Competitive Weighted Throughput Analysis of Greedy Protocols on DAGs

EYAL GORDON
Dept. of Computer Science, Technion, Israel
and
ADI ROSÉN
CNRS and University of Paris 11, France

The combination of the buffer sizes of routers deployed in the Internet, and the Internet traffic itself, leads routinely to the dropping of packets. Motivated by this, we are interested in the problem of maximizing the throughput of protocols that control packet networks. Moreover, we are interested in a setting where different packets have different priorities (or weights), thus taking into account Quality-of-Service considerations.

We first extend the Competitive Network Throughput (CNT) model introduced by Aiello et al. [Aiello et al. 2003] to the weighted packets case. We analyze the performance of online, local-control protocols by their competitive ratio, in the face of arbitrary traffic, using as a measure the total weight of the packets that arrive to their destinations, rather than being dropped en-route. We prove that on directed acyclic graphs (DAGs), any greedy protocol is competitive, with competitive ratio independent of the weights of the packets. Here we mean by a "greedy protocol" a protocol that not only does not leave a resource idle unnecessarily, but also also prefers packets with higher weight over those with lower weight. We give two independent upper bounds on the competitive ratio of general greedy protocols on DAGs. We further give lower bounds that show that our upper bounds cannot be improved (other than constant factors) in the general case. Both our upper and lower bounds apply also to the unweighted case, and they improve the results given in [Aiello et al. 2003] for that case. We thus give tight (up to constant factors) upper and lower bounds for both the unweighted and weighted cases.

In the course of proving our upper bounds we prove a lemma that gives upper bounds on the delivery times of packets by any greedy protocol on general DAGs (without buffer size considerations). We believe that this lemma may be of independent interest and may find additional applications.

Categories and Subject Descriptors: C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Packet-Switching Network, Store and Forward Networks*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Non Numerical Algorithms and Problems—*Routing and Layout, Sequencing and Scheduling*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Network Problems*

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: Buffer Management, Competitive Network Throughput, Online Algorithms, Competitive Analysis

A preliminary version of this paper appeared in the Proc. of PODC 2005, pp. 227–236.
Authors' addresses: Eyal Gordon, Dept. of Computer Science, Technion, Haifa 32000, Israel. Adi Rosén, CNRS & University of Paris 11, LRI Bât. 490, Université Paris Sud, 91405 Orsay, France (e-mail: `adiro@lri.fr`).

1. INTRODUCTION

Packet data networks, such as the Internet, are becoming the dominant technology for transferring all types of data. The research into the performance of the protocols that control them is therefore of key importance, and is attracting increasing attention. In this paper we consider the question of maximizing the throughput guaranteed by the protocols that control the network, that is, maximizing the amount of data that arrive to destination rather than being dropped en-route.

In the Internet, the combination of the buffer sizes of deployed routers and the Internet traffic itself leads routinely to the dropping of packets. Buffer space in the routers is limited, and at the same time, the behavior of traffic in the Internet is both hard to predict and hard to mathematically (probabilistically) model. Devising protocols that will maximize the throughput under these conditions, and giving rigorous analysis for their performance, is therefore a challenging mission. In the context of a single switch, the work of Aiello et al. [Aiello et al. 2005] initiated the research into the performance of online protocols to maximize throughput in the face of limited buffer space and arbitrary traffic, and was followed by a number of papers in recent years, e.g. [Lotker and Patt-Shamir 2002; Kesselman et al. 2001; Andelman et al. 2003; Kesselman and Rosén 2006; Azar and Richter 2004a] (for a short survey see [Epstein and van Stee 2004]). However, good performance on the single switch level does not necessarily guarantee good performance on the network level, where traffic is injected continuously in various points in the networks, and various traffic streams interact with each other.

This has motivated, more recently, the introduction of the Competitive Network Throughput (CNT) model [Aiello et al. 2003], which is aimed at analyzing the throughput of protocols that control whole networks in the face of limited buffer space and arbitrary traffic. In this model, the network is represented by a directed graph, where each node models a router and each edge models a communication link. There is a buffer at the tail of each edge, capable of storing up to B packets, for some value B. Packets travel in the network in a store-and-forward manner, controlled by the protocol that manages the network. Since buffer space at the tail of the edges is limited, some packets must at times be dropped. The performance of an online, local-control protocol that controls the network is analyzed by its competitive ratio where the measure is the *number* of packets that arrive to their destination, rather than being dropped en-route. Of particular interest is the analysis of greedy protocols, that is, protocols that do not leave a resource (buffer space, or communication link) idle when this resource can be used. Such protocols may give preference to the packets according to various policies, e.g., FIFO, Nearest-To-Go (NTG), Furthest-To-Go (FTG), Longest-In-System (LIS) etc. Aiello et al. [Aiello et al. 2003] gave a number of results showing which greedy policies are superior to others.

However, the more traditional best-effort approach, which treats all packets equally, is not sufficient when heterogeneous traffic is transmitted through the network. For example, the DiffServ approach [Clark and Wroclawski 1997], requires

to treat different packets differently. This is usually modeled by assigning to each packet a weight which signifies its importance or priority.

In the present work we address this setting, and analyze the *weighted throughput* of online, local-control protocols that control packet networks. Our work extends and improves the work of Aiello et al. [Aiello et al. 2003] in two ways. First, we extend the CNT model to the weighted case, by assigning each packet a weight. The weights of the packets may have any range, and may contain any number of distinct values. The aim of the protocol is now not to maximize the *number* of packets that reach their destinations, but rather to maximize the *total weight* of those packets that reach their destinations. Second, in this work, we focus on directed acyclic graphs (DAGs) and prove that any greedy protocol is competitive on all DAGs, with competitive ratios independent of the range of the weights of the packets, or the number of distinct weights. Here we mean by a "greedy protocol" a protocol that not only does not leave a resource idle unnecessarily, but also also prefers packets with higher weight over those with lower weight. We give two independent *tight bounds* (up to constant factors) on the competitive ratios of greedy protocols on DAGs under the extended, weighted model. Our upper bounds also improve upon the upper bounds given in [Aiello et al. 2003] for the unweighted case, and our lower bounds apply also to the unweighted case. Thus we also improve the results given in [Aiello et al. 2003] and give tight (up to constant factors) bounds for both the unweighted and weighted cases.

1.1 Our results

We start by defining a generic protocol named *Maximal-Weight (MAXW)*. This protocol is greedy, as defined in [Aiello et al. 2003]: No packets are dropped when buffer space is available, and a packet is forwarded on an edge whenever possible. We extend the definition of greediness into the weighted packets model by stating that a greedy protocol always favors packets of higher weight over packets of lower weight when deciding which packets are to be stored in buffers, and which are first forwarded over a link. MAXW is a generic greedy protocol, which can store in a buffer any set of packets with maximal total weight. The remaining packets, if any, are dropped. When deciding which packet to forward over a link, any packet with maximal weight in the buffer can be chosen. Defining MAXW as a generic greedy protocol ensures that any upper bound proved for MAXW applies also to any specific protocol that adheres to the greediness property.

We prove that any greedy protocol is competitive on any DAG in the weighted model. We give two independent upper bounds on the competitive ratio of the generic protocol MAXW on a general DAG $G = (V, E)$ (where $|E| = m$, $|V| = n$, and d denotes the length of the longest simple path in G).

—We prove that MAXW is $4m(1 + \frac{d}{B})$ - competitive on any DAG.

—Next, we prove that on a DAG G, MAXW is $f(G)$-competitive, where $f(G)$ is the maximal following value computed for an edge e in G: sum over all edges e' the number of different *lengths* of paths which start at e' and end at e. More formally, for any two edges $e, e' \in E$, and any k, $1 \leq k \leq d$, let $f'(e', k, e)$ be 1 if there exists a path of length k edges whose first edge is e' and last edge is e, and 0 otherwise. We define for any $e \in E$:

$f(e) \triangleq \sum_{e' \in E} \sum_{k=1}^{d} f'(e', k, e)$.
For the graph G we define:
$f(G) \triangleq \max_{e \in E} f(e)$.

The first upper bound takes into account the size of the buffer, B, while the second one does not. For constant values of B the second upper bound significantly improves upon the first one for many network topologies, such as layered graphs where $f(G) \leq m$; for general DAGs, if $d \leq 4$ or $B \leq \frac{4d}{d-4}$, the second upper bound improves upon the first one by at least a constant. However, the first upper bound takes into account the size of the buffer B, and shows that with large enough buffer sizes (e.g. $B = \Omega(d)$) one may reach the competitive ratio of $O(m)$ on *any* DAG, while the upper bound of $f(G)$ may be $\omega(m)$ on general DAGs. As we discuss below, we further give lower bounds that show that there exist DAGs on which our upper bounds are tight up to constant factors. Naturally, our upper bounds hold for the case of equal weight packets as well. This makes them comparable to the results shown in [Aiello et al. 2003] for the competitive ratio of any greedy protocol on DAGs in the unweighted case. Those results use a different function, $h(G)$, and show an upper bound of $O(h(G))$ on the competitive ratio of any greedy protocol. The function $h(G)$ used in [Aiello et al. 2003] is exponential on certain DAGs, whereas both upper bounds shown in the present work are polynomial. Moreover, for any DAG G, the upper bound $f(G)$ that we give here is better (i.e., on many topologies smaller, and never larger) than the upper bound $O(h(G))$ given in [Aiello et al. 2003].

In the course of proving our first upper bound we prove a lemma (Lemma 6) that gives upper bounds on the delivery times of packets by any greedy protocol on general DAGs when buffer size is unbounded. We show that if k packets, with arbitrary paths to follow, are injected into the network and the last injection time is t, then *all* k packets are delivered to their destinations by time $t + d + k - 1$. We believe that this lemma may be of independent interest and may find additional applications.

We complement our results with two lower bounds. We first give a certain DAG and show on it a lower bound of $\Omega(n^2(1+\frac{n}{B}))$ on the competitive ratio of the protocol Furthest-To-Go (FTG) for the unweighted packets model (and consequently for the variable-weight packet model). This lower bound matches, up to a constant factor, our first upper bound, given for any greedy protocol (i.e. MAXW) on the variable-weight packet model. This result has two consequences: (1) It shows that for general DAGs and a generic greedy protocol, our upper bound of $4m(1 + \frac{d}{B})$ cannot be improved (up to constant factors), and (2) it strengthens conclusions given in [Aiello et al. 2003], showing that although FTG is a stable protocol in the Adversarial Queuing Theory model (see [Andrews et al. 1996]), it is not a good protocol when throughput is in question. In fact our result shows that the competitive ratio of FTG on a general DAG is the worst competitive ratio possible for a greedy protocol. We then give another DAG and show another, simpler lower bound of $\Omega(n^2)$ on the competitive ratio of any greedy protocol on a DAG in the unweighted model (and consequently on the weighted packet model). The DAG G used in this proof is a layered graph, for which $f(G)$, our second upper bound, is m. This lower bound thus matches our second upper bound, up to a constant factor.

1.2 Related work

The competitive analysis of the throughput of packet switches and packet networks has received increasing attention in recent years. In the context of a single switch, the work of Aiello et al. [Aiello et al. 2005] gave the first competitive analysis of a number of protocols to maximize the weighted throughput of a single non-preemptive FIFO buffer. A number of works then followed, extending the model and the results, e.g. to the preemptive case (e.g., [Kesselman et al. 2001; Lotker and Patt-Shamir 2002]), to the multiple-value case (e.g., [Andelman et al. 2003]), or to models of switches (e.g., [Kesselman and Rosén 2006; Azar and Richter 2004a]). For a short survey see [Epstein and van Stee 2004].

Aiello et al. [Aiello et al. 2003] then extended this approach to whole networks and defined the Competitive Network Throughout (CNT) model. This model was defined for equal-weight packets, analyzing the *number* of packets that reach their destinations, a case which is of interest when whole networks, rather than single switches, are in question. This work showed some greedy protocols, such as Nearest-To-Go (NTG), to be competitive on general networks with a competitive ratio of $O(md)$, and other greedy protocols to be non-competitive at all on general networks (e.g. FTG). Aiello et al. [Aiello et al. 2003] further gave several upper and lower bounds on the competitive ratio of certain greedy protocols on the specific topology of the line. They showed a general lower bound of $\Omega(\sqrt{n})$ for any greedy protocol, and showed that NTG has a competitive ratio of $O(n^{\frac{2}{3}})$, while other greedy protocols, such as FTG and Longest-In-System (LIS) have a lower bound of $\Omega(n)$ on the competitive ratio. Of particular interest in the context of the present work are the results of [Aiello et al. 2003] on general DAGs. [Aiello et al. 2003] showed that on any DAG G, any greedy protocol is $O(h(G))$ competitive, where $h(G)$ is the maximal number of paths ending at one node in the network (including the null path). We note that this bound can be exponential on certain DAGs, and that for any DAG, we prove in the present work better upper bounds both for the weighted and unweighted cases.

Azar and Richter [Azar and Richter 2004b] show an upper bound of $n + 1$ on the competitive ratio of any greedy protocol on any graph with an in-degree of at most 1 (e.g. lines, rings). They use a weighted packets model which is similar to the CNT model, but differs in that it restricts the order of output from any buffer to be FIFO, both for the protocol being considered, and for any optimal protocol. The definition of a network in the model used by Azar and Richter implies that the $n + 1$ upper bound which they show is equivalent to an upper of n in our model. In the same paper, Azar and Richter introduce the "zero-one principle" for switching networks that may simplify the analysis of certain network protocols that deal with weighted packets. We make use of this principle in the proof of one of our upper bounds. Kesselman et al. [Kesselman et al. 2003] consider the case of trees, and give several results in a model which is related to (but different than) our model. Centralized algorithms for the unweighted case are considered in [Azar and Zachut 2005; Angelov et al. 2005], where it is shown that centralized algorithms with polylogarithmic competitive ratios exist for the topology of the line.

Organization In section 2 we formally define our model. In section 3 we prove our two upper bounds for any greedy protocol on DAGs, and in section 4 we give

our two lower bounds. We discuss some conclusions in section 5.

2. THE MODEL

We model a packet network as a directed graph $G = (V, E)$, $|V| = n$, $|E| = m$, where nodes represent routers and edges represent unidirectional communication links. We denote by d the length of the longest simple path in G. The system is synchronous. Time proceeds in discrete steps. All links have unit capacity: in each time step, each link can transmit one packet in the direction it is oriented. Each directed link is identified with an output port in the router at the tail of that link and with an input port in the router at the head of that link. At the output port, there is a buffer of size $B > 0$ which can store up to B packets. At the input port, there is a buffer of size one. All buffers are initially empty. In addition to the ports associated with network links, each node may also have several traffic input ports. The number of traffic input port per node is not restricted. When packets are injected into the network, they are injected into the traffic input ports, where each traffic input port has an input buffer of size one. Packets are injected into the network over time. Each packet is injected into a traffic input buffer of a source node and is labeled with a given destination, together with a prescribed simple path that it has to follow from its source to its destination. Each packet p has also a weight, denoted $w(p)$, where $w(p)$ may take any real value satisfying $1 \leq w(p) \leq \alpha$ for some parameter of the model α. Packets travel in the network in a store-and-forward manner. Each time step is divided into two sub-steps: the *forwarding sub-step* and the *switching sub-step*. In the forwarding sub-step, for each link, a packet may be selected from the output buffer at the tail of the link. Then the selected packet is forwarded to the input buffer at the head of the link. This sub-step also includes the injection of at most one packet into each traffic input buffer. In the switching sub-step, within each router, all the link and traffic input buffers are emptied of their packets. The packets from the input buffers are either transferred to the output buffers of the next link on their path or dropped, or simply output if the node is the packet's destination. In the latter case we say that the packet is *delivered*. The dropped packets may be packets from input buffers, or packets already in the output buffer (i.e. preemption is allowed). When more than B packets are to be placed in a certain buffer then some packets must be dropped, and it is possible to drop packets even if the buffer does not overflow.

We are interested in algorithms that are online and local-control. These algorithms, henceforth called protocols, operate as a set of separate algorithms, one for each node. They make their decision in each time step based only on the information available in that node at that time step (without full view of the network, or knowledge of future arrival of packets). A protocol is *greedy* if it never drops a packet when buffer space is available, and it never keeps a link idle when there is a packet to be forwarded. We extend the definition of greediness into the weighted packet model by stating that a greedy protocol always favors packets of higher weight over packets of lower weight when deciding which packets are to be stored in buffers, and which are first forwarded over a link. Our generic greedy protocol for the weighted case is formally defined in the next section.

In our model we are interested in maximizing the *total weight* of the packets

that reach their destinations, rather than being dropped. This is an extension of the Competitive Network Throughput (CNT) model introduced in [Aiello et al. 2003] to the case of weighted packets. We use the competitive ratio of the *weighted throughput* as a measure for analyzing the protocol. We say that the protocol A is c-competitive if, for *every* sequence of packets injected into the network σ, it holds that, $OPT(\sigma) \leq c \cdot A(\sigma) + \beta$, where $A(\sigma)$ is the total weight of the packets that A delivers (rather than drops), $OPT(\sigma)$ is the total weight of packets that an optimal (centralized, clairvoyant) algorithm delivers, and β is a constant independent of the input traffic σ.

3. UPPER BOUNDS FOR GREEDY PROTOCOLS ON DAGS

In this section we prove that all greedy protocols have finite competitive ratios on all DAGs, and give two independent upper bounds on their competitive ratio. Both upper bounds are independent of the range of packet weights, and of the number of distinct packet weights. We define a generic greedy protocol, that gives preference to the packets according to their weights. That is, the packets are given relative priority according to their weights, when ties may be broken in an arbitrary way. The highest priority packet in each link output buffer is sent over the link. Then, for each link output buffer, the highest priority packets requiring that link are placed in the buffer, and the rest are discarded. In what follows we prove upper bounds on the competitive ratio of any protocol that adheres to this generic framework. More formally, the generic protocol, which we call MAXW, is defined in Figure 1.

For each node v, edge e emanating from v, and each time step t, do the following:

. Forwarding sub-step:
Let P be the set of packets in the buffer at the tail of e in the beginning of the forwarding sub-step of t. If P is not empty, choose a packet from P with maximal weight and forward it over e (ties are broken arbitrarily).

. Switching sub-step:
Let A be the set of packets at node v requiring e at the beginning of the switching sub-step of t. If A is not empty, place in the buffer of e a subset of A having maximal total weight, while not exceeding the buffer size B (If more than one such set exists, the set can be chosen arbitrarily.)

Fig. 1. Generic Protocol $MAXW$

3.1 A $4m(1 + \frac{d}{B})$ Upper Bound

In this section we prove the following theorem.

THEOREM 1. *For any DAG G, the protocol MAXW is $4m(1+\frac{d}{B})$ - competitive.*

COROLLARY 2. *For the unweighted packets case, any greedy protocol is $4m(1+\frac{d}{B})$ - competitive.*

We note that similarly to the proof we show for Theorem 10, it is possible to slightly simplify the proof of Theorem 1 using the zero-one principle for switching networks [Azar and Richter 2004b]. For completeness, we give below a proof of Theorem 1 that does not use the zero-one principle.

Proof of Theorem 1: We start by providing a technical claim, that shows that if packets are injected into the network, then MAXW delivers a set of packets of "high" weight "quickly".

LEMMA 3. *Let G be an arbitrary DAG, $T_1 \leq T_2$ time units, and $k \leq B$ an integer. If a set Q of k packets, $Q = \{q_1, q_2, \ldots, q_k\}$, with a total weight W is injected into G in the time interval $[T_1, T_2]$, then there exists a (possibly other) set of k packets with a total weight of at least W, which MAXW delivers in time frame $[T_1, T_2 + d + k - 1]$.*

PROOF. We first give an outline of the proof. For the proof we use a mechanism of virtual tokens. For each packet $q \in Q$, our mechanism of virtual tokens will (virtually) inject into the network, along with q, a token with the same weight. The purpose of this token is to "remember" the weight of q and show that there is a distinct packet that MAXW delivers by $T_2 + d + k - 1$, which weighs at least as the token, i.e. at least as q. According to the mechanism that we describe below, the token is piggybacked throughout the network on real packets, weighing at least as the token. It is handed over from one real packet to another, until it is absorbed (virtually delivered) when a real packet carrying it is delivered by MAXW. A real packet carries at most one token when it is forwarded by MAXW on an edge. The first main point that will be established is that a token is always carried on packets which weigh at least as the token. The second main point will be that all the virtual tokens are virtually delivered by time $T_2 + d + k - 1$.

We proceed by defining the mechanism of tokens in the network. For the usage of this mechanism, at the tail of each edge we have an additional virtual buffer: a *token buffer*. We start by describing how tokens are created. When packet $q_i \in Q$ is injected into the network, we create a token s_i, assign it a weight w_i equal to the weight of packet q_i, and place it in the token buffer at the tail of the first edge on the path of q_i. Note that exactly k virtual tokens are created, and they are all created in the time interval $[T_1, T_2]$.

Next we define a prioritization on the k tokens. For each $1 \leq i, j \leq k$ where $i \neq j$, token s_i has a higher priority than token s_j if and only if $w(s_i) > w(s_j)$ or $(w(s_i) = w(s_j)$ and $i > j)$. This prioritization remains constant over time, since the weights of the k tokens do not change. We assign a distinct priority value between 1 and k for each token. A token is assigned a priority value i if there are $i - 1$ tokens with lower priority.

Now we describe how tokens are handled. When MAXW forwards some packet p on an edge e, if the token buffer at the tail of e is not empty, the token s with the highest priority in the token buffer is piggybacked on p. When a token s arrives at a node v piggybacked on packet p, if v is the destination of p, we say that s is *virtually delivered*, and it disappears from the network. Otherwise, we place token s in the token buffer at the tail of the next edge on the path of p.

We proceed by proving that a token is always piggybacked on packets of equal or greater weight, and that if a token buffer is non-empty, there is always an available real packet to piggyback a token out of this token buffer.

LEMMA 4. *The following two conditions hold for any time step t such that $t \geq T_1$:*

(1) *If token s is piggybacked on packet p in the forwarding sub-step of t, then $w(s) \leq w(p)$.*

(2) *Let e be an edge in the network such that the token buffer of e is not empty at the end of t, and let $S = \{s_1, s_2, \ldots, s_h\}$ be the set of tokens in this token buffer. Then, there exists a set of h packets $P = \{p_1, p_2, \ldots, p_h\}$ (and possibly more packets) in the buffer at the tail of e, such that for each j, $1 \leq j \leq h$, $w(s_j) \leq w(p_j)$.*

PROOF. By induction on time. We start with the base case $t = T_1$:

Part 1 of the claim holds trivially, since there are no tokens in any token buffer at the beginning of the forwarding sub-step of T_1, and so no token is piggybacked in T_1.

For part 2, assume there are tokens in the network at the end of time T_1. Let v be a node in the network, and let e be an edge emanating from v, such that the token buffer at the tail of e is not empty at the end of time T_1. Let $S = \{s_1, s_2, \ldots, s_h\}$ be this set of tokens. Since S is a subset of all k tokens that are ever created in the network, we have $h \leq k \leq B$. The token creation mechanism implies that all tokens are created in the time interval $[T_1, T_2]$, and so all of these h tokens must have been created in T_1, due to injection of packets into v. At the beginning of the switching sub-step of T_1, for each of the tokens in S there is a distinct packet at a traffic input port of node v, the one that caused its creation, with the same weight as the token, and requiring edge e. The claim clearly follows due to the greedy property of MAXW which maximizes the total weight of the packets chosen to be placed in the buffer at the tail of e. MAXW will place in the buffer at the tail of e a set of h packets $P = \{p_1, p_2, \ldots, p_h\}$ (and possibly more packets) such that for each j, $1 \leq j \leq h$, $w(s_j) \leq w(p_j)$.

We proceed with the proof for $t > T_1$:

For part 1 of the claim, assume token s is piggybacked on packet p over an edge e in the forwarding sub-step of t. By part 2 of the induction hypothesis, at the end of time $t-1$, there exists a packet p' in the buffer at the tail of e such that $w(s) \leq w(p')$. Since MAXW chooses to forward p on e, we conclude that $w(p') \leq w(p)$, and therefore $w(s) \leq w(p)$.

For part 2, let v be a node in the network and let e be an edge emanating from node v, such that the token buffer of e is not empty at the end of time step t. Let $S = \{s_1, s_2, \ldots, s_h\}$ be this set of tokens. S is a subset of all k tokens that are created in the network. Therefore, it holds that $h \leq k \leq B$. Each of the tokens in S was either present in the token buffer of e at the end of $t-1$, created in time t, or was piggybacked on a packet to node v in the forwarding sub-step of t.

Let S_1 be the set of tokens that were present in the token buffer of e at the end of $t-1$. If S_1 is not empty, then by part 2 of the induction hypothesis, each token in S_1 has, at the end of $t-1$, a distinct packet in the buffer at the tail of e weighing at least as the token. Let s_1 be the token with the highest priority in S_1, which implies that s_1 has maximal weight in S_1. MAXW chooses a packet p_1 with maximal weight and sends it over e in the forwarding sub-step of t. Token s_1 is piggybacked on p_1, and it holds that $w(s_1) \leq w(p_1)$. Token s_1 has maximal weight in S_1 and p_1 has maximal weight in the buffer. Therefore, at the beginning of the switching sub-step of t, each of the remaining tokens in $S_1 \setminus \{s_1\}$ still has a distinct

packet at node v of equal or greater weight, requiring edge e.

Let S_2 be the set of tokens in the token buffer of e that were created in the forwarding sub-step of t. At the beginning of the switching sub-step of t, each of the tokens in S_2 has a distinct packet at a traffic input port of node v, i.e., the packet that caused its creation, with the same weight as the token, and requiring edge e.

Let S_3 be the set of tokens in the token buffer of e that arrived to node v piggybacked on packets in the forwarding sub-step of t. By part 1 of the induction hypothesis, each token was piggybacked on a packet weighing at least as the token. Therefore, at the beginning of the switching sub-step of t, each of the tokens in S_3 has a distinct packet at node v, i.e., the packet that carried it to node v, weighing at least as the token and requiring edge e.

All tokens in the set S belong to $S_1 \setminus \{s_1\}$, S_2, or S_3. Therefore, each token $s \in S$ has, at the beginning of the switching sub-step of t, a distinct packet p satisfying $w(s) \leq w(p)$, at node v, and requiring edge e. The claim clearly follows due to the greedy property of MAXW which maximizes the total weight of the packets chosen to be placed in the buffer at the tail of e. Because $h \leq B$, MAXW will place in the buffer at the tail of e a set of h packets $P = \{p_1, p_2, \ldots p_h\}$ (and possibly more packets) such that for each j, $1 \leq j \leq h$, $w(s_j) \leq w(p_j)$. □

We continue by bounding the time it takes MAXW to "deliver" the k tokens created in the time interval $[T_1, T_2]$. We prove the following claim:

Claim 5. *Each token in the network is eventually virtually delivered.*

Proof. By induction on the tokens, ordered by decreasing priority. We start with the token s with priority k - the highest priority. Part 2 of Lemma 4 implies that in each time unit in which s is present in the network at a token buffer at the tail of some edge e, the packet buffer at the tail of e is not empty. Since s has the highest priority of all tokens, and since MAXW always sends a real packet if it can do so, s is piggybacked on a real packet in every time step it is present in the network. Since the network is a DAG, the path which s traverses is acyclic, and so s is virtually delivered within at most d time steps from the time it is created.

We continue with the proof for token s with a priority i, $i < k$. Let t be the time step in which the last token with a priority greater than i is virtually delivered. Assume s is still in the network at the end of t. From $t + 1$ and onward, s is the token with the highest priority in the network. Therefore it is piggybacked in each time step starting from $t + 1$ as long as it is in the network. Since the network is a DAG, the path which s traverses is a-cyclic, and so s is virtually delivered by $t + d$. □

The arguments of the above proof also show that all tokens are delivered by time $T_2 + k \cdot d$. However, we need a better upper bound on the time all tokens are (virtually) delivered in order to prove Lemma 3. In the course of proving this better upper bound, we use the fact that all tokens are delivered in finite time, as proved in Claim 5.

We now proceed with the proof of Lemma 3. By the second part of Lemma 4, if a token buffer is non-empty, there is an available real packet to piggyback a token out of this token buffer. Therefore, and by the greediness of MAXW, if a token s

is not piggybacked at some time step t, this is because another token, s', with a higher priority, was piggybacked instead of s out of the same token buffer. We refer to this event as s being *blocked* by s' at time t. Note that in each time step, every token in the network is either piggybacked on a real packet or blocked by another token. A token which is piggybacked on a real packet at a given time blocks all other tokens in the token buffer it was taken out of.

We now give the required upper bound on the delivery times of all tokens.

LEMMA 6. *All k tokens are virtually delivered by time $T_2 + d + k - 1$.*

PROOF. Let s^* be an arbitrary token in the network. We construct a series of sets $A_1, A_2, \ldots, A_j$ which will satisfy for each i, $A_i \subseteq A_{i+1}$. Each set A_i has i elements. The elements in these sets are events of tokens being blocked at nodes. We will analyze this series of sets to prove the upper bound on the delivery time of token s^*. The elements of these sets are tuples $< s, t, v, x, y >$ where s is a token, t is a time step in which a blocking of s occurred, v is the node at which this event occurred, x is a path emanating from v, and y is the length (in edges) of x. We will show that the series of sets is finite. The blocking events (i.e. elements) in the set with the maximum index will be used to prove that token s^* is virtually delivered by time $T_2 + d + k - 1$.

For the definition of A_1 we consider two cases: If s^* is never blocked then we define t_1 to be the creation time of s^*. v_1 is defined as the node in which s^* is created, x_1 is defined as the path which s^* traverses until its virtual delivery, and y_1 is defined as the length of x_1. We define $A_1 = \{< s_1 = s^*, t_1, v_1, x_1, y_1 >\}$ to be last in this series of sets. Otherwise, s^* is blocked at least once. We know by Claim 5 that s^* is eventually virtually delivered, and therefore blocked a finite number of times. Let t_1 be the latest time unit in which s^* is blocked, and let v_1 be the node at which this event happens. Let x_1 denote the path which s^* travels from v_1 after it is blocked in t_1 until it is virtually delivered, and let y_1 denote the length of x_1. We define $A_1 = \{< s_1 = s^*, t_1, v_1, x_1, y_1 >\}$ and continue the process by defining A_2.

We inductively construct the set A_{i+1} from A_i, for $i \geq 1$:

Given $< s_i, t_i, v_i, x_i, y_i > \in A_i$, s_i is blocked at time t_i in node v_i. Let s_{i+1} be the token which blocks s_i in time t_i. We consider two cases: If s_{i+1} is not blocked from its creation time until it blocks s_i at time step t_i, we define t_{i+1} to be the creation time of s_{i+1}. v_{i+1} is defined as the node at which s_{i+1} is created, x_{i+1} is the path that s_{i+1} travels from its creation to v_i where it blocks s_i (it may be an empty path), and y_{i+1} is the length of the path x_{i+1}. We define $A_{i+1} = A_i \cup \{< s_{i+1}, t_{i+1}, v_{i+1}, x_{i+1}, y_{i+1} >\}$ and stop constructing the series of sets. Otherwise, s_{i+1} is blocked at some node before t_i. We look at the last time step before t_i in which s_{i+1} is blocked. We denote this time step as t_{i+1}. Let v_{i+1} be the node at which s_{i+1} is blocked in time t_{i+1}. Let x_{i+1} be the path that s_{i+1} traveled between v_{i+1} and v_i (it may be an empty path). Let y_{i+1} be the length of path x_{i+1}. We define $A_{i+1} = A_i \cup \{< s_{i+1}, t_{i+1}, v_{i+1}, x_{i+1}, y_{i+1} >\}$ and continue the inductive process by defining the set A_{i+2}.

We now observe the following facts which lead us up to the conclusion of the proof. For each i, s_{i+1} blocks s_i, and so the priority of s_{i+1} is greater than the priority of s_i. Therefore, the series of sets is finite, and j, the maximum index in

the series, satisfies $j \leq k$, since there are k tokens in the network. A_j has at most k blocking events since one blocking event is generated in each step of the inductive process used to construct the series of sets. t_j is the creation time of token s_j. Therefore it holds that $t_j \leq T_2$, since all tokens are created in the time interval $[T_1, T_2]$. For each i, if A_i and A_{i+1} are defined, then the last node of x_{i+1} is the first node of x_i. This is the node in which s_{i+1} blocks s_i at time t_i. Therefore, we can concatenate the paths in A_j to create a path in the network: $x_j \circ x_{j-1} \circ \cdots \circ x_1$. The network is a DAG, and so the length of this path is bounded by d, meaning that $\sum_{i=1}^{j} y_i \leq d$. For each i, s_i is piggybacked on path x_i continuously starting from t_i+1, without being blocked, for y_i consecutive time steps. Thus, the delivery time of s^* is $t_1 + y_1$, where we take into account the last time step in which s^* is blocked, after which it is continuously piggybacked until it is virtually delivered. Moreover, for each $i > 1$, if A_i is defined, then $t_i + y_i + 1 = t_{i-1}$. We thus have that the delivery time of s^* is:

$$\begin{aligned} D_1 &= t_j + [y_j + 1] + [y_{j-1} + 1] + ... + [y_2 + 1] + y_1 \\ &= t_j + j - 1 + \sum_{i=1}^{j} y_i \\ &\leq T_2 + k - 1 + d \ . \end{aligned}$$

This completes the proof of Lemma 6. □

Lemma 3 is an immediate corollary of Lemma 6 and part 1 of Lemma 4. All tokens are virtually delivered by $T_2 + k + d - 1$, and each token is virtually delivered piggybacked on a distinct real packet which MAXW delivers. The total weight of the k packets which piggyback the tokens and virtually deliver them is at least the total weight of the k original packets of Q. This completes the proof of Lemma 3.

We now proceed with the proof of Theorem 1, giving an upper bound of $4m(1+\frac{d}{B})$ on the competitive ratio of MAXW against any adversary on any DAG. We divide the time axis into *frames* of length $d + B$ time units. Frame j, $j \geq 1$, is defined as $[(j-1)(d+B), j(d+B))$. We denote a packet eventually delivered by the adversary as a *D-packet*. We define the following values for a given frame j:

—a_j is the number of *D-packets* injected into the network in time frame j.
—b_j is the number of packets delivered by MAXW in time frame j.
—c_j is the total weight of the a_j *D-packets* injected into the network in time frame j.
—d_j is the total weight of the packets delivered by MAXW in time frame j.
—$e_j = \min\{a_j, B\}$.
—f_j is the total weight of the e_j heaviest *D-packets* injected in time frame j.

We now give a lower bound on the total weight of packets delivered by MAXW in 2 consecutive time frames.

Claim 7. *For each j, $d_j + d_{j+1} \geq f_j$.*

Proof. Let $p_1, p_2, \ldots, p_{e_j}$ be the e_j heaviest *D-packets* injected in time frame j (we chose an arbitrary set if this set is not unique). By definition, $f_j = w(p_1) +$

$w(p_2) + \ldots + w(p_{e_j})$. By definition of e_j, it holds that $e_j \leq B$. Since all these e_j packets were injected in time frame j, by Lemma 3, MAXW delivers (at least) e_j packets with a total weight of at least f_j within the two time frames j and $j+1$ (since each time frame is $d+B$ time units long). □

We now show an upper bound on the ratio between c_j and f_j.

CLAIM 8. *For each j,* $\frac{2m(d+B)}{B} \cdot f_j \geq c_j$.

PROOF. There are two cases:

(1) $a_j \leq B$: By definition, we have $e_j = a_j$. Therefore, we get $f_j = c_j$ by definition of f_j. In addition, we note that $\frac{2m(d+B)}{B} \geq 1$. Multiplying each side of the inequality by f_j, we get $\frac{2m(d+B)}{B} \cdot f_j \geq f_j = c_j$.

(2) $a_j > B$: By definition, we have $e_j = B$. First observe that for any j, $a_j \leq mB + m(d+B) \leq 2m(d+B)$. This is because any *D-packet* injected cannot be dropped by the adversary. The number of packets injected but not dropped in a time interval of length T time units is at most $mB + mT$ (since at most mT packets may be delivered and at most mB packets may be stored in the buffers at the end of the time frame).
The total weight of the B heaviest *D-packets* injected in frame j is f_j. Therefore, their average weight is $\frac{f_j}{B}$. The total weight of all a_j *D-packets* injected in frame j is c_j. Therefore, their average weight is $\frac{c_j}{a_j}$. Since the average weight of the B heaviest *D-packets* injected in frame j is greater or equal to the average weight of all a_j *D-packets*, we get $\frac{f_j}{B} \geq \frac{c_j}{a_j}$. Since $a_j \leq 2m(d+B)$, we get $\frac{f_j}{B} \geq \frac{c_j}{2m(d+B)}$ which is our claim.

□

Now, applying Claims 7 and 8 we have:

$$\begin{aligned} & \frac{2m(d+B)}{B} \cdot (d_j + d_{j+1}) \\ \geq\ & \frac{2m(d+B)}{B} \cdot f_j \\ \geq\ & c_j. \end{aligned}$$

We can now conclude the proof of Theorem 1. For any time t, we bound the total weight of the *D-packets* injected by the adversary by time t, in terms of the total weight of packets delivered by MAXW by time t. Let s be such that $(s-1)(d+B) \leq t < s(d+B)$. First observe that $2 \cdot \sum_{j=1}^{s} d_j \geq \sum_{j=1}^{s-1}(d_j + d_{j+1}) \geq \frac{B}{2m(d+B)} \cdot \sum_{j=1}^{s-1} c_j$. In addition, $ADV(t) \leq \sum_{j=1}^{s-1} c_j + 2m(d+B)\alpha$,[1] and $MAXW(t) \geq \sum_{j=1}^{s} d_j - 2m(d+B)\alpha$, since in a single frame any protocol can deliver packets with a total weight of at most $m(d+B)\alpha$.

[1]Recall that for every packet p, its weight $w(p)$ satisfies $1 \leq w(p) \leq \alpha$.

By applying the above inequalities, we get:

$$\begin{aligned} MAXW(t) &\geq \sum_{j=1}^{s} d_j - m(d+B)\alpha \\ &\geq \frac{B}{4m(d+B)} \sum_{j=1}^{s-1} c_j - m(d+B)\alpha \\ &\geq \frac{B}{4m(d+B)}(ADV(t) - 2m(d+B)\alpha) - m(d+B)\alpha \\ &= \frac{B}{4m(d+B)} ADV(t) - O(m(d+B)\alpha). \end{aligned}$$

□

3.2 An Upper Bound of at Most md in Graph Structure Terms

We now give another upper bound for MAXW on DAGs. This upper bound is defined for any DAG G in terms of the structure of the graph, more specifically, in terms of the function $f(G)$, defined below. We note that this function is bounded by md for any DAG, and by m for layered DAGs such as trees, lines, and butterflies.

DEFINITION 9. *For every two edges $e, e' \in E$ (not necessarily distinct), and integer k such that $1 \leq k \leq d$, we define $f'(e', k, e)$ to be 1 if there exists a path of length k edges whose first edge is e' and last edge is e, and 0 otherwise. We define for any $e \in E$:*

$$f(e) \triangleq \sum_{e' \in E} \sum_{k=1}^{d} f'(e', k, e).$$

For the graph G we define:

$$f(G) \triangleq \max_{e \in E} f(e).$$

For an edge e, the value $f(e)$ sums over all edges e' the number of distinct *lengths* of paths that are in the network, which start with e' and end with e. We will prove the following theorem:

THEOREM 10. *For any DAG G the protocol MAXW is $f(G)$-competitive.*

COROLLARY 11. *For the unweighted packets model, any greedy protocol on DAG G is $f(G)$ - competitive.*

In our proof we will show that MAXW is a *comparison based* protocol, a term defined by Azar and Richter in [Azar and Richter 2004b]. This property will enable us to use the zero-one principle for switching networks (also defined in [Azar and Richter 2004b]) in order to limit the analysis in our proof to packets of weights zero and one. Informally, a protocol is comparison based if it bases its decisions only on the relative order between the weights of packets, and may break ties either arbitrarily or according to a certain policy. Formally, we have the following definition:

DEFINITION 12. *[Azar and Richter 2004b] Given a function $g : \Re \to \Re$ we denote by $g(\sigma)$, for a packet sequence σ, the packet sequence σ with the modified weight function $g(w(p))$ for each packet $p \in \sigma$. For a given algorithm A we denote by $\mathcal{A}(\sigma)$ the set of possible outputs of A on σ, that is, a set of (possible) sequences of packets delivered by A on input σ.* [2] *A is called* comparison-based *if and only if for every monotonically increasing function $g : \Re \to \Re$ we have $\mathcal{A}(\sigma) \subseteq \mathcal{A}(g(\sigma))$, for every σ.*

The zero-one principle is defined in [Azar and Richter 2004b] as follows: Let A be a comparison-based switching algorithm. A is a c-approximation algorithm if and only if A achieves c-approximation for all packet sequences whose values are restricted to be in$\{0, 1\}$,

We continue with the following claim:

CLAIM 13. *$MAXW$ is a comparison-based protocol.*

PROOF. Let $g : \Re \to \Re$ be a monotonically increasing function, and σ an input packet sequence. Let $MAXW^*(\sigma)$ be a possible output of $MAXW$ on the input sequence σ (i.e., a (possible) set of packets which $MAXW$ delivers to their destinations). The fact that g is monotonically increasing ensures that the relative order between the weights of any two packets in σ also holds in $g(\sigma)$. Therefore, any (possible) contention-resolution and scheduling decision made by $MAXW$ for σ (e.g., choosing which packets to store at the buffers, and which packets to forward) is a valid decision for $g(\sigma)$ as well. Thus, there exists an execution of $MAXW$ on $g(\sigma)$ which makes the same decisions, and therefore has the same output sequence $MAXW^*(\sigma)$. □

Proof of Theorem 10: Since MAXW is a comparison-based protocol, we can use the zero-one principle in order to prove an upper bound on the competitive ratio of MAXW. It suffices to prove the competitive ratio for packet sequences with weights restricted to 0's and 1's, which we refer to as 0-packets and 1-packets.

Let $MAXW(t)$ be the total weight of packets delivered by MAXW by time t, and $ADV(t)$ be the total weight of packets delivered by the adversary by time t. We prove that for any t,

$$ADV(t) \leq f(G) \cdot MAXW(t) + m(B + f(G)).$$

For the purpose of the proof we define virtual packets and a mechanism which can "virtually deliver" them, an extension and improvement on the virtual packet mechanism introduced in [Aiello et al. 2003]. These virtual packets cross edges piggybacked on real packets that MAXW transmits and are stored in two types of special buffers, maintained at the tail of every edge: a *virtual holding buffer* and a *virtual transit buffer*. We first describe how to handle virtual packets already in the system, and then how virtual packets are created. When virtual packets arrive at node v piggybacked on a packet p, then if v is the destination of p the virtual packets are "virtually delivered". Otherwise let e be the next edge on the path of p. Then the virtual packets are placed in the virtual transit buffer of e. When a

[2] Note that this is a set of output sequences if algorithm A has some freedom in its definition, e.g., "can break ties arbitrarily".

(real) packet p leaves some node v on edge e, then all the packets in the virtual transit buffer of e are piggybacked on p and the virtual transit buffer is emptied. In addition, if the virtual holding buffer of e is not empty, a virtual packet is extracted from the virtual holding buffer of e, and is piggybacked on p as well.

In the following we call a 1-packet that the adversary eventually delivers a D-packet. The throughput of the adversary up to a given time step is the number of 1-packets it delivers to their destination, i.e. the number of delivered D-packets. Therefore, we assume w.l.o.g. that the adversary accepts into the network only D-packets which it eventually delivers and drops all other packets at their injection time. Note that the adversary may inject 0-packets and 1-packets into the network which it drops at their injection time. Now we describe how virtual packets are created. We create a new virtual packet for each 1-packet that the adversary accepts into the network. This will account for each 1-packet that the adversary eventually delivers. Let p be a 1-packet injected in the forwarding sub-step of time step t. In the switching sub-step of t, if the adversary accepts p into a buffer at the tail of a certain edge e, then a new virtual packet is created and placed in the virtual holding buffer of e.

Consider an edge e and time τ which may be the end of a forwarding sub-step, or the end of a switching sub-step (which is the end of a time step). We denote by $MAXW_1$ and ADV_1 the number of 1-packets stored by MAXW and the adversary at the tail of e at time τ, respectively. We also denote by $HOLDING$ the number of virtual packets in the virtual holding buffer at the tail of e at time τ.

We show a bound on the size of the virtual holding buffers at any given time:

Claim 14. *For any time τ and edge e, the following two conditions hold:*

(*1*) $MAXW_1 \geq HOLDING$.

(*2*) $ADV_1 \geq HOLDING$.

Proof. The claim is proved by induction on time. Initially, the claim clearly holds, since initially $MAXW_1 = ADV1 = HOLDING = 0$, because all buffers are initially empty. We continue with the proof for any τ, assuming the claim holds for any $t < \tau$. Since time steps are divided into forwarding and switching sub-steps, it suffices to prove the claim for τ at the end of a forwarding sub-step, and at the end of a switching sub-step.

We start with the forwarding sub-step. If $HOLDING = 0$ at the beginning of the forwarding sub-step, then $HOLDING = 0$ at the end of the forwarding sub-step, so the claim continues to hold. Otherwise $HOLDING > 0$, and so $MAXW_1 > 0$. Therefore, MAXW forwards a 1-packet over e, which piggybacks a virtual packet from the holding buffer (and possibly some more virtual packets from the transit buffer). $HOLDING$ and $MAXW_1$ decrease by 1, and ADV_1 either remains unchanged or decreases by one. Thus, both parts of the claim continue to hold at the end of the forwarding sub-step.

We now consider the switching sub-step. We note that $MAXW_1$, ADV_1, and $HOLDING$ cannot decrease: $MAXW_1$ cannot decrease because of the greedy property of MAXW; ADV_1 cannot decrease, since we assume w.l.o.g that the adversary drops only newly injected packets; Since no packets are taken out of the virtual holding buffer, $HOLDING$ does not decrease either. Let δ be the number

of D-packets injected by the adversary in the forwarding sub-step which preceded the switching sub-step, requiring edge e. ADV_1 and $HOLDING$ both increase by δ, because a virtual packet is created and placed in the virtual holding buffer for each newly injected D-packet. Therefore, part 1 of the claim continues to hold. Now, by the greediness of MAXW, either $MAXW_1$ grows by δ or more, or the buffer of MAXW is full with 1-packets, i.e., $MAXW_1 = B$. In the former case part 2 of the claim clearly holds. In the latter case we have, using part 1 of the claim already established, that $MAXW_1 = B \geq ADV_1 \geq HOLDING$ which proves part 2. □

COROLLARY 15. *A virtual packet is piggybacked only on 1-packets forwarded by MAXW, and never on 0-packets.*

PROOF. By induction on time steps. For $t = 0$ the claim clearly holds, since at $t = 0$ packets are only injected into the network, and no packets are transmitted. For $t > 0$, let p^* be a virtual packet piggybacked on a real packet p over an edge e emanating from a node v. p^* is taken out of a virtual holding buffer or out of a virtual transit buffer. If p^* is taken out of a virtual holding buffer, then by Claim 14 there is at least one 1-packet in the real buffer at the tail of e in the beginning of time t. Therefore, packet p which MAXW forwards is a 1-packet. Otherwise, p^* is taken out of a transit buffer. This means that it arrived to node v at $t-1$ piggybacked on a real packet q requiring edge e. Since we assume that the claim holds up to time $t-1$, q is a 1-packet. Therefore, at least one 1-packet was placed by MAXW in the real buffer of e in the switching sub-step of $t-1$, so p which is forwarded by MAXW on e in time step t is a 1-packet. □

We proceed by showing that once a virtual packet is taken out of a virtual holding buffer and piggybacked on a real packet, it is continuously piggybacked on real packets in every time step until it is virtually delivered.

CLAIM 16. *with MAXW, if a virtual packet p^* is taken out of a virtual holding buffer and piggybacked on a real packet in time step t, then for each time step $t' \geq t$, if not already virtually delivered, p^* is piggybacked on a real packet.*

PROOF. By induction on time steps, starting from t. For $t' = t$, p^* is piggybacked on a real packet and so the claim holds. For $t' > t$, if p^* is not virtually delivered by the end of $t'-1$, then according to the induction hypothesis it arrived to a certain node v in $t'-1$ piggybacked on a real packet p whose destination node is not v. In the switching sub-step of $t'-1$ MAXW places at least one packet in the buffer at the tail of the next edge e on the path of p, and p^* is added to the virtual transit buffer at the tail of e. Therefore, in t', MAXW forwards a real packet on e, and this packet piggybacks all packets in the virtual transit buffer, including p^*. □

The next claim bounds from above the number of virtual packets that are piggybacked on a single real packet in a single time step.

CLAIM 17. *The number of virtual packets that are piggybacked on a single real packet in a given time step t is at most $f(G)$.*

PROOF. Consider P^*, the set of virtual packets piggybacked on a real packet in a certain time step t over an edge e. For each virtual packet $p^* \in P^*$ there is a

time t' when it was initially taken out of a virtual holding buffer, and the edge e' of this virtual holding buffer. By Claim 16, once a virtual packet is taken out of a virtual holding buffer, it is continuously forwarded until it is virtually delivered. Therefore, the length of the path on which p^* travels up to (and including) time step t is exactly $t - t' + 1$. This implies that there is a path with $t - t' + 1$ edges whose first edge is e' and last edge is e. Moreover, since only one virtual packet is taken out of any holding buffer at any given time step, p^* is the only packet in P^* which was taken out of the virtual holding buffer at the tail of e' at time t'. Thus, we can associate each packet in P^* with a distinct pair (e', t') which satisfies the property that there exists a path of length $t - t' + 1$ whose first edge is e' and last edge is e. In order to bound the number of virtual packets in P^* we count the number of pairs (e', t') which satisfy this property. This number is by definition $f(e)$. Therefore, $f(G)$, the maximum value of $f(e)$ in the graph G, is an upper bound for the size of P^* for any time t and any edge e. □

COROLLARY 18. *The size of any transit buffer is bounded by $f(G)$, since all packets in a transit buffer are piggybacked on the next transmitted real packet.*

Now, we can conclude the proof of Theorem 10. The total number of D-packets delivered by the adversary by time t is bounded from above by the total number of virtual packets created by time t. At time t, these packets are either in the network or already virtually delivered. Claims 14 and Corollary 18 bound the size of the holding and transit buffers at any given time, to be B and $f(G)$, respectively. Claim 17 and Corollary 15 bound the ratio between the total number of virtual packets that are virtually delivered by time t and the number of 1-packets delivered by MAXW by time t to be $f(G)$. Thus, we have:

$$ADV(t) \leq f(G) \cdot MAXW(t) + m(B + f(G)).$$

□

4. LOWER BOUNDS FOR GREEDY PROTOCOLS ON DAGS

In this section we show two lower bounds for the competitive ratio of greedy protocols on DAGs. The first lower bound is given for the protocol Furthest-To-Go (FTG). We then give a second, simpler lower bound that applies to any greedy protocol.

Both lower bounds are given using packets of equal weight and they thus also apply to the original non-weighted CNT model [Aiello et al. 2003]. The first lower bound matches up to a constant factor the upper bound of $4m(1 + \frac{d}{B})$ given in Theorem 1. The second lower bound matches the upper bound of $f(G)$, given in Theorem 10, up to a constant factor. Thus, together with the results of the previous section, these bounds give tight (up to constant factors) upper and lower bounds on the competitive ratio of greedy protocols on DAGs, in both the weighted and unweighted cases.

4.1 A Lower Bound of $\Omega(n^2(1 + \frac{n}{B}))$ for FTG

FTG prioritizes packets according to their remaining path lengths, preferring packets with longer remaining paths to packets with shorter remaining paths. The

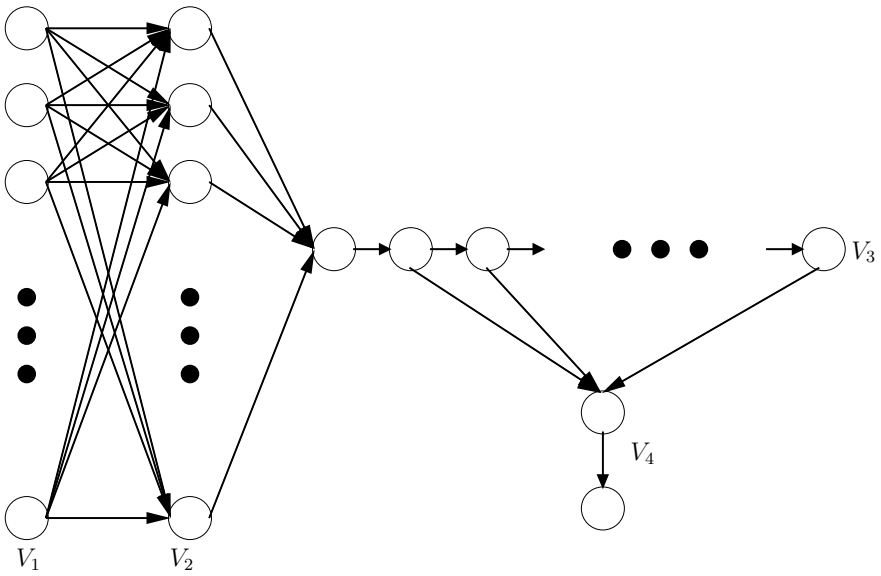

Fig. 2. Graph used in the $\Omega(n^2(1+\frac{n}{B}))$ lower bound for FTG

following lower bound demonstrates how this property undermines the throughput of FTG, in comparison to the throughput of an adversary.

THEOREM 19. *There exist a DAG G such that the competitive ratio of FTG on G is $\Omega(n^2(1+\frac{n}{B}))$.*

PROOF. The DAG on which the lower bound is achieved is constructed as follows (See Figure 2):

Using $n+2$ nodes we define $V = V_1 \cup V_2 \cup V_3 \cup V_4$. We take n which is a multiple of 3. We define:

—$V_1 = \{v_1, v_2, \ldots, v_{\frac{n}{3}}\}$
—$V_2 = \{v_{\frac{n}{3}+1}, v_{\frac{n}{3}+2}, \ldots, v_{\frac{2n}{3}}\}$.
—$V_3 = \{v_{\frac{2n}{3}+1}, v_{\frac{2n}{3}+2}, \ldots, v_n\}$.
—$V_4 = \{v_{n+1}, v_{n+2}\}$.

The set of edges is defined to be $E = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5$ where:

—$E_1 = \{e = (v_i, v_j) \,|\, v_i \in V_1, v_j \in V_2\}$. E_1 creates a complete unidirectional bipartite graph between V_1 and V_2.
—$E_2 = \{e = (v_i, v_{\frac{2n}{3}+1}) \,|\, v_i \in V_2\}$. Note that $v_{\frac{2n}{3}+1}$ is the "first" node in V_3.
—$E_3 = \{e = (v_i, v_{i+1}) \,|\, v_i, v_{i+1} \in V_3\}$.
—$E_4 = \{e = (v_i, v_{n+1}) \,|\, v_i \in V_3 \setminus \{v_{\frac{2n}{3}+1}\}\}$. Note that v_{n+1} is the "first" node in V_4.
—$E_5 = \{e = (v_{n+1}, v_{n+2})\}$.

All packets have equal weight and are injected into the nodes of V_1. At time $t = 0$ the adversary injects:

—For each pair of nodes $v_i \in V_1$ and $v_j \in V_2$:
(1) One packet into v_i with a path of length 1 (v_i, v_j). We refer to these packets as *short* packets.
(2) B packets into v_i with a path of length 5 $(v_i, v_j, v_{\frac{2n}{3}+1}, v_{\frac{2n}{3}+2}, v_{n+1}, v_{n+2})$. We refer to these packets as *medium* packets.

—One packet into v_1 with a path of length $\approx \frac{n}{3}$ $(v_1, v_{\frac{n}{3}+1}, v_{\frac{2n}{3}+1}, v_{\frac{2n}{3}+2}, v_{\frac{2n}{3}+3}, \ldots, v_n, v_{n+1}, v_{n+2})$. We refer to this packet as a *long* packet.

At time steps $t = 1$ through $t = \frac{n}{3} - 3$ the adversary injects a slightly different sequence of packets:

—For each pair of nodes $v_i \in V_1$ and $v_j \in V_2$:
(1) One packet into v_i with a path of (v_i, v_j). These are the *short* packets.
(2) One packet into v_i with a path of $(v_i, v_j, v_{\frac{2n}{3}+1}, v_{\frac{2n}{3}+2}, v_{n+1}, v_{n+2})$. These are the *medium* packets.

—One packet into v_1 with the path $(v_1, v_{\frac{n}{3}+1}, v_{\frac{2n}{3}+1}, v_{\frac{2n}{3}+2}, v_{\frac{2n}{3}+3}, \ldots, v_{n-t}, v_{n+1}, v_{n+2})$. These are the *long* packets.

We analyze how FTG schedules this sequence of packets. The short packets will be dropped at the time of their injection since FTG will favor the medium and long packets over the short packets. Whenever a short packet is injected, there will be (at least) B medium or long packets for FTG to choose from, causing the short packet to be dropped. The situation at time $\frac{n}{3}+1$ will be as follows: (a) None of the packets are delivered yet. (b) For each time $t \in [0, \frac{n}{3} - 3]$, the long packet injected at t will be at node v_{n-t}, with v_{n+1} as its next node. (c) Medium packets that have not been dropped are in one of the nodes of V_1, V_2 or in the first two nodes of V_3 which are $v_{\frac{2n}{3}+1}$ and $v_{\frac{2n}{3}+2}$. Starting from time $\frac{n}{3} + 3$, a packet will cross edge (v_{n+1}, v_{n+2}) for at most $4B + 1$ consecutive time steps. After that, no packet will be in the network. The explanation is as follows: After B steps, no packets will be in any of the nodes of V_1, and no packets will remain in any of the nodes $v_{\frac{2n}{3}+3}$ through v_n. After $2B$ steps no packets will remain in any of the nodes of V_1, V_2, and $[v_{\frac{2n}{3}+3}, v_n]$. At most B packets will be in $v_{\frac{2n}{3}+1}$, at most one packet will be at node $v_{\frac{2n}{3}+2}$, and at most B packets will be in v_{n+1}. After $3B$ steps no packet will be in $v_{\frac{2n}{3}+1}$, V_1, V_2, and $[v_{\frac{2n}{3}+3}, v_n]$. At most one packet will be in $v_{\frac{2n}{3}+2}$ and at most B packets will be in v_{n+1}. After $3B + 1$ steps at most B packets will be in v_{n+1}, and all other nodes are empty. After $4B + 1$ steps no packets will remain in $v_{\frac{2n}{3}+2}$, and all other nodes are empty as well. The throughput of FTG is therefore bounded from above by $4B + 1$ for this injection sequence.

The adversary can drop the following packets upon their injection in each time step $t \in [0, \frac{n}{3} - 3]$: (a) All long packets, and (b) One medium packet at the tail of each edge between V_1 and V_2. All short packets will be forwarded to their destination in V_2 one time step after their injection. At the beginning of time $\frac{n}{3} - 1$, all short packets have been delivered, and there are $B - 1$ medium packets in the buffer at the tail of each of the edges between V_1 and V_2. These buffers will be drained sequentially such that no packet is dropped and all reach their

destination v_{n+2}. The throughput of the adversary is $(\frac{n}{3})^2(\frac{n}{3}-2)$ short packets, plus $(\frac{n}{3})^2(B-1)$ medium packets. In all, FTG delivers at most $3B+3$ packets and the adversary delivers $(\frac{n}{3})^2(\frac{n}{3}-2)+(\frac{n}{3})^2(B-1)$ packets.

This process may be repeated an unlimited number of times. It follows that that there is a lower bound of $\Omega(n^2(1+\frac{n}{B}))$ on the competitive ratio of FTG. □

4.2 An $\Omega(n^2)$ Lower Bound for any Greedy Protocol

We continue by showing a simpler lower bound for any greedy protocol on a DAG. This lower bound illustrates a property of greedy protocols which may flood a bottleneck of the network with packets arriving from different paths. Since no coordination is done between the nodes, a large number of packets with different paths may end up trying to go simultaneously through a bottle neck, forcing any greedy protocol to drop packets. An adversary may choose in such a situation to route the packets sequentially through the bottle neck, and deliver all packets to their destinations.

THEOREM 20. *There exists a DAG G, such that the competitive ratio of any greedy protocol on G is $\Omega(n^2)$.*

PROOF. The network is defined as $G=(V,E)$. Using $n+2$ nodes we define $V=V_1\cup V_2\cup V_3$. We take an even n. We define the graph as follows (See Figure 3):

—$V_1=\{v_1,v_2,\ldots,v_{\frac{n}{2}}\}$
—$V_2=\{v_{\frac{n}{2}+1},v_{\frac{n}{2}+2},\ldots,v_n\}$.
—$V_3=\{v_{n+1},v_{n+2}\}$.

The set of edges is defined to be $E=E_1\cup E_2\cup E_3$ where:

—$E_1=\{e=(v_i,v_j)\,|\,v_i\in V_1,v_j\in V_2\}$. E_1 creates a complete unidirectional bipartite graph between V_1 and V_2.
—$E_2=\{e=(v_i,v_{n+1})\,|\,v_i\in V_2\}$. Note that v_{n+1} is the "first" node in V_3.
—$E_3=\{e=(v_{n+1},v_{n+2})\}$.

At time $t=0$, for each pair of nodes $v_i\in V_1$ and $v_j\in V_2$ the adversary injects B equal-weight packets into node v_i with a path of $(v_i,v_j,v_{n+1},v_{n+2})$. After this, no more packets are injected into the network. Any greedy protocol will forward a packet to its destination on $e=(v_{n+1},v_{n+2})$ starting from time $t=3$ for $3B$ consecutive time steps. After that, no packets will remain in the network. The total number of packets delivered is therefore $3B$. The explanation is as follows: The first packet transmitted on $e=(v_{n+1},v_{n+2})$ is at time $t=3$. After B steps, no packets are in any of the V_1 nodes. Each node in V_2 has at most B packets, and node v_{n+1} has at most B packets as well. After $2B$ steps, no packets remain in any of the nodes of V_1 and V_2, and node v_{n+1} has at most B packets. After $3B$ steps, no packets remain in the system. Therefore, $3B$ packets are delivered to their destination by any greedy protocol. The adversary can drain each buffer, one after the other, so that no packet is dropped. The total number of packets the adversary delivers is $B(\frac{n}{2})^2$.

This process may be repeated an arbitrary number of times. Therefore, we conclude that any greedy protocol has a lower bound of $\Omega(n^2)$ on this DAG. □

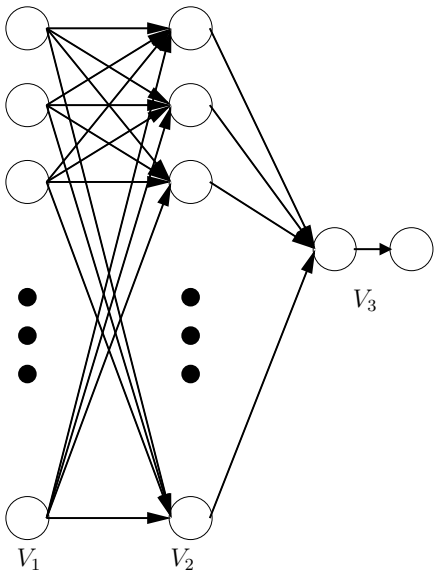

Fig. 3. Graph used in the $\Omega(n^2)$ lower bound for any greedy protocol

5. CONCLUSIONS

In this paper we extend the Competitive Network Throughout (CNT) model introduced in [Aiello et al. 2003] into the weighted packet case, thus modeling Quality of Service considerations and the DiffServ setting. We show that on any DAG, any greedy protocol is competitive, with competitive ratio independent of the weights of the packets. We give two independent upper bounds on the competitive ratio of general greedy protocols on DAGs. We further construct certain DAGs and adversaries that show that our upper bounds cannot be improved (other than constant factors) in the general case. Our results also improve upon the corresponding results given in [Aiello et al. 2003] for the unweighted case, and we thus give tight (up to constant factors) bounds for both the unweighted and weighted cases.

An interesting and important direction for further work is to analyze the performance of *non-greedy* protocols. These may have better performance than greedy ones for both the weighted and the unweighted cases. The question of the existence of a non-greedy protocol that has better performance than greedy protocols is open even for the simple topology of the line, for which a lower bound of $\Omega(\sqrt{n})$ for any greedy protocol is known [Aiello et al. 2003]. *Centralized* online algorithms for the line with a polylogarithmic competitive ratio have been given in [Azar and Zachut 2005; Angelov et al. 2005].

REFERENCES

Aiello, W., Kushilevitz, E., Ostrovsky, R., and Rosén, A. 2003. Dynamic routing on networks with fixed size buffers. In *Proceedings of the 14th annual ACM-SIAM Symposium on Discrete Algorithms*. 771 – 780.

Aiello, W., Mansour, Y., Rajagopolan, S., and Rosén, A. 2005. Competitive queue policies for differentiated services. *Journal of Algorithms 60,* 1, 60–83.

Andelman, N., Mansour, Y., and Zhu, A. 2003. Competitive queueing policies for qos switches. In *Proceedings of the 14th annual ACM-SIAM Symposium on Discrete Algorithms*. 761–770.

Andrews, M., Awerbuch, B., Fernandez, A., and J. Kleinberg, T. Leighton, Z. L. 1996. Universal stability results for greedy contention-resolution protocols. In *37th IEEE Symp. on Foundations of Computer Science*. 380 – 389.

Angelov, S., Khanna, S., and Kunal, K. 2005. The network as a storage device: Dynamic routing with bounded buffers. In *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. 1 – 13.

Azar, Y. and Richter, Y. 2004a. An improved algorithm for cioq switches. In *Proceedings of the 12th annual European Symposium on Algorithms*. 65–76.

Azar, Y. and Richter, Y. 2004b. The zero-one principle for switching networks. In *Proceedings of the 36th annual ACM Symposium on Theory of Computing*. 64 – 71.

Azar, Y. and Zachut, R. 2005. Packet routing and information gathering in lines, rings and trees. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA)*. 484 – 495.

Clark, D. and Wroclawski, J. 1997. An approach to service allocation in the internet. Internet draft, draft-clark-diff-svc-alloc-00.txt.

Epstein, L. and van Stee, R. 2004. Buffer management problems. In ACM SIGACT News.

Kesselman, A., Lotker, Z., Mansour, Y., and Patt-Shamir, B. 2003. Buffer overflows of merging streams. In *Proceedings of the 11th annual European Symposium on Algorithms*. 349–360.

Kesselman, A., Lotker, Z., Mansour, Y., Patt-Shamir, B., Schieber, B., and Sviridenko, M. 2001. Buffer overflow management in qos switches. In *Proceedings of the 33rd annual ACM Symposium on Theory of Computing*. 520–529.

Kesselman, A. and Rosén, A. 2006. Scheduling policies for cioq switches. *Journal of Algorithms 60,* 1, 60–83.

Lotker, Z. and Patt-Shamir, B. 2002. Nearly optimal fifo buffer management for diffserv. In *Proceedings of the 21st annual symposium on Principles of distributed computing*. 134–143.

Received Month Year; revised Month Year; accepted Month Year