STABILITY OF LOAD BALANCING ALGORITHMS IN DYNAMIC ADVERSARIAL SYSTEMS[*]

ELLIOT ANSHELEVICH[†], DAVID KEMPE[‡], AND JON KLEINBERG [§]

Abstract. In the dynamic load balancing problem, we seek to keep the job load roughly evenly distributed among the processors of a given network. The arrival and departure of jobs is modeled by an adversary restricted in its power. Muthukrishnan and Rajaraman (1998) gave a clean characterization of a restriction on the adversary that can be considered the natural analogue of a cut condition. They proved that a simple local balancing algorithm proposed by Aiello et. al. (1993) is stable against such an adversary if the insertion rate is restricted to a $(1-\varepsilon)$ fraction of the cut size. They left as an open question whether the algorithm is stable at rate 1.

In this paper, we resolve this question positively, by proving stability of the local algorithm at rate 1. Our proof techniques are very different from the ones used by Muthukrishnan and Rajaraman, and yield a simpler proof and tighter bounds on the difference in loads.

In addition, we introduce a multi-commodity version of this load balancing model, and show how to extend the result to the case of balancing two different kinds of loads at once (obtaining as a corollary a new proof of the 2-commodity Max-Flow Min-Cut Theorem). We also show how to apply the proof techniques to the problem of routing packets in adversarial systems. Awerbuch et. al. (2001) showed that the same load balancing algorithm is stable against an adversary inserting packets at rate 1 with a single destination, in dynamically changing networks. Our techniques give a much simpler proof for a different model of adversarially changing networks.

Key words. Adversarial load balancing, packet routing, multi-commodity flow

AMS subject classifications. 68W15,68W40

1. Introduction.

Load Balancing. In a distributed network of computing hosts, the performance of the system can depend crucially on dividing up work effectively across the participating nodes. This type of *load balancing problem* has been studied in many different models, centered around the idea that an algorithm should avoid creating "hot spots" that degrade system performance [24].

We consider a basic model of load balancing in a distributed network, which has formed the basis of a number of earlier studies [1, 16, 5, 21, 22]. A network of identical processors is represented by an undirected graph $G = (V, E)$. There are a number of *jobs* to be processed in the system, abstractly represented by unit-size *tokens*. Time progresses in discrete steps called *rounds*; in a given round, each token is held by one of the nodes, which is viewed as processing the associated job, and the *load* on a node is defined to be the number of tokens it holds. The goal is to *balance* the loads, so that no single node has too many tokens; this can be accomplished by transmitting tokens between neighboring nodes of the graph, at a rate of one token per edge per round. We are particularly interested in *local* algorithms for this problem: rather than using a centralized approach to coordinate the movement of tokens, each node will simply

[*]A preliminary version of this paper appeared in *Proc. 34th Annual ACM Symposium on Theory of Computing*, 2002.

[†]Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180. Email: `eanshel@cs.rpi.edu`. This research was supported in part by an NSF graduate fellowship.

[‡]Department of Computer Science, University of Southern California, Los Angeles, CA 90089. Email: `dkempe@usc.edu`. This resesarch was supported by an NSF graduate fellowship.

[§]Department of Computer Science, Cornell University, Ithaca, NY 14853. Email: `kleinber@cs.cornell.edu`. Research supported in part by a David and Lucile Packard Foundation Fellowship, an ONR Young Investigator Award, and NSF Faculty Early Career Development Award CCR-9701399.

compare its load to those of its neighbors, and decide whether to move a token across an edge based on this information.

This model is clearly very simple in a number of respects, particularly in the uniformity of the processors (nodes) and jobs (tokens), and the fact that any job can be executed on any processor. More subtly, it is not even clear in all settings that balancing load evenly is the optimal strategy in a distributed network of processors (see, e.g., [12]). At the same time, however, the model cleanly captures the basic constraints imposed by an underlying interconnection topology in the process of distributing jobs through a network, as evidenced by the results of previous analysis [1, 16, 5, 21, 22]; the simplicity of the model allows one to reason very clearly about the effect of these constraints.

Early work on the model focused on the *static* version of the problem: each node is given a set of tokens initially, and nodes must exchange tokens as rapidly as possible so that they all end up with approximately the same number [1, 16, 5, 22]. However, load balancing is a natural setting in which to study algorithms designed to run indefinitely; jobs (tokens) may arrive and depart from the network, and at all times, the algorithm must maintain an approximately uniform load across nodes. This is a type of *stability* condition: no load should diverge arbitrarily from the average as time progresses. For a number of different models, such *dynamic* algorithms have been studied in a probabilistic framework, where one assumes an underlying randomized process that generates job arrivals and departures; see, e.g., [13, 20] and the references therein.

An Adversarial Model. Motivated by work in the related area of packet routing [7, 8, 11, 4], Muthukrishnan and Rajaraman proposed an *adversarial* framework for studying dynamic load balancing in the token-based model we have been discussing [21]. Rather than considering a probabilistic process that generates tokens, they posit an *adversary* that is allowed at the beginning of each round to introduce tokens at some nodes (corresponding to new jobs) and remove tokens from others (corresponding to jobs that have finished). Subsequently, an algorithm is allowed to move tokens across edges as above so as to try to maintain balanced loads. This alternation of moves by the adversary and algorithm continues for an arbitrary number of rounds. Note that by allowing the adversary to control the removal of tokens as well as their arrival, one is modeling a worst-case assumption that jobs may have arbitrary duration, and the algorithm does not know how much processing time a job has remaining until the moment it ends.

If we let a_t denote the average number of tokens per node in the system at the beginning of round t, and $h_t(v)$ denote the number of tokens at node v (the *height* of v) at round t, then the goal of a dynamic load balancing algorithm in this model is to keep $h_t(v)$ close to a_t for all nodes v and rounds t. Formally, we say that an algorithm is *stable* against a given adversary if there is a constant B such that $|h_t(v) - a_t| \leq B$ for all nodes v and rounds t. Note that stability in this context imposes a bound on deviation from the average; it is not required that the actual number of tokens in the system remain bounded.

As in the case of packet routing [11, 4], one needs to find a suitable restriction on the adversary: an arbitrarily powerful adversary could flood a particular set of nodes $S \subseteq V$ with tokens much faster than these nodes can spread the tokens out to the rest of the graph, and thereby prevent any algorithm from maintaining stability. This consideration motivates the following natural definition of an adversary [21] with *rate* r. For a set $S \subseteq V$, let $e(S)$ denote the set of edges with exactly one end in S, and $\delta_t(S)$ the net increase in tokens in set S due to the addition and removal of jobs in round t (note that $\delta_t(S)$ could be negative). If the heights of nodes in S were to change precisely according to average, then the net change in tokens in S would be $|S| \cdot (a_{t+1} - a_t)$. One wants the difference between these two quantities to be "accounted for" by the edges in $e(S)$. We say that the adversary has *rate r* if for all $S \subseteq V$, one has

$$|\delta_t(S) - |S|(a_{t+1} - a_t)| \leq r \cdot |e(S)|, \tag{1.1}$$

For rate $r > 1$, there are adversaries against which no algorithm (whether online or offline) can be stable. Muthukrishnan and Rajaraman gave a local-control algorithm that is stable against all adversaries of rate r, for every $r < 1$. As an open question, they asked whether there exists a local-control algorithm that is stable against all adversaries of rate 1.

The present work. We begin by providing a local-control load balancing algorithm that is stable against every adversary of rate 1, thereby resolving the open question of Muthukrishnan and Rajaraman. In fact, we show that the following simple rule has this stability property, for every value of the parameter θ:

At any round t, if the number of tokens on node u exceeds the number of tokens on its neighbor v by at least θ, then u moves a token to v.

This type of algorithm was considered in earlier work on the static model by Aiello et al. [1], as well as by many of the subsequent papers. Setting $\theta = 2\Delta+1$, where Δ is the maximum node degree in G, yields the specific algorithm studied by Muthukrishnan and Rajaraman.

Beyond simply showing the stability of local algorithms at the *critical rate* $r = 1$, our analysis is based on a new proof technique in which a potential function bound is maintained not only for the entire node set V, but for every subset of V. Compared to [21], we obtain significantly improved bounds on the deviation from the average, and a simpler proof. Specifically, we show that the maximum possible deviation from the average is $O(\Delta n)$, where n is the number of nodes of G, and this is asymptotically optimal in the worst case; the analysis in [21] had established a bound of $O(\Delta^2 n^{2.5}(1-r)^{-1})$ when $r < 1$. Our analysis also shows stability in a more general model where edges of G can appear and disappear over time.

Following this, we introduce a *multi-commodity* version of this load balancing model. We consider a system in which there are k distinct types of jobs. The jobs of one given type induce the same load on each processor; but the different types of jobs place different resource requirements on the nodes, and so we require the load balancing condition to apply to each type separately. Formally, we have the same adversarial model as before with a network G and a collection of tokens; but now the tokens are partitioned into k *commodities* and the stability requirement must hold when the tokens of each commodity are considered separately. In a single round, at most one token in total can be sent across any one edge. (This is in keeping with the standard multi-commodity notion that constraints at nodes must be satisfied by each commodity separately, while shared edge capacities must be respected by the commodities cumulatively.)

We show that the natural rate condition on adversaries — essentially obtained by summing Equation (1.1) over the commodities — can be related in a precise sense to the cut condition for standard multi-commodity network flow. As a result, applying well-known results on the cut condition [17, 18, 19], we find that for every $k > 2$, there is a k-commodity adversary of rate $r_k \leq 1$ against which no load balancing algorithm can be stable.

For $k = 2$, however, the cut condition does not pose an obstacle to having algorithms that are stable all the way up to rate 1. Indeed, we are able to generalize our first result to show that for 2-commodity load balancing, there is a simple local-control algorithm that is stable against every adversary of rate 1. We also use the relationship between adversaries and cut conditions to provide a new proof of Hu's Max-Flow Min-Cut Theorem for 2-commodity flow [17]. While our proof is not necessarily shorter than other proofs discovered subsequent to Hu's [19, 23], it is arguably more elementary: it does not require linear programming duality (as in [19]) or even the traditional Max-Flow Min-Cut Theorem for single-commodity flow (as in [23]).

Finally, we further develop the connection between dynamic load balancing and network flows by extending our analysis to packet routing in the adversarial model considered by Aiello et al. [2] and Gamarnik [15]. We give an adaptive routing algorithm that is stable against adversaries of rate 1 in the case where packets can be injected at multiple sources but are destined for a single sink; our algorithm is stable in a dynamic network model where edges can appear and disappear. A stable algorithm for this version of the problem was previously given in a recent paper of Awerbuch et al. [6], using a different, but essentially more general, notion of a dynamic network; our proof, a direct adaption of the analysis of our single-commodity load balancing algorithm, is considerably shorter and simpler.

Recent Progress. Since the original publication of this work, several papers have studied a model wherein nodes can exchange arbitrarily many jobs in one step, and each node with non-empty load executes one job in each time step. This type of load-balancing algorithm (often called the diffusion algorithm) has been very well studied, and is known to have many nice properties, although its behavior in the presence of an adversary was unknown until recently. In this model, the cut condition becomes trivial; instead, the adversary is allowed to insert at most n jobs in each time step, and stability is defined in terms of an upper bound on the load of all nodes.

Berenbrink, Friedetzky, and Golberg [9] show that the work-stealing algorithm, in which only processors with empty queues request jobs from others, is stable against adversaries of rate strictly below 1. They assume that processors can request jobs from any other processor. Anagnostopoulos, Kirsch, and Upfal [3] show the same type of stability for a local protocol that makes nodes equalize load with their neighbors. Most recently, Berenbrink, Friedetzky, and Martin [10] proved stability against rate-1 adversaries for a protocol in which nodes exchange load with all their neighbors. Both the protocol and the analysis are very similar to the ones in the present work.

2. Single-commodity load balancing.

In this section, we will study the load balancing problem for a single commodity, and in particular prove that the natural balancing algorithm is stable at rate 1, thus settling an open question from [21]. We first define precisely the model and the algorithm, and introduce the necessary notation.

2.1. Model and Algorithm.

The network is represented by a connected undirected graph $G = (V, E)$ with $n = |V|$ nodes. In each round, the adversary first adds or removes tokens (this is called the *Adversary step*). Subsequently, in the *Redistribution step*, up to one token can be moved along each edge $e \in E$ by the algorithm (or up to c_e tokens in the case of networks with edge capacities).

The adversary is limited by the following *cut condition*: For a subset $S \subseteq V$ of nodes, let $\delta_t(S)$, a_t, and $e(S)$ be defined as above. Then, the insertion and removal of tokens by the adversary during round t has to satisfy

$$|\delta_t(S) - |S|(a_{t+1} - a_t)| \leq |e(S)|. \tag{2.1}$$

Nodes have queues associated with them, in which they store their tokens. The *height* $h_t(v)$ of a node v is the number of tokens in v's queue at the beginning of round t. The *imbalance* is $b_t(v) = h_t(v) - a_t$, i.e., the number of excess (or missing) tokens at node v with respect to the average over the entire network. $\overline{h}_t(v)$ and $\overline{b}_t(v)$ denote the same quantities after the Adversary step of round t.

It is the decision of the algorithm along which edges to send tokens. The goal of any balancing algorithm is to keep the imbalance bounded for all nodes. If an algorithm ensures that there is an absolute bound B such that $|b_t(v)| \leq B$ for all nodes v and all times t, we call the algorithm *stable.* We will show that the following very simple family of local-control algorithms is stable against every adversary respecting the cut condition. It has a *threshold parameter* $\theta \geq 1$, which determines how aggressively the algorithm balances.

```
Algorithm SCLB$_\theta$
At each time $t$, for each edge $e = (u, v)$:
    If $h_t(u) \geq h_t(v) + \theta$, then send a token from $u$ to $v$.
    If $h_t(v) \geq h_t(u) + \theta$, then send a token from $v$ to $u$.
```

This algorithm does not specify whether tokens are sent along an edge (u, v) when $|h_t(u) - h_t(v)| < \theta$. All of our subsequent statements will remain true independently of what the algorithm does in this case. (Notice that this algorithm only requires local information, and can therefore be executed in a distributed fashion in a network.)

2.2. Stability of the algorithm. Our main theorem in this section is that the algorithm $\texttt{SCLB}_\theta$ is stable against any adversary respecting the cut condition of Inequality 2.1. We allow for tokens to be in the system at time 1, and let $H := \max_{v \in V} h_1(v)$.

THEOREM 2.1. *For any adversary respecting the cut condition, and any $\theta \geq 1$, the algorithm $\texttt{SCLB}_\theta$ is stable, i.e., there is a constant B (depending on H, θ and G), such that $|b_t(v)| \leq B$ for all nodes v at all times t.*

The intuition behind our proof is based on the (incorrect) observation that the algorithm seems to ensure that the height difference between adjacent nodes cannot grow beyond θ. Hence, the largest difference between the heights of any two nodes should be achieved when G is a simple path, and the two nodes are the endpoints of the path — having a height difference of about $n\theta$.

It is, however, easy to see that the height difference between two adjacent nodes can become more than θ, because the adversary can "rearrange" the heights within sets to a certain extent. Imagine, for instance, a long path with the adversary adding tokens to the first node of the path, until the first node has height $n\theta$, and the height of each successive node decreases by θ. Then it is possible to rearrange the tokens on the nodes previously having heights $n\theta$, $(n-1)\theta$ and $(n-2)\theta$ so that they each have $(n-1)\theta$ tokens, which would result in the height difference of 2θ between the third and fourth node of the path, since the height of the fourth node remains $(n-3)\theta$. The adversary can do this by adding two tokens to the node with height $(n-2)\theta$ and subtracting one from the node with height $n\theta$ for θ successive rounds. During each Redistribution step of this process, a token will move from the third to the fourth node, but these tokens will continue traveling downhill along the path, so that the height of the fourth node will never grow to be much larger than $(n-3)\theta$. This process obeys the cut condition, but afterwards, there are two adjacent nodes with almost 2θ height difference. Fortunately, although there are now more nodes with large heights, the adversary had to pay for this rearrangement by making the highest queue smaller. In an amortized sense, the situation has not become worse.

These observations suggest maintaining height bounds for each subset of the nodes, and showing that these bounds form an invariant. For convenience, we will write $b_t(S) = \sum_{v \in S} b_t(v)$ for any set $S \subseteq V$ of vertices (and similarly for other quantities like $h_t(S)$ and $\delta_t(S)$). With Δ denoting the maximum degree of any vertex, we write $\gamma = 2\Delta + \theta$. The key invariant is the following:

$$|b_t(S)| \le \sum_{j=n-|S|+1}^{n} (H + \gamma \cdot j) \quad \text{for all } S \subseteq V. \tag{2.2}$$

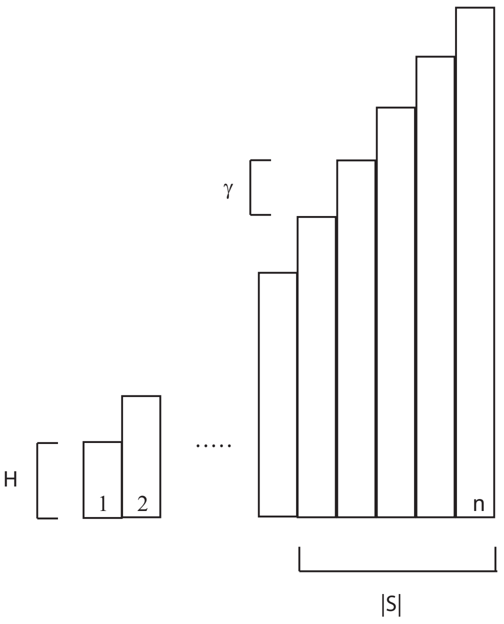

FIG. 2.1. *An illustration of Invariant 2.2*

Figure 2.1 illustrates the upper bound of this invariant pictorially as the sum of the column heights of the right-most $|S|$ columns.

Below, we prove Lemma 2.2, showing that Inequality (2.2) is indeed an invariant over time for the algorithm $\texttt{SCLB}_\theta$, against any adversary respecting the cut condition.

LEMMA 2.2. *If the adversary respects the cut condition, and the invariant (2.2) holds at the beginning of round t, then it holds at the beginning of round $t+1$.*

Using this lemma, the proof of Theorem 2.1 is straightforward.

Proof of Theorem 2.1. We prove by induction that (2.2) holds at every time t. At time 1, $h_1(v) \le H$ for all v by definition, so

$$|b_1(S)| \quad \le \quad \textstyle\sum_{v \in S} H \quad \le \quad \sum_{j=n-|S|+1}^{n} (H + \gamma \cdot j)$$

for all sets $S \subseteq V$. The induction step from t to $t+1$ follows from Lemma 2.2, and we can apply the resulting guarantee to the singleton sets $\{v\}$, yielding a bound of $B = H + \gamma \cdot n$. $\square$

Proof of Lemma 2.2. The proof is by contradiction. Assume that the invariant (2.2) holds at the beginning of round t, but not at the beginning of round $t+1$. Let S be a set maximizing

$$\Phi(S) := |b_{t+1}(S)| - \sum_{j=n-|S|+1}^{n} (H + \gamma \cdot j).$$

If several sets achieve the maximum value, let S have minimal size among all these sets. First off, notice that the choice of S guarantees that either all $u \in S$ have positive $b_{t+1}(u)$, or they all have negative $b_{t+1}(u)$, and hence $|b_{t+1}(S)| = \sum_{u \in S} |b_{t+1}(u)|$.

Since (2.2) was assumed to hold at the beginning of round t, and fails at the beginning of round $t+1$, we know that $|b_{t+1}(S)| > |b_t(S)|$. How can the values $h_t(u)$ for nodes $u \in S$ change?

Adversary step: Substituting the definitions of $\overline{b}$ and b, we obtain that for any set S,

$$\begin{aligned}
|\overline{b}_t(S)| &= |\overline{h}_t(S) - |S| \cdot a_{t+1}| \\
&= |h_t(S) + \delta_t(S) - |S| \cdot a_{t+1}| \\
&= |b_t(S) + \delta_t(S) - |S| \cdot (a_{t+1} - a_t)| \\
&\le |b_t(S)| + |\delta_t(S) - |S| \cdot (a_{t+1} - a_t)| \\
&\le |b_t(S)| + |e(S)|.
\end{aligned}$$

The two inequalities hold because of the Triangle Inequality and the cut condition on the adversary.

Redistribution step: Fix an edge $e = (u, v)$ with $u \in S$ and $v \notin S$. Because S maximizes Φ and has minimal size, u has imbalance $|b_{t+1}(u)| > H + \gamma \cdot (n - |S| + 1)$, since otherwise $\Phi(S - u) \ge \Phi(S)$. In particular, $|b_{t+1}(u)| > \gamma$.

Because v was not included in S, its imbalance $b_{t+1}(v)$ must either have sign opposite to the sign of $b_{t+1}(S)$, or have absolute value $|b_{t+1}(v)| \le H + \gamma \cdot (n - |S|)$. In either case, $|b_{t+1}(u) - b_{t+1}(v)| > \gamma$, and simply substituting the definition of γ, we also obtain $|h_{t+1}(u) - h_{t+1}(v)| > 2\Delta + \theta$. During the Redistribution step of round t, at most Δ tokens can have moved to or from nodes u and v, so their heights can have changed by at most Δ each, and therefore their previous height difference is at least $|\overline{h}_t(u) - \overline{h}_t(v)| > \theta$. Figure 2.2 illustrates the gap of at least γ between the queue heights of nodes in S and outside S.

If $b_{t+1}(u) \ge 0$, then $\overline{h}_t(u) - \overline{h}_t(v) > \theta$, so the algorithm $\texttt{SCLB}_\theta$ moves a token from u to v along e, and no tokens from v to u, thereby decreasing $\overline{b}_t(u)$ by 1. On the other hand, if $b_{t+1}(u) < 0$, then $\overline{h}_t(v) - \overline{h}_t(u) > \theta$, and a token must be moved from v to u along e, increasing the (negative) imbalance $\overline{b}_t(u)$ by 1. Because $|b_{t+1}(u)| > \gamma = \theta + 2\Delta$, and therefore $|\overline{b}_t(u)| > \theta + \Delta$, the sign of the imbalance does not change during the Redistribution step, even if Δ tokens were moved to or from u, and hence, $|\overline{b}_t(u)|$ decreased by 1 as a result of edge e.

This holds for every edge $e \in e(S)$, and using the fact that the average a does not change during the Redistribution step, we obtain that

$$\begin{aligned}
|b_{t+1}(S)| &= \textstyle\sum_{u \in S} |b_{t+1}(u)| \\
&\le (\textstyle\sum_{u \in S} |\overline{b}_t(u)|) - |e(S)| \\
&= |\overline{b}_t(S)| - |e(S)|.
\end{aligned}$$

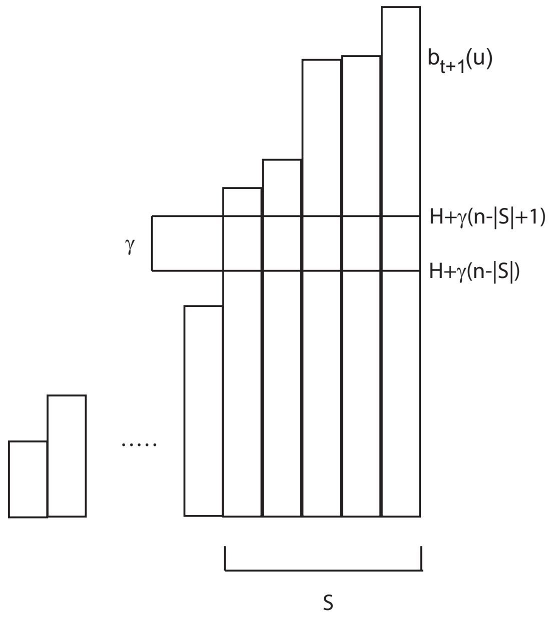

FIG. 2.2. *The gap of γ between the queue heights of nodes in S and outside S in the case that $b_{t+1}(S)$ is positive.*

Putting the arguments for the two steps together, we obtain that $|b_{t+1}(S)| \leq |\overline{b}_t(S)| - |e(S)| \leq |b_t(S)|$. This contradicts our assumption that $|b_{t+1}(S)| > |b_t(S)|$, and thus completes the proof. □

Notice that our bound $B = H + \gamma \cdot n$ is asymptotically tight. To see this, consider a simple path of length n. It is certainly legal for the adversary to insert one token at node n in every round, and never remove tokens. After about $\theta \cdot \frac{n^2}{2}$ rounds, each node k will contain about $k\theta$ tokens, and hence the imbalance of node n is about $\theta \cdot \frac{n}{2}$.

2.3. Capacities, dynamic networks, and time windows. The result of Theorem 2.1 can be easily extended to the case that the edges have (integer) capacities c_e associated with them, and up to c_e tokens can be sent along e in every round. We assume that whenever the algorithm decides to send tokens from u to v along e, it sends as many as possible, i.e., bounded only by the capacity c_e and the number of tokens at u. The cut condition now requires that the imbalance created by the adversary be restricted by the total capacity of the cut. Showing that the above algorithm is still stable in this capacitated version is easy given Theorem 2.1.

COROLLARY 2.3. *In the above capacitated scenario, for any adversary respecting the cut condition, and any $\theta \geq 1$, the algorithm $\texttt{SCLB}_\theta$ is stable, i.e., there is a constant B (depending on H, θ, G, and the maximum edge capacity $\max_e c_e$), such that $|b_t(v)| \leq B$ for all nodes v at all times t.*

Proof. Replace each edge with capacity c_e by c_e parallel edges of capacity 1. By Theorem 2.1, the algorithm is stable on this new graph, and therefore, it is also stable on the capacitated graph. Notice that the constant γ and hence the bound B now depend on the maximum capacity, and so instead of $B = H + 2\Delta n + \theta n$, the new bound becomes $B = H + 2\Delta n \cdot \max_e c_e + \theta n$, where Δ is the maximum degree of the graph. □

Dynamic networks. Another easy extension concerns dynamically changing networks. That is, the set of available edges may change over time, and we assume that it is also controlled by the adversary. For each time t, we have a set E_t of available edges. The cut condition on the adversary must be satisfied at the specific time when the imbalance is created, i.e., $|\delta_t(S) - |S|(a_{t+1} - a_t)| \leq |e_t(S)|$. Here, $e_t(S)$ are the edges from E_t that have exactly one endpoint in S.

COROLLARY 2.4. *In the above dynamically changing network, for any adversary respecting the cut condition, and any $\theta \geq 1$, the algorithm $\mathtt{SCLB}_\theta$ is stable, i.e., there is a constant B (depending on H, θ, and G), such that $|b_t(v)| \leq B$ for all nodes v at all times t.*

Proof. By syntactically replacing all terms $e(S)$ with $e_t(S)$ in the proof of Lemma 2.2, we obtain a proof for the model of dynamically changing networks, with exactly the same bound B. □

Time windows. An extension often considered in the contexts of load balancing or packet routing is to relax the restriction on the adversary by allowing it to violate the cut condition for a certain time, provided that it hold "in the long run". Specifically, a *window size* W is specified, and it is required that for any set S and any time window $[t, t+W)$, the imbalance created on set S over that time window be bounded by the total capacity, i.e., $|(\sum_{r=t}^{t+W-1} \delta_r(S)) - |S|(a_{t+W} - a_t)| \leq W \cdot |e(S)|$.

COROLLARY 2.5. *For any adversary respecting the cut condition on average over a window size W, and any $\theta \geq 1$, the algorithm $\mathtt{SCLB}_\theta$ is stable, i.e., there is a constant B (depending on H, θ, W, and G), such that $|b_t(v)| \leq B$ for all nodes v at all times t.*

Proof. It is not difficult to see that by allowing B to depend on W, we can also extend the stability result to this model — once the imbalance grows too large on a set S, all edges $e \in e(S)$ will be moving tokens so as to reduce the imbalance for every single round of an entire window, so that the imbalance cannot grow further. □

3. Multi-commodity load balancing.

In the previous section, we considered the problem of balancing loads on processors where the loads were interchangeable. However, we are also interested in the case of different kinds of loads that are to be balanced simultaneously. For instance, think of jobs that have an emphasis on different resources of the machine they are running on. Balancing the different classes of jobs independently could be desirable in order to avoid processing time becoming a bottleneck on one machine, and memory size an issue on another.

In the general multi-commodity load balancing problem, we have k different kinds of jobs (or tokens), which are stored in separate queues at the nodes. Our goal is to ensure an absolute bound on the deviation of any queue height from the average queue height for that commodity. Each round t is divided into the same two steps as before, the Adversary step and the Redistribution step.

In analogy to the single-commodity case, we use the following notation: For a node v and commodity i, let $h_t^{(i)}(v)$ be the number of tokens of commodity i on node v at the beginning of round t. Similarly, $a_t^{(i)}$, $b_t^{(i)}(v)$, $\delta_t^{(i)}(v)$, $\overline{h}_t^{(i)}(v)$, and $\overline{b}_t^{(i)}(v)$ are all defined for commodity i exactly as their single-commodity equivalents.

The algorithm will now not only have to choose when to send a token across an edge, but also which of several available (and conflicting) kinds of tokens to send. Our class of algorithms is practically identical to the one from [2] and [6], and can be formalized as follows:

```
Algorithm MCLB$_\theta$
```

```
At each time $t$, for each edge $e = (u, v)$:
    Choose $i$ to maximize $|h_t^{(i)}(u) - h_t^{(i)}(v)|$.
    If $h_t^{(i)}(u) \geq h_t^{(i)}(v) + \theta$, then send a token of commodity $i$ from $u$ to $v$.
    If $h_t^{(i)}(v) \geq h_t^{(i)}(u) + \theta$, then send a token of commodity $i$ from $v$ to $u$.
```

In the case of a single commodity, this algorithm specializes to $\texttt{SCLB}_\theta$.

The natural analogue of the cut condition for a single commodity is to require that the adversary satisfy

$$\sum_i |\delta_t^{(i)}(S) - |S|(a_{t+1}^{(i)} - a_t^{(i)})| \leq |e(S)| \tag{3.1}$$

for all node sets $S \subseteq V$ and times t. This would require that the total imbalance for set S created by the adversary could be "balanced" along edges leaving S.

Unfortunately, Inequality (3.1) is too weak a restriction — it allows the adversary to create patterns of addition and removal that cannot be balanced by any algorithm, whether offline or online. At the end of this section, we show how to use a reduction from the multi-commodity flow problem to create such an adversary with $k \geq 3$ commodities.

For the special case $k = 2$, however, the Max-Flow Min-Cut Theorem still holds, and in fact, we can show that the cut condition is sufficient to ensure that algorithm $\texttt{MCLB}_\theta$ is stable.

3.1. Stability for k=2. We let $H = \max_{v \in V}\{h_1^{(1)}(v) + h_1^{(2)}(v)\}$ be the maximum height of the queues at any node during the start of the execution, and Δ the maximum degree of any vertex. This time, we define γ' slightly differently, namely $\gamma' = 2\Delta + 2\theta$.

THEOREM 3.1. *There is a constant B (depending on H, θ and G), such that for any adversary respecting the cut condition, $\texttt{MCLB}_\theta$ ensures $|b_t^{(i)}(v)| \leq B$ at all times t, for all vertices v, and commodities $i = 1, 2$.*

Proof. At the start of the execution, $|b_1^{(1)}(S)| + |b_1^{(2)}(S)| \leq \sum_{v \in S} H$, by definition of H. The key Lemma 3.2 establishes that for all $S \subseteq V$, times t, and commodities $i = 1, 2$,

$$|b_t^{(1)}(S)| + |b_t^{(2)}(S)| \leq \sum_{j=n-|S|+1}^{n} (H + \gamma' \cdot j) \tag{3.2}$$

We can then apply the result to all singleton sets $\{v\}$, proving the theorem. □

LEMMA 3.2. *If (3.2) holds at the beginning of round t, it holds at the beginning of round $t + 1$.*

Proof. The proof is by contradiction. Let S be a set (of minimum size in case of ties) maximizing

$$\Phi(S) := |b_t^{(1)}(S)| + |b_t^{(2)}(S)| - \sum_{j=n-|S|+1}^{n} (H + \gamma' \cdot j).$$

In the case of two commodities, a node might be included in S because it contributes a lot to the imbalance in one of the commodities, although its contribution to the other commodity might actually be negative. To capture the imbalance contribution of a node to each commodity, we define the *signed imbalance* $\beta_{t'}^{(i)}(v) := \text{sgn}(b_{t+1}^{(i)}(S)) \cdot$

$b_{t'}^{(i)}(v)$, $\overline{\beta}_{t'}^{(i)}(v) := \operatorname{sgn}(b_{t+1}^{(i)}(S)) \cdot \overline{b}_{t'}^{(i)}(v)$, for every node $v \in V$, and every time step t'. Here, sgn denotes the sign of a term. Notice that we use the sign of $b_{t+1}^{(i)}(S)$ at all time steps t' in the definition of $\beta_{t'}^{(i)}(v)$ (not the sign of $b_{t'}^{(i)}(S)$ or of $b_{t'+1}^{(i)}(S)$). We can now rewrite the total imbalance over the set S at time $t+1$ as

$$|b_{t+1}^{(1)}(S)| + |b_{t+1}^{(2)}(S)| = \sum_{u \in S} (\beta_{t+1}^{(1)}(u) + \beta_{t+1}^{(2)}(u)). \tag{3.3}$$

Again, we show that the change in the imbalance for set S cannot be positive, by comparing the increase in the imbalance of S during the Adversary step with the decrease during the Redistribution step, and thus obtain a contradiction.

Adversary step: We know that for each commodity $i = 1, 2$, the imbalance on set S after the Adversary step is at most $|\overline{b}_t^{(i)}(S)| \le |b_t^{(i)}(S)| + |\delta_t^{(i)}(S) - |S| \cdot (a_{t+1}^{(i)} - a_t^{(i)})|$ by the Triangle Inequality. Now, summing over $i = 1, 2$ and applying the cut condition on the adversary yields that

$$|\overline{b}_t^{(1)}(S)| + |\overline{b}_t^{(2)}(S)| \le |b_t^{(1)}(S)| + |b_t^{(2)}(S)| + |e(S)|.$$

Redistribution step: Fix a node $u \in S$, and an edge $e = (u, v)$ of G with $v \notin S$. As in the proof of Lemma 2.2, we can use the definition of S (as maximizing Φ and being of minimal size) to obtain that the signed imbalances at nodes u and v satisfy $\beta_{t+1}^{(1)}(u) + \beta_{t+1}^{(2)}(u) > \beta_{t+1}^{(1)}(v) + \beta_{t+1}^{(2)}(v) + \gamma'$. As u and v can lose and gain at most Δ tokens each during the Redistribution step,

$$\overline{\beta}_t^{(1)}(u) + \overline{\beta}_t^{(2)}(u) > \overline{\beta}_t^{(1)}(v) + \overline{\beta}_t^{(2)}(v) + 2\theta. \tag{3.4}$$

In particular, there must be a commodity i such that $\overline{\beta}_t^{(i)}(u) > \overline{\beta}_t^{(i)}(v) + \theta$, and thus also $|\overline{h}_t^{(i)}(u) - \overline{h}_t^{(i)}(v)| > \theta$. Hence, $\texttt{MCLB}_\theta$ moved a token along edge e during the Redistribution step (w.l.o.g., it was a token of commodity 1). We want to show that this token actually decreased the signed imbalance of node u. Assume that it did not. This means that if the token moved from u to v, then $\operatorname{sgn}(b_{t+1}^{(1)}(S))$ is negative, and if the token moved from v to u, then $\operatorname{sgn}(b_{t+1}^{(1)}(S))$ is positive. In either case, the signed imbalance for commodity 1 at node v must be higher than at node u, and so

$$\overline{\beta}_t^{(1)}(v) - \overline{\beta}_t^{(1)}(u) = |\overline{b}_t^{(1)}(v) - \overline{b}_t^{(1)}(u)| = |\overline{h}_t^{(1)}(v) - \overline{h}_t^{(1)}(u)|.$$

Because $\texttt{MCLB}_\theta$ maximizes the difference in its choice of commodity, we obtain that

$$\begin{aligned} \overline{\beta}_t^{(1)}(v) - \overline{\beta}_t^{(1)}(u) &= |\overline{h}_t^{(1)}(v) - \overline{h}_t^{(1)}(u)| \\ &\ge |\overline{h}_t^{(2)}(v) - \overline{h}_t^{(2)}(u)| \\ &= |\overline{\beta}_t^{(2)}(v) - \overline{\beta}_t^{(2)}(u)| \\ &\ge \overline{\beta}_t^{(2)}(u) - \overline{\beta}_t^{(2)}(v). \end{aligned}$$

Rearranging this inequality yields that

$$\overline{\beta}_t^{(1)}(v) + \overline{\beta}_t^{(2)}(v) \ge \overline{\beta}_t^{(1)}(u) + \overline{\beta}_t^{(2)}(u),$$

and thus a contradiction with Inequality (3.4). Therefore, every edge (u, v) with $v \notin S$ decreases the signed imbalance of u by 1. Summing over all edges and all nodes $u \in S$, gives us $\beta^{(1)}_{t+1}(S)+\beta^{(2)}_{t+1}(S) \leq \overline{\beta}^{(1)}_{t}(S)+\overline{\beta}^{(2)}_{t}(S)-|e(S)|$. Using Equation (3.3) and the fact that $\overline{\beta}^{(1)}_{t}(S)+\overline{\beta}^{(2)}_{t}(S) \leq |\overline{b}^{(1)}_{t}(S)|+|\overline{b}^{(2)}_{t}(S)|$, we obtain $|b^{(1)}_{t+1}(S)|+|b^{(2)}_{t+1}(S)| \leq |\overline{b}^{(1)}_{t}(S)|+|\overline{b}^{(2)}_{t}(S)|-|e(S)|$.

Combining the arguments for the two steps, $|b^{(1)}_{t}(S)|+|b^{(2)}_{t}(S)|$ increases by at most $|e(S)|$ during the Adversary step, and decreases by at least $|e(S)|$ during the Redistribution step. Therefore, in total $|b^{(1)}_{t+1}(S)|+|b^{(2)}_{t+1}(S)| \leq |b^{(1)}_{t}(S)|+|b^{(2)}_{t}(S)|$, a contradiction. □

The result for two commodities can be extended to networks with edge capacities, adversarially changing edge sets, and adversaries with restrictions only for larger window sizes, just like for the single commodity case.

3.2. Load balancing and flows. By omitting the adversarial and dynamic nature in the load balancing problem, and forcing the adversary to repeat the same pattern of token additions in every round, we can infer from the stability of the load balancing algorithm the existence of a multi-commodity flow. Suppose that we are given a multi-commodity flow instance with edge capacities c_e, source-sink pairs (s_i, t_i), and demands d_i. Let $D = \max_i d_i$, and let $\mathcal{A}$ be the adversary inserting, in every round and for each commodity i, $D + d_i$ tokens at node s_i, $D - d_i$ tokens at node t_i, and D tokens everywhere else.

LEMMA 3.3. *If there is a load-balancing algorithm that is stable against the adversary $\mathcal{A}$, then there is a (fractional) multi-commodity flow f with source-sink pairs (s_i, t_i) and demands d_i.*

Proof. Because we assumed the algorithm to be stable, all imbalances $b^{(i)}_{t}(v)$ are always bounded in absolute value by some constant B. Therefore, there are at most $(2B+1)^{kn}$ different combinations of imbalances for the entire network, and so there must exist two timesteps t and t' such that $t < t'$ and $b^{(i)}_{t}(v) = b^{(i)}_{t'}(v)$ for all nodes v and commodities i.

For each edge $e = (u, v)$, let $\sigma^{(i)}_{t}(u, v)$ denote the number of tokens of commodity i sent from u to v in round t, and define a flow f by

$$f^{(i)}_{(u,v)} := \frac{1}{t'-t} \cdot \sum_{r=t}^{t'-1} (\sigma^{(i)}_{r}(u,v) - \sigma^{(i)}_{r}(v,u)).$$

Notice that we define negative flows, but only for symmetry and ease of notation. We want to verify that f is indeed a feasible multi-commodity flow for demands (s_i, t_i, d_i).

Capacity constraints: The total flow along any edge (u, v) is

$$\begin{aligned}
\sum_i |f^{(i)}_{(u,v)}| &\leq \frac{1}{t'-t} \cdot \sum_i \sum_{r=t}^{t'-1} |\sigma^{(i)}_{r}(u,v) - \sigma^{(i)}_{r}(v,u)| \\
&= \frac{1}{t'-t} \cdot \sum_{r=t}^{t'-1} \sum_i |\sigma^{(i)}_{r}(u,v) - \sigma^{(i)}_{r}(v,u)| \\
&\leq \frac{1}{t'-t} \cdot \sum_{r=t}^{t'-1} c_{(u,v)} \\
&= c_{(u,v)}.
\end{aligned}$$

The first inequality is simply the Triangle inequality, and the second inequality holds because the balancing algorithm never exceeds the capacity of any edge with any of its token moves, and therefore both $\sigma_r^{(i)}(u,v)$ and $\sigma_r^{(i)}(v,u)$ lie between 0 and $c_{(u,v)}$.

Flow Conservation: For any node v and commodity i, we can write

$$\begin{aligned}
&(t'-t)\sum_{(u,v)\in E} f^{(i)}_{(u,v)}\\
&= \sum_{(u,v)\in E}\sum_{r=t}^{t'-1}(\sigma_r^{(i)}(u,v)-\sigma_r^{(i)}(v,u))\\
&= \sum_{r=t}^{t'-1}\sum_{(u,v)\in E}\sigma_r^{(i)}(u,v)-\sum_{r=t}^{t'-1}\sum_{(u,v)\in E}\sigma_r^{(i)}(v,u)\\
&= h_{t'}^{(i)}(v)-h_t^{(i)}(v)-\sum_{r=t}^{t'-1}\delta_r^{(i)}(v)\\
&= b_{t'}^{(i)}(v)+a_{t'}^{(i)}-b_t^{(i)}(v)-a_t^{(i)}-(t'-t)\cdot\delta_t^{(i)}(v)\\
&= (t'-t)(D-\delta_t^{(i)}(v)).
\end{aligned}$$

The third equality is true because $h_{t'}^{(i)}(v)-h_t^{(i)}(v)$ is exactly the number of tokens that entered v during the time period $[t,t'-1]$, minus the amount of tokens that left v during this period. In the last equality, we used that $b_{t'}^{(i)}(v)=b_t^{(i)}(v)$ for all i and v, and that $a_r^{(i)}=r\cdot D$ for all times r. Now, if node v is neither the source nor the sink for commodity i, then $\delta_t^{(i)}(v)=D$, so flow is conserved. If v is the source of commodity i, then $\delta_t^{(i)}(v)=D+d_i$, so the total flow entering node s_i is $-d_i$. If v is the sink for commodity i, then the total flow entering node t_i is d_i, because $\delta_t^{(i)}(v)=D-d_i$ by definition. Hence, f satisfies flow conservation and all demands.

f conserves flow, satisfies all demands, and does not exceed any edge capacities, so it is a feasible multi-commodity flow for the given demands. □

By combining Lemma 3.3 with the stability of $\texttt{MCLB}_\theta$ proved in Theorem 3.1, we obtain as a corollary an alternate proof of the two-commodity Max-Flow Min-Cut Theorem. It only remains to verify that the adversary $\mathcal{A}$ as defined in Lemma 3.3 indeed respects the cut-condition.

COROLLARY 3.4 (2-Commodity Max-Flow Min-Cut). *Let $G=(V,E)$ be a graph with edge capacities c_e, and two demand pairs (s_1,t_1), (s_2,t_2) with demands d_1, d_2 such that for any vertex set $S\subseteq V$, the total demand of commodities $i\in\{1,2\}$ with exactly one of $\{s_i,t_i\}$ in S is at most $\sum_{e\in e(S)}c_e$. Then, there exists a feasible two-commodity flow sending d_i units of flow from s_i to t_i for $i=1,2$.*

Proof. Define the adversary $\mathcal{A}$ as in Lemma 3.3. To show that the algorithm $\texttt{MCLB}_\theta$ is stable against $\mathcal{A}$, we merely have to verify that $\mathcal{A}$ satisfies the cut condition. Let $S\subseteq V$ be arbitrary. For convenience, we write $[u\in S]:=1$ if $u\in S$, and 0 otherwise. Then, for any time t

$$\begin{aligned}
&\sum_{i=1,2} |\delta_t^{(i)}(S) - |S| \cdot (a_{t+1}^{(i)} - a_t^{(i)})| \\
&= \sum_{i=1,2} ||S| \cdot D + [s_i \in S] \cdot d_i - [t_i \in S] \cdot d_i - |S| \cdot D| \\
&= \sum_{i=1,2} d_i \cdot |[s_i \in S] - [t_i \in S]|.
\end{aligned}$$

In the first equality, we used the definition of the insertion pattern for $\mathcal{A}$. The contribution of commodity i to this sum is d_i if and only if exactly one of s_i, t_i lies in S — otherwise, it is 0. Hence, the value of the sum is the total demand of commodities i with exactly one of $\{s_i, t_i\}$ in S, which by assumption is bounded by $\sum_{e \in e(S)} c_e$. Hence, $\mathcal{A}$ satisfies the cut condition.

We can therefore apply Theorem 3.1 to obtain that $\texttt{MCLB}_\theta$ is stable against $\mathcal{A}$, which in turn implies the existence of a feasible multi-commodity flow f for the given instance via Lemma 3.3. $\square$

The 2-Commodity Max-Flow Min-Cut Theorem was first proved by Hu [17], essentially repeating Ford and Fulkerson's original [14] augmenting paths argument for two commodities. Seymour [23] showed a short and simple explicit reduction to the single-commodity case. Subsequently, Linial, London and Rabinovich [19] gave a novel proof using geometric embeddings and linear programming duality. Our proof uses yet different (and much more elementary) techniques, and does not rely on the single-commodity Max-Flow Min-Cut Theorem.

Another corollary we obtain from Lemma 3.3 is the existence of an adversary respecting the cut condition for $k = 3$ commodities, such that no algorithm (offline or online) can balance the insertion pattern. This is not surprising, since load balancing algorithms (and our algorithm in particular) attempt to generate a flow from over-loaded nodes to under-loaded nodes, and we know that the best flows can be significantly worse than the best cuts for more than 2 commodities (i.e., the Max-Flow Min-Cut Theorem does not hold for $k \geq 3$). This means that the constraint on the adversary is not powerful enough to guarantee the existence of a good flow and therefore a good algorithm. We make the above discussion precise in the following corollary.

COROLLARY 3.5. *There exists an adversary respecting the cut condition for $k = 3$ commodities, such that no algorithm (offline or online) is stable against this adversary.*

Proof. To prove this, we simply take a 3-commodity instance with a graph $G = (V, E)$ and demand pairs $(s_i, t_i), i \in \{1, 2, 3\}$ (with demands d_i) such that for all cuts $(S, V \setminus S)$, the total demand across the cut is at most the capacity of the edges crossing the cut, yet there is no (fractional) multi-commodity flow satisfying all demands. The first such example for $k = 3$ was given in [17]. (A simpler well-known example for $k = 4$ is the complete bipartite graph $K_{2,3}$ with a unit demand between every pair of nodes that are at a distance of 2 from each other.) Let $\mathcal{A}$ be the adversary defined from this instance as in Lemma 3.3. If any load balancing algorithm were stable against $\mathcal{A}$, Lemma 3.3 would guarantee a feasible multi-commodity flow, a contradiction. $\square$

The above corollary and Lemma 3.3 illustrate the relation of multi-commodity flows to load-balancing algorithms, and tell us that the cut condition is not enough to guarantee the existence of a stable algorithm for $k \geq 3$. In Section 5 we will discuss an alternative approach for a restriction on a multi-commodity adversary.

4. Packet Routing. There is a natural connection between the load balancing problem studied in the previous sections, and the problem of routing packets in an adversarial network. It has been observed previously [2, 6] that the natural balancing algorithm $\texttt{SCLB}_\theta$ is also stable for packet routing.

The model for packet routing differs from the load balancing one in that after the Redistribution step, there is an additional *Removal step*, during which all packets that have reached their destination are removed from the network. Stability of an algorithm is now defined as meaning that there is an absolute bound on all queue heights at all times, i.e., $h_t^{(i)}(v) \leq B$ for some constant B.

In the single-commodity packet routing problem, we can again restrict the adversary by a cut condition: the total number $\delta_t(S)$ of packets inserted into a set S must be at most $|e(S)|$ for any set S not containing the sink of the packets. If S does contain the sink, then there is no restriction. In the multi-commodity case, the adversary specifies a source s_i and a sink t_i for each packet inserted, and must guarantee that there is a set of edge-disjoint paths connecting all s_i-t_i pairs.

In [6], it was shown that for the single-commodity case, the algorithm $\texttt{SCLB}_\theta$ is stable against an adversary guaranteeing the existence of a path for all packets inserted, even when edges dynamically appear and disappear. Aiello et. al. [2] proved that an algorithm essentially equivalent to $\texttt{MCLB}_\theta$ is stable for the multi-commodity packet routing problem if the paths specified by the adversary are not only disjoint, but leave an ε fraction of capacity for every edge unused over a given window length W. Recently, [**?**] extended this result to dynamically changing networks.

Our techniques from Section 2 can be used to obtain an alternate (and simpler) proof for the stability of algorithm $\texttt{SCLB}_\theta$ in the packet routing model. Our proof also works for the case of adversarially appearing and disappearing edges, although the restriction on the adversary is different from (and essentially less general than) the one in [6].

We define Δ and H as before, and let $\gamma = 2\Delta + \theta$. Then, the stability of $\texttt{SCLB}_\theta$ against an adversary respecting the cut condition is guaranteed by the following theorem.

THEOREM 4.1. *For any time t and set $S \subseteq V$,*

$$h_t(S) \leq \sum_{j=n-|S|+1}^{n} (H + \gamma \cdot j). \tag{4.1}$$

Proof. By definition of H, Invariant (4.1) certainly holds at time 1. Assume that the theorem is wrong, and let t be the earliest time such that there is a set S violating (4.1) at time $t+1$. Among all such sets, let S be the one maximizing $h_{t+1}(S) - \sum_{j=n-|S|+1}^{n}(H + \gamma \cdot j)$, and break ties for minimal size. Then, we can show as in the proof of Lemma 2.2 that for all nodes $u \in S, v \notin S$, $h_{t+1}(u) > h_{t+1}(v) + \gamma$, and $h_{t+1}(u) > H$. In particular, the set S cannot contain the sink (because the sink contains no tokens after the Removal step).

Therefore, the adversary can have inserted at most $|e(S)|$ tokens into S in the Adversary step. During the Redistribution step of round t, a token leaves the set S along each edge $e = (u, v) \in e(S)$, because even if u had lost Δ tokens and v gained Δ tokens during the Redistribution step, $\overline{h}_t(u)$ would still exceed $\overline{h}_t(v)$ by at least θ. Taken together, this shows that the number of tokens in S cannot have increased, contradicting the choice of t and S. $\square$

Extensions. Like the proofs in the previous sections, the proof of Theorem 4.1 can be easily extended to deal with dynamically changing networks, edge capacities, and time windows in the cut condition. In addition, this proof extends to directed graphs G, if we redefine $e(S)$ in the cut condition to be the edges coming out of S. We can also show stability for a wider class of single-commodity balancing algorithms. Specifically, let $g : \mathbb{N} \to \mathbb{N}$ be a function with $g(x) \geq x$ for all x. The balancing algorithm $\texttt{SCLB}_g$ always sends a token from u to v (and never sends a token from v to u) if there is an edge $e = (u, v) \in E$ and $h_t(u) > g(h_t(v))$. It does not matter what the algorithm does if neither $h_t(u) > g(h_t(v))$ nor $h_t(v) > g(h_t(u))$. Extending the above proof only slightly, we obtain that $\texttt{SCLB}_g$ is stable for all such functions g. Of course, the bound B now depends on the rate of growth of g.

Our packet routing results also imply the stability of an interesting load balancing scenario, which is very similar to the one in [10]. Suppose each node processes one job per round, so that one token is removed every round from each node with positive queue height. If the adversary $\mathcal{A}$ satisfies the constraint that the number of tokens it adds to S is at most $e(S) + |S|$, then we can use Theorem 4.1 to show that the queue heights are bounded. Consider adding a sink node v to our graph, and add an edge from every node to v. We can now think of $\mathcal{A}$ as adding packets to the new graph, with v being the packets' destination. The number of edges coming out of a set S in the new graph is exactly $e'(S) = e(S) + |S|$, so $\mathcal{A}$ satisfies the condition needed for Theorem 4.1. It is easy to see that if the queue heights are bounded in this new packet routing scenario, then they are also bounded in the original load balancing one.

The proofs for Theorem 4.1 and Theorem 2.1 (and the proofs in [2] and [21]) are so similar in nature that one suspects a formal reduction from the packet routing problem to the load balancing problem (which seems more general). However, we have not yet been able to determine such a reduction. It would certainly be interesting, since it would allow us to focus on the load balancing problem in the future.

As with the load balancing problem, we can obtain a multi-commodity flow if $\texttt{MCLB}_\theta$ is stable against a suitably defined adversary $\mathcal{A}$. The proof is practically identical to the one for Lemma 3.3, and we therefore omit it.

LEMMA 4.2. *Let $\mathcal{A}$ be an adversary inserting d_i tokens of commodity i (whose destination is t_i) into node s_i in every round. If there is a routing algorithm that is stable against this adversary, then there is a (fractional) multi-commodity flow f with source-sink pairs (s_i, t_i) and demands d_i.*

5. Conclusions.

In this paper, we have shown that a simple local load-balancing algorithm is stable against dynamic adversarial addition and removal of jobs in a network, so long as the adversary is bounded by a natural extension of the cut condition in the sense defined in [21]. This settles an open question from [21]. Our proof techniques extend to the case of balancing two commodities at once, and to routing packets injected by an adversary. They yield easier proofs and essentially tight bounds for the general case. In addition, the stability of the load balancing algorithm for two commodities gives a new proof of the two-commodity Max-Flow Min-Cut Theorem.

This work leaves open a number of interesting questions. Most importantly, we would like to be able to show stability of the multi-commodity load balancing algorithm for an arbitrary number of commodities, both for the problem of routing packets and balancing loads. If we want to prove stability of load-balancing algorithms for more commodities, we will have to use a different condition on the adversary. As a consequence of Lemma 3.3, any reasonable restriction on the adversary will have to guarantee the existence of multi-commodity flows for all instances where we hope to prove stability. We therefore suggest the following restriction:

> First, define demands $d_t^{(i)}(v)$ for commodity i, node v and time t by $d_t^{(i)}(v) := \delta_t^{(i)}(v) - (a_{t+1}^{(i)} - a_t^{(i)})$. Then, the adversary is restricted to moves that guarantee the existence of a (fractional) multi-commodity flow in G satisfying all these demands.

The disadvantage of this condition is that it bears no direct relation to the load balancing problem — it arises from observing the insufficiency of the more natural cut condition rather than from having an actual meaning for the problem of balancing loads. Nevertheless, this condition should be considered the right restriction on the adversary to measure the quality of $\texttt{MCLB}_\theta$ or other load balancing algorithms.

Alternatively, we might investigate whether the cut condition is sufficient for balancing multiple loads if we restrict our attention to specific networks. For example, it is well known that for trees or cycles, the cut condition implies the existence of multi-commodity flows, and we might hope that it would hence be sufficient to prove stability.

Acknowledgments. We thank Martin Pál and Christian Scheideler for valuable discussions on the subjects of load balancing and adversarial packet routing.

REFERENCES

[1] W. Aiello, B. Awerbuch, B. Maggs, and S. Rao, *Approximate load balancing on dynamic and asynchronous networks*, in Proc. 25th ACM Symp. on Theory of Computing, 1993.

[2] W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosén, *Adaptive packet routing for bursty adversarial traffic*, in Proc. 30th ACM Symp. on Theory of Computing, 1998.

[3] A. Anagnostopoulos, A. Kirsch, and E. Upfal, *Load balancing in arbitrary network topologies with stochastic adversarial input*, SIAM J. Comput., 34 (2005), pp. 616–639.

[4] D. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, F. Leighton, and Z. Liu, *Universal stability results for greedy contention–resolution protocols*, in Journal of the ACM, vol. 48, 2001, pp. 39–69.

[5] F. M. auf der Heide, B. Oesterdiekhoff, and R. Wanka, *Strongly adaptive token distribution*, Algorithmica, 15 (1996).

[6] B. Awerbuch, P. Berenbrink, A. Brinkmann, and C. Scheideler, *Simple routing strategies for adversarial systems*, in Proc. 42nd IEEE Symp. on Foundations of Computer Science, 2001.

[7] B. Awerbuch and F. Leighton, *A simple local-control approximation algorithm for multi-commodity flow*, in Proc. 34th IEEE Symp. on Foundations of Computer Science, 1993.

[8] ———, *Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks*, in Proc. 26th ACM Symp. on Theory of Computing, 1994.

[9] P. Berenbrink, T. Friedetzky, and L. Goldberg, *The natural work-stealing algorithm is stable*, SIAM J. Comput., 32 (2003), pp. 1260–1279.

[10] P. Berenbrink, T. Friedetzky, and R. Martin, *Dynamic diffusion load balancing*, in Proc. 32nd Intl. Colloq. on Automata, Languages and Programming, 2005, pp. 1386–1398.

[11] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. Williamson, *Adversarial queueing theory*, in Journal of the ACM, vol. 48, 2001, pp. 13–38.

[12] M. Crovella, M. Harchol-Balter, and C. Murta, *Task assignment in a distributed system: Improving performance by unbalancing load*, in Proc. ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems, 1998.

[13] D. Eager, E. Lazowska, and J. Zahorjan, *Adaptive load sharing in homogeneous distributed systems*, IEEE Transactions on Software Engineering, 12 (1986).

[14] L. Ford and D. Fulkerson, *Maximal flow through a network*, Can. J. Math, 8 (1956), pp. 399–404.

[15] D. Gamarnik, *Stability of adaptive and non-adaptive packet routing policies in adversarial queueing networks*, in Proc. 31st ACM Symp. on Theory of Computing, 1999.

[16] B. Ghosh, F. Leighton, B. Maggs, S. Muthukrishnan, C. Plaxton, R. Rajaraman,

A. Richa, R. Tarjan, and D. Zuckerman, *Tight analyses of two local load balancing algorithms*, in Proc. 27th ACM Symp. on Theory of Computing, 1995.
[17] T. Hu, *Multi-commodity network flows*, Operations Research, 11 (1963).
[18] F. Leighton and S. Rao, *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*, Journal of the ACM, 46 (1999), pp. 787–832.
[19] N. Linial, E. London, and Y. Rabinovich, *The geometry of graphs and some of its algorithmic applications*, Combinatorica, 15 (1995).
[20] M. Mitzenmacher, *On the analysis of randomized load balancing schemes*, in Proc. 9th ACM Symp. on Parallel Algorithms and Architectures, 1997.
[21] S. Muthukrishnan and R. Rajaraman, *An adversarial model for distributed dynamic load balancing*, in Proc. 10th ACM Symp. on Parallel Algorithms and Architectures, 1998.
[22] D. Peleg and E. Upfal, *The token distribution problem*, SIAM J. on Computing, 18 (1989).
[23] P. Seymour, *A short proof of the two-commodity flow theorem*, J. of Combinatorial Theory, 26 (1979).
[24] B. Shirazi, A. Hurson, and K. Kavi, *Scheduling and Load Balancing in Parallel and Distributed Systems*, IEEE Computer Society Press, 1995.