_k traverses input edge i_k. Also, no packet is injected during the first step of the injection phase. Hence, only packets of type k will be awaiting to traverse edge e_k at the start of $C_{0,i}$'s injection phase. Since the scheduling protocol is greedy and given that the precondition phase guarantees that during all of the injection phase a packet of type k will be available for traversing e_k, it follows that edge e_k will serve a type k packet the first time step of the injection phase and will continue doing so for a period of total length l. In addition, during the $l^- = l - 1$ last steps of the injection phase, exactly $lr/2$ packets of type $\widetilde{k'}$ requiring edge e_k are also injected. When the injection phase finishes, there are no more packets of type k at e_k's queue. Moreover, exactly l packets of type k' and $lr/2$ packets of type $\widetilde{k'}$ are waiting at e_k's queue. The hypothesis guarantees that $l \geq lr/2 \geq \theta$. Thus, the switching protocol can choose to service next either the column traversing packets or the newly injected connector traversing packets. If the newly injected packets are served, gadget activation will last $l + lr/2$ steps. Otherwise it will last l time steps. A crucial fact that we have used here is that S is a shift-oblivious policy, so in no case one of the edges e_0 and e_1 will chose to serve column traversing packets and the other edge will chose to serve newly injected packets. Fig. 6 illustrates the two situations that can arise.

For connector gadgets the argument is the same but substituting l by $lr/2$, observing that by hypothesis $(lr/2)r/2 > \theta$ and recalling that no packets are injected in internal connector gadgets. Thus, their activation period lasts $lr/2$. □

We can now establish that A is a bounded rate adversary.

Lemma 3. *The adversary A is an $(r, 1)$ adversary.*

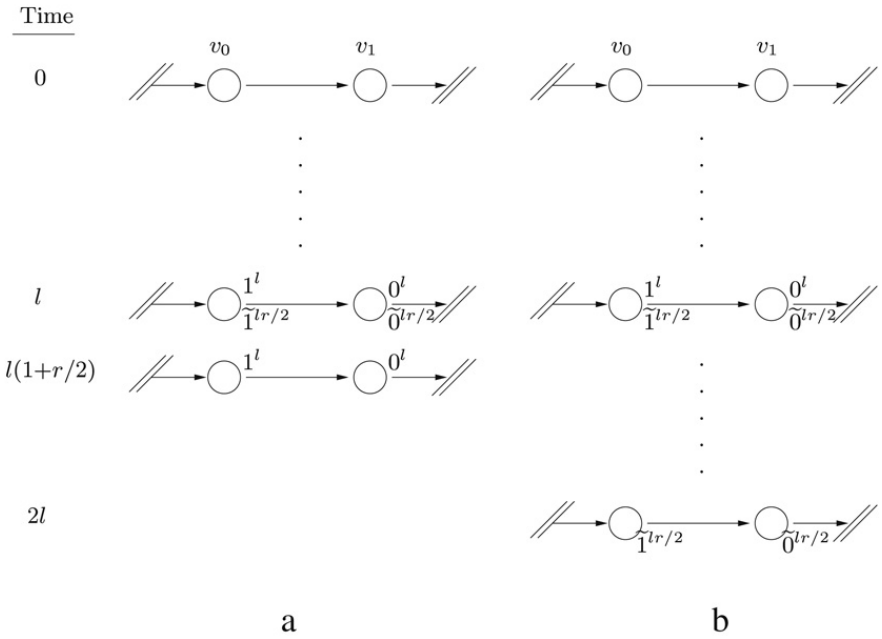

Fig. 6. State of a gadget's load edge queues during a gadget activation phase. Two situations are depicted; (a) Gadget activation phase lasting $l(1 + r/2)$ steps, and (b) Gadget activation phase lasting l steps. On the left, the time steps since the start of the gadget activation phase are displayed. Gadget nodes and load edges are shown at different times. Queues at each edge are specified by displaying p^s close to the edge's tail if s packets of type p are waiting to traverse the edge.

Proof. By Lemma 2, it suffices to note that over any interval of length l^- the adversary injects packets in at most one gadget. By the way packets are injected, we know that at most $lr/2$ of them are injected in any given node of a gadget during an interval of length l provided $lr/2 \geq 1$. It follows that over any given interval of time $[t_1, t_2)$ the adversary never injects more than $r(t_2 - t_1) + 1$ packets requiring a given edge.

We also have to make sure that during the l^- last steps of the injection phase the adversary can inject at any active gadget node exactly $lr/2$ packets. This follows because an $(r, 1)$ adversary can inject up to $rl^- + 1 > lr$ packets requiring a single edge during an interval of length l^-. Hence, it can inject $lr/2$ packets in each of the two nodes of an active gadget during the last l^- steps of an injection phase. □

Proof (*Theorem* 8). By Lemma 2 we have that the $(j+1)$-th gadget in the connector concatenated to a given column gadget $C_{a,i}$ is activated at most $l + jlr/2 \leq l + \beta lr/2$ times steps after $C_{a,i}$ is activated (at most l steps for newly injected packets to start traversing the connector concatenated to $C_{a,i}$ and $jlr/2$ steps to traverse the first j-gadgets of the bridge). Choosing $\beta \geq \alpha(1+2/r)$ so $l+\beta lr/2 \geq \alpha l(1+r/2)$ we can guarantee that all packets injected in column C_a during a given stage will not reach column $C_{a'}$ before the bottom gadget of C_a has finished its activation period. Let $\alpha \geq 4/r$. By appropriately using shortcuts we can make all $\alpha lr/2 \geq 2l$ packets of a given type injected in column C_a during a given stage to start continuously arriving at the appropriate input edge of the first gadget of column $C_{a'}$.

Summarizing, if $\beta \geq \alpha(1+2/r), \alpha \geq 4/r$, and $l \geq 4\theta/r^2$, then A is an $(r, 1)$ adversary and $(G, 0, (P, S_\theta); A)$ is unstable. □

A careful analysis of the proof of Theorem 8 shows that a weaker notion than shift-oblivious is sufficient for the proof to work. In fact, even for a switching policy such as PRIORITY one can make the proof argument go through. The exact condition on the switching policy we use its to technical to be of general interest. This explains why we settled on the weaker notion of shift-oblivious policies.

5. Fluid model

Gamarnik [20, Theorem 1] showed how stability of a queueing network can be established by considering an associated fluid model. In this section we extend this result to the multi-type with setups scenario,

Consider a network system $(G = (V, E), \Delta, (P, S); A)$. Let $\mathcal{R}$ denote the set of simple paths in G. For each path $P \in \mathcal{R}$ of length $\ell(P)$ let $e_0^P \dots e_{\ell(P)}^P$ be the sequence of consecutive edges in P. Let $A_{P,i}(t_1, t_2)$ be the total number of packets of type i that the adversary injects along path P during the interval $[t_1, t_2)$. Moreover, let $A_{e,i}(t_1, t_2)$ be the analogue quantity measured over all paths P that cross e, i.e. $A_{e,i}(t_1, t_2) = \sum_{P;e\in P} A_{P,i}(t_1, t_2)$.

Let $B_{e,i}(t_1, t_2)$ be the total number of packets of type i that arrives at the tail node of edge e during $[t_1, t_2)$. Also, define $D_{P,e,i}(t_1, t_2)$ as the total number of type i packets following P that crosses e during $[t_1, t_2)$. Let $Q_{e,i}(t)$ be the total number of type i packets that is waiting at e's queue at time t and let $S_e(t_1, t_2)$ be the total setup time incurred by e during $[t_1, t_2)$. Finally, let $Y_e(t_1, t_2)$ be the idle time of e during $[t_1, t_2)$.

For the sake of simplicity we denote $A_{P,i}(0, t)$ by $A_{P,i}(t)$ and adopt a similar convention for B, D, S, and Y. When a sub-index is missing for some previously defined quantity, it is to be understood that summation is over the range of the missing index, e.g. $Q_e(t) = \sum_{i\in\mathcal{I}} Q_{e,i}(t)$.

Discrete model equations: Let $\mathcal{R}_e$ denote the set of paths starting with edge e and $\mathcal{R}_{f,e}$ the set of paths where e immediately follows f. Up to time step t the number of type i packets that is injected by the adversary (arrive from some other node in the network, respectively) at e's queue is $\sum_{P\in\mathcal{R}_e} A_{P,i}(t)$ ($\sum_{f\in T(e)}\sum_{P\in\mathcal{R}_{f,e}} D_{P,f,i}(t)$, where $T(e)$ denotes the set of edges incident to the node at the tail of e, respectively). It is easy to see that the dynamics of the adversarial network is fully described by the load and setup constraints (Eqs. (1) and (2) respectively) and the following equations valid for each $i \in \mathcal{I}$ and $e \in E$:

$$B_{e,i}(t) = \sum_{P\in\mathcal{R}_e} A_{P,i}(t) + \sum_{f\in T(e)}\sum_{P\in\mathcal{R}_{f,e}} D_{P,f,i}(t), \tag{8}$$

$$Q_{e,i}(t) = Q_{e,i}(0) + B_{e,i}(t) - D_{e,i}(t), \tag{9}$$

$$S_e(t) + D_e(t) + Y_e(t) = t. \tag{10}$$

Let $\vec{A}(t) = (A_{P,i}(t))_{P,i}$, $\vec{B}(t) = (B_{e,i}(t))_{e,i}$, $\vec{Y}(t) = (Y_e(t))_e$, $\vec{Q}(t) = (Q_{e,i}(t))_{e,i}$, $\vec{D}(t) = (D_{P,e,i}(t))_{P,e,i}$ and $\vec{S}(t) = (S_{e,i}(t))_{e,i}$. The time dependent vector sequence $(\vec{A}, \vec{B}, \vec{D}, \vec{Q}, \vec{S}, \vec{Y})$ is called a *realization* of the discrete model if it satisfies (1)–(2) and (8)–(10).

For multi-type queuing networks we say that a realization is *work conserving* if no edge is idle when its queue is non-empty. Formally,

$$\forall t \in [t_1, t_2),\ Q_e(t) > 0 \Longrightarrow Y_e(t_1) = Y_e(t_2).$$

Moreover, a realization is called *stable* if the number of packets in the system stays bounded over time, i.e. $\sup_t \sum_{e\in E} Q_e(t) < \infty$. Finally, an adversarial multi-type queuing network G is called *universally stable* against adversary A if it is stable for every setup cost Δ and work conserving realization.

Fluid model equations: Fluid models of queuing networks correspond to continuous approximation of queuing networks. In these models one associates to relevant network parameters real-valued functions (time-dependent, continuous, non-decreasing and non-negative). Specifically, for each edge e, path P and type i, we consider functions: $\widehat{A}_{P,i}(t)$, $\widehat{B}_{e,i}(t)$, $\widehat{D}_{P,e,i}(t)$, $\widehat{Q}_{e,i}(t)$, $\widehat{S}_{e,i}(t)$, and $\widehat{Y}_e(t)$. We also define the vectors $\widehat{A}(t)$, $\widehat{B}(t)$, $\widehat{D}(t)$, $\widehat{Q}(t)$, $\widehat{S}(t)$ and $\widehat{Y}(t)$ as in the discrete model.

A continuous mapping from t to $(\widehat{A}(t), \widehat{B}(t), \widehat{D}(t), \widehat{Q}(t), \widehat{S}(t), \widehat{Y}(t))$ is called a *fluid solution* for the adversarial multi-type model if it satisfies the following set of equations: For each $i \in \mathcal{I}$, $P \in \mathcal{R}$, and $e \in E$,

$$\widehat{A}_P(t) \le r \cdot t, \tag{11}$$

$$\widehat{B}_{e,i}(t) = \sum_{P\in\mathcal{R}_e} \widehat{A}_{P,i}(t) + \sum_{f\in T(e)}\sum_{P\in\mathcal{R}_{f,e}} \widehat{D}_{P,f,i}(t), \tag{12}$$

$$\widehat{Q}_{e,i}(t) = \widehat{Q}_{e,i}(0) + \widehat{B}_{e,i}(t) - \widehat{D}_{e,i}(t), \tag{13}$$

$$\widehat{S}_e(t) + \widehat{D}_e(t) + \widehat{Y}_e(t) = t. \tag{14}$$

A fluid solution is called *work conserving* if

$$\forall t \in [t_1, t_2],\ \widehat{Q}_e(t) > 0 \Longrightarrow \widehat{Y}_e(t_1) = \widehat{Y}_e(t_2).$$

A fluid solution $(\widehat{A}(t), \widehat{B}(t), \widehat{D}(t), \widehat{Q}(t), \widehat{S}(t), \widehat{Y}(t))$ is called *stable* if there is some time τ after which the fluid network stays empty, i.e. $\widehat{Q}(t) = 0$ for all $t \ge \tau$. A fluid network is called *globally stable* if there is a time τ such that $\widehat{Q}(t) = 0$ for all $t \ge \tau$ for any fluid solution with initial vector of queue lengths $\widehat{Q}(0)$ for which $\sum_e \widehat{Q}_e(0) = 1$,

This section's main result is an analogue of the one due to Gamarnik [20, Theorem 1] concerning the single-type case. Specifically we prove the following:

Theorem 9. *In the AQT model with setups, if $(G, \Delta, (P, S); A)$ is a stable network system whose associated fluid network is globally stable, then it holds that $(G, \Delta, (P, S); A)$ is stable.*

Proof. Consider a realization $(\vec{A}, \vec{B}, \vec{D}, \vec{Q}, \vec{S}, \vec{Y})$ that satisfies (1)–(2) and (8)–(10). From Gamarnik [20, Proposition 1] we deduce that for a non-decreasing sequence of integer times $t_1, t_2, \ldots, t_k, \ldots$ there exists a sequence of positive integers $k_1, k_2, \ldots, k_n, \ldots$ such that for any non-negative real number t the following quantities converge as n goes to infinity:

$$\widehat{Q}_e(t) = \lim_{n\to\infty} \frac{1}{k_n} Q_e(t_{k_n} + tk_n),$$

$$\widehat{A}_e(t) = \lim_{n\to\infty} \frac{1}{k_n} \left(A_e(t_{k_n} + tk_n) - A_e(t_{k_n})\right),$$

$$\widehat{D}_e(t) = \lim_{n\to\infty} \frac{1}{k_n} \left(D_e(t_{k_n} + tk_n) - D_e(t_{k_n})\right),$$

$$\widehat{B}_e(t) = \lim_{n\to\infty} \frac{1}{k_n} \left(B_e(t_{k_n} + tk_n) - B_e(t_{k_n})\right).$$

Moreover, the limits satisfy (11)–(13). Denote each of these limits by $\widehat{Q}_e(t)$, $\widehat{A}_e(t)$, $\widehat{D}_e(t)$ and $\widehat{B}_e(t)$ respectively. We now show that we can associate with $S_e(\cdot)$ and $Y_e(\cdot)$ fluid limits which together with $\widehat{A}(\cdot)$, $\widehat{B}(\cdot)$, $\widehat{D}(\cdot)$, and $\widehat{Q}(\cdot)$ yield a work conserving fluid solution if $(\vec{A}, \vec{B}, \vec{D}, \vec{Q}, \vec{S}, \vec{Y})$ is a work conserving realization. Indeed, note that from (10) we get that for any non-decreasing integer sequence $t_1, t_2, \ldots$ and positive rational number q,

$$\frac{1}{k}\left(S_e(t_k+qk) - S_e(t_k)\right) \leq \frac{qk}{k} = q.$$

Thus, again by [20, Proposition 1], we conclude that the fluid limit $\widehat{S}_e(\cdot)$ exists. A similar argument shows that the fluid limit $\widehat{Y}_e(\cdot)$ exists. It is easy to check that $\widehat{A}$, $\widehat{B}$, $\widehat{D}$, $\widehat{Q}$, $\widehat{S}$, and $\widehat{Y}$ must satisfy (11)–(14), i.e. $(\widehat{A}, \widehat{B}, \widehat{D}, \widehat{Q}, \widehat{S}, \widehat{Y})$ is a fluid solution of the fluid model.

We now want to show that the fluid solution is work conserving. For the sake of contradiction, suppose that $\widehat{Q}_e(t) > 0$ for all $t \in [t_1, t_2]$. Since $\widehat{Q}_e(\cdot)$ is continuous and $[t_1, t_2]$ is a compact set, there is a $\gamma > 0$ such that $\widehat{Q}_e(t) > \gamma$ for all $t \in [t_1, t_2]$. But $\widehat{Q}_e(t) = \lim_{n\to\infty} Q_e(t_{k_n}+tk_n)/k_n$, hence $Q_e(t_{k_n}+tk_n) > \gamma k_n$ for sufficiently large n. This implies, since the discrete realization is work conserving, that $Y_e(t_{k_n}+t_1k_n) = Y_e(t_{k_n}+t_2k_n)$ for sufficiently large n, hence $\widehat{Y}_e(t_1) = \widehat{Y}_e(t_2)$.

Because of the global stability of G's associated fluid network, we have that there is a time τ such that $\widehat{Q}(t) = 0$ for all $t \geq \tau$ for any fluid solution with initial vector of queue lengths $\widehat{Q}(0)$ for which $\sum_e \widehat{Q}_e(0) = 1$. However, this is impossible if $(\vec{A}, \vec{B}, \vec{D}, \vec{Q}, \vec{S}, \vec{Y})$ is an unstable realization (for details, see [20, Lemma 3]). □

6. Final comments

It is worth noting that much of what was known for the standard AQT model seems to hold also in the multi-type scenario. Specifically we have that:

(1) Acyclic graphs are universally stable in the single-type ([9, Theorem 1]) and multi-type setting (Theorem 4).
(2) Directed rings are universally stable in the single-type ([5, Theorem 3.7]) and multi-type setting (Theorem 5).

The above two claims are subsumed by the following consequence of [2, Theorem 8] and Theorem 7:

(3) A digraph is universally stable in the standard AQT model if and only if it is universally stable in the AQT model with setups.

Perhaps the strongest contrast between the AQT models with and without setup arises when one compares universal stability of scheduling policies in the single-type scenario with universal stability of scheduling protocols in the multi-type scenario:

(4) For the standard AQT model, there are universally stable greedy scheduling policies against bounded rate adversaries (e.g. SHORTEST-IN-SYSTEM, FARTHEST-TO-GO [5, Table 1]). In contrast, for the AQT model with setups no greedy scheduling protocol (P, S) is universally stable against bounded rate adversaries if S is shift-oblivious (Theorem 8).

Finally, we note that from a fluid model perspective, stability issues in both single-type and multi-type adversarial networks can be addressed in a similar way (contrast for example Theorem 9 and [20, Theorem 1]).

Arguably the most interesting issue that this work leaves open is finding conditions for which, in the AQT with setups model, there is a canonical adaptation of a packet scheduling protocol P and a way of choosing a switching policy S such that a routing network G will remain stable against an (r, b) adversary A, but in the multi-class scenario where setup costs equal Δ.

Acknowledgments

We are grateful to David Gamarnik for many valuable discussions and for referring us to the work of Dai and Jennings [17]. We also wish to thank Ottis Jennings and Jim Day for sharing their knowledge about fluid models, and Ashish Goel for pointing us to relevant references.

The first author gratefully acknowledges the support of CONICYT via FONDAP in Applied Mathematics, FONDECYT 1050710 and Anillo en Redes ACT08. The second author gratefully acknowledges the support of CONICYT via FONDAP in Applied Mathematics. The third author gratefully acknowledges the support of Facultad de Ciencias Físicas y Matemáticas via a postgraduate fellowship, Proyecto Mecesup UCH0009, CONICYT via Anillo en Redes ACT08 and FONDAP in Applied Mathematics.

References

[1] C. Àlvarez, M. Blesa, J. Díaz, A. Fernández, M. Serna, Adversarial models for priority based networks, Networks 45 (1) (2005) 23–35.
[2] C. Àlvarez, M. Blesa, M. Serna, A characterization of universal stability in the adversarial queueing model, SIAM J. Comput. 34 (1) (2004) 41–66.

[3] M. Andrews, Instability of FIFO in the permanent sessions model at arbitrarily small network loads, in: Proc. of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 219–228.
[4] M. Andrews, Instability of FIFO in session-oriented networks, J. Algorithms 50 (2) (2004) 232–245.
[5] M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, T. Leighton, Z. Liu, Universal stability results and performance bounds for greedy contention resolution protocols, J. ACM (2001) 39–69.
[6] E. Anshelevich, D. Kempe, J. Kleinberg, Stability of load balancing algorithms in dynamic adversarial systems, in: Proc. of the 34th ACM Symposium on Theory of Computing, 2002, pp. 399–406.
[7] B. Awerbuch, P. Berenbrink, A. Brinkmann, C. Scheideler, Simple routing strategies for adversarial systems, in: Proc. of the 42nd IEEE Symposium on Foundations of Computer Science, 2001, pp. 158–167.
[8] R. Bhattacharjee, A. Goel, Z. Lotker, Instability of FIFO at arbitrarily low rates in the adversarial queueing model, SIAM J. Comput. (2005) 318–332.
[9] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, D. Williamson, Adversarial queueing theory, J. ACM (2001) 13–38.
[10] M. Bramson, Instability of FIFO queuing networks, Ann. Appl. Probab. 4 (1994) 414–431.
[11] M. Bramson, Instability of FIFO queuing networks with quick service times, Ann. Appl. Probab. 4 (1994) 693–718.
[12] H. Chen, Fluid approximations and stability of multi class queueing networks: Work conserving disciplines, Ann. Appl. Probab. (1995) 636–665.
[13] V. Cholvi, J. Echagüe, Stability of FIFO networks under adversarial models: State of the art, Comput. Netw. 51 (15) (2007) 4460–4474.
[14] R. Cruz, A calculus for network delay. Part I, IEEE Trans. Inf. Theory (1991) 114–131.
[15] R. Cruz, A calculus for network delay. Part II, IEEE Trans. Inf. Theory (1991) 132–141.
[16] J. Dai, On positive Harris recurrence of multiclass queueing networks: A unified approach via fluid limit models, Ann. Appl. Probab. (1995) 49–77.
[17] J. Dai, O. Jennings, Stabilizing queueing networks with setups, Math. Oper. Res. (2004) 891–922.
[18] J. Dai, S. Meyn, Stability and convergence of moments for networks and their fluid models, IEEE Trans. Automat. Control (1995) 1889–1904.
[19] J. Díaz, D. Koukopoulos, S. Nikoletseas, M. Serna, P. Spirakis, D. Thilikos, Stability and non–stability of the FIFO protocol, in: Proc. of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures, 2001, pp. 48–52.
[20] D. Gamarnik, Stability of adversarial queues via fluid model, in: Proc. of the 39th Annual Symposium on Foundations of Computer Science, 1998, pp. 60–70.
[21] S.B. Gershwin, Stochastic scheduling and set-ups in manufacturing systems, Internat. J. Production Res. 33 (1995) 1849–1870.
[22] A. Goel, Stability of networks and protocols in the adversarial queueing model for packet routing, Networks (2001) 219–224.
[23] O. Jennings, Multiclass queueing networks with setup delays: Stability analysis and heavy traffic approximation, Ph.D. Thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, 2000.
[24] F. Kelly, Reversibility and Stochastic Systems, John Wiley & Sons, 1979.
[25] M. Kiwi, A. Russell, The chilean highway problem, Theoret. Comput. Sci. (2004) 329–342.
[26] L. Kleinrock, Queuing Systems, Volume 1: Theory, John Wiley & Sons, 1975.
[27] D. Koukopoulos, S. Nikoletseas, P. Spirakis, The range of stability for heterogeneous and FIFO queueing networks, Electronic Colloquium on Computational Complexity, TR-01-099, 2001.
[28] P. Kumar, T. Seidman, Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems, IEEE Trans. Automat. Control (1990) 289–298.
[29] Z. Lotker, B. Patt-Shamir, A. Rosén, New stability results for adversarial queuing, SIAM J. Comput. 33 (2) (2004) 286–303.
[30] A. Rybko, A. Stolyar, Ergodicity of stochastic processes describing the operation of open queueing networks, Probl. Inf. Transm. 28 (1996) 366–375.
[31] L. Shunong, A distributed concurrency control protocol considering read-only transactions, Technical Report, Department of Computer Science, U. of Illinois Urbana-Champaign, Urbana, Illinois, 1991.