Competitive Weighted Throughput Analysis of Greedy Protocols on DAGs

Eyal Gordon†
Dept. of Computer Science
Technion
Haifa 32000, Israel
geyal@cs.technion.ac.il

Adi Rosén†
Dept. of Computer Science
Technion
Haifa 32000, Israel
adiro@cs.technion.ac.il

ABSTRACT
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. [2] 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. We give upper and lower bounds for both the unweighted and weighted cases. We further give tight upper 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 [2] 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 importance and may find additional applications.

Categories and Subject Descriptors
F.2.2 [Analysis of Algorithms and Problem Complexity]: Non-numerical Algorithms and Problems—Routing and layout;
G.2.2 [Discrete Mathematics]: Graph Theory—Network problems

†Part of the work of this author was done at IBM Haifa Labs, Haifa 31905, Israel
†Research supported in part by a grant from the United States - Israel Binational Science Foundation (BSF)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODC ’05, July 17–20, 2005, Las Vegas, Nevada, USA.
Copyright 2005 ACM 1-59593-994-2/05...$5.00.

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. [1] 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. [12, 10, 3, 11, 5] (for a short survey see [8]). 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 [2], 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, proto-
cols 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. [2] 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 [7], require 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. [2] 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 to 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. 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 [2] for the unweighted case, and our lower bounds apply also to the unweighted case. Thus we also improve the results given in [2] 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 [2]: 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 proven for MAXW applies also to any specific protocol that adheres to the greediness criteria.

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 \( |E| = m \), and \( d \) denotes the length of the longest simple path in \( G \)).

- We prove that \( \text{MAXW} \) is \( \frac{4m(d+1)}{B} \) - 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 \( e \in E \): \( f(e) = \sum_{k=1}^{d} f'(e', k, e) \).

For the graph \( G \) we define: \( f(G) = \max_{e \in E} f(e) \).

The first upper 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 \( B \leq \frac{d^2}{d-1} \), 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 larger than \( \Omega(m) \) 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 [2] 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 [2] is exponential on certain DAGs, whereas both upper bounds shown in the present paper 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 \( h(G) \) given in [2].

In the course of proving our first upper bound we prove a lemma (Lemma 5) that gives upper bounds on the delivery times of packets by any greedy protocol on general DAGs (without buffer size considerations). 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 \( t + d + k - 1 \), where \( d \) is the length of longest (simple) path. 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 \( \Omega\left(\frac{n^2}{n+B}\right) \) 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 greedy greedy protocol, our upper bound of \( \frac{4m(d+1)}{B} \) cannot be improved (up to constant factors), and (2) it strengthens conclusions given in [2], showing that although FTG is a stable protocol in the Adversarial Queuing model [4], 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 \( \Omega\left(\frac{m^2}{m+B}\right) \) 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. [1] 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., [10, 12]), to the multiple-value case (e.g., [3]), or to models of switches (e.g., [11, 5]). For a short survey see [8].

Aiello et al. [2] then extended this approach to whole networks and defined the Competitive Network Throughput (CNT) model. This model was defined for equal-weight packets, analyzing the number of packets that reach their destinations. This work showed some greedy protocols to be competitive on general networks (e.g., NTG), and others greedy protocols to be non-competitive at all on general networks (e.g. FTG). [2] further gave several upper and lower bounds on the competitive ratio of certain greedy protocols on the specific topology of the line.

Of particular interest in the context of the present work are the results of [2] on general DAGs. [2] 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. We note that this bound can be exponential on certain DAGs, and that for any DAG, we prove in the present paper better upper bounds both for the weighted and unweighted cases.

Azar and Richter [6] show an upper bound of $n + 1$ on the competitive ratio of any greedy protocol on the line, for weighted packets. Our results for general DAGs improve upon their result, giving for the line the upper bound of $n - 1$. 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. [9] consider the case of trees, and give several results in a model which is related to (but different than) our model.

Organisation 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 give 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 time 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. In addition to the ports associated with network links, each node may also have several traffic input ports. 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. 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 $1 \leq w(p) \leq \alpha$ for some parameter $\alpha$. 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 buffer of the next link on their path or dropped (or simply output if the node is the packet’s destination). When more than $B$ packets are to be placed in a certain buffer then some packets must be dropped. The dropped packets may be packets from input buffers, or packets already in the output buffer (i.e. preemption is allowed).

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 [2] to the case of weighted packets. We use the competitive ratio of the weighted throughput as measure for analyzing the protocol. We say that the protocol $A$ is $c$-competitive if, for every sequence of packets injected into the network $\sigma$, 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 $\beta$ is a constant independent of the input traffic $\sigma$. In the analysis which follows, we assume that an adversary is controlling the input traffic. The competitive analysis is performed against the throughput of the adversary who is capable of achieving an optimal throughput on the input traffic.

3. UPPER BOUNDS FOR GREEDY PROTOCOLS ON DAGS

In this section we prove that all greedy protocols are competitive 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 packets are stored in the buffer, and the rest are discarded. The highest priority packet is then sent over the link. 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.

3.1 Upper Bound in terms of Buffer Size $B$

In this section we prove the following theorem.

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

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”.
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 $e$ in the beginning of the switching sub-step of $t$. If $A$ is not empty, place in the buffer of $v$ 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.)

Figure 1: Generic Algorithm $MAXW$.

**Lemma 2:** 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 start by giving 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 tokens are virtually delivered by $T_2 + d + k - 1$.

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 a packet $q_i \in Q$ is injected into the network, we create a token $r_i$, 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 $r_i$ has a higher priority than token $r_j$ if and only if $w(r_i) > w(r_j)$ or $(w(r_i) = w(r_j)$ 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 $r$ with the highest priority in the token buffer is piggybacked on $p$. When a token $r$ arrives at a node $v$ piggybacked on packet $p$, if $v$ is the destination of $p$, we say that $r$ is virtually delivered, and it disappears from the network. Otherwise, we place token $r$ 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.

**Lemma 3:** 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 $S = \{s_1, s_2, \ldots, s_k\}$ 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 over time. We start with the proof for $t = T_1$:

1. 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 $T_1$.
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 $T_1$. Let $S = \{s_1, s_2, \ldots, s_k\}$ 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 $e$. At the beginning of the switching sub-step of $T_1$, each of the tokens in $S$ has the distinct packet at 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 $t > T_1$:

1. 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 step $t - 1$, 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 $w(s) \leq w(p)$.
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 $S = \{s_1, s_2, \ldots, s_k\}$ 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 $S_t$ 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 substep 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 the distinct packet at node \( v \), the one 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 \( S_2 \) has the distinct packet at node \( v \), the one 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 switching the 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 \( j, 1 \leq j \leq h \), \( w(s_j) \leq w(p_j) \).

\[
\square
\]

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 4:** Each token in the network is eventually virtually delivered.

**Proof Sketch.** By induction on the tokens, ordered by decreasing priority. For the basis of the induction, the token with the highest priority is piggybacked on a packet in each time step from its creation time to its delivery. This is because (a) according to Lemma 3 there is always an available real packet to carry the token, and (b) due to its priority, this token is always the one forwarded from its buffer. Since the network is a DAG, this token is eventually virtually delivered. For the induction step, we consider a token at the time when all higher priority tokens have already been virtually delivered. We can then apply the same arguments to prove that this token is eventually virtually delivered.

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 2. 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 4.

We now proceed with the proof of Lemma 2. By the second part of Lemma 3, if a token buffer is non-empty, there is always 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 5:** All \( k \) tokens are virtually delivered by time \( T_2 + d + k - 1 \).

**Proof.** Let \( s_1 \) be an arbitrary token in the network. We construct a series of sets \( A_1, A_2, \ldots, A_i \) 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 \( s_1 \). 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 \( s_1 \) is virtually delivered by time \( T_2 + d + k - 1 \).

For the definition of \( A_i \) we consider two cases: If \( s_1 \) is never blocked then we define \( t \) to be the creation time of \( s_1 \), \( v_1 \) is defined as the node in which \( s_1 \) is created, \( x_1 \) is defined as the path which \( s_1 \) traverses until its virtual delivery, and \( y_1 \) is defined as the length of \( x_1 \). We define \( A_1 = \{<s_1, t_1, v_1, x_1, y_1>\} \) to be last in this series of sets. Otherwise, \( s_1 \) is blocked at least once. We know by Claim 4 that \( s_1 \) is eventually virtually delivered, and therefore blocked a finite number of times. Let \( t_1 \) be the latest time unit in which \( s_1 \) is blocked, and let \( v_1 \) be the node at which this event happens. Let \( x_1 \) denote the path which \( s_1 \) 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 \( A_1 = \{<s_1, t_1, v_1, x_1, y_1>\} \) 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 of \( x_{i+1} \). 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 \( t_{i+1} \). \( x_{i+1} \) is 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 the path of \( x_{i+1} \). 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 \( j \leq k \). \( A_j \) has at most \( k \) blocking events. \( t_j \) 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
Lemma 2 is an immediate corollary of Lemma 5 and part 1 of Lemma 3. 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 $Q$.

We now proceed with the proof of Theorem 1, giving an upper bound of $\frac{2m(d+B)}{B}$ 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 $j$ is defined as $\{t + (j-1)(d+B), j(d+B)\}$. 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 $c_j$ 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 6:** For each $j$, $d_j + d_{j+1} \geq f_j$.

**Proof.** Let $p_1, p_2, \ldots, p_{c_j}$ be the $c_j$ heaviest $D$-packets injected in time frame $j$ (we chose an arbitrary set if this set is not unique). By definition, $f_j = w(p_1) + w(p_2) + \ldots + w(p_{c_j})$. 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 2, 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 7:** 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 $f_j = e_j$ 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 $\frac{2m(d+B)}{B} \cdot f_j \geq f_j = e_j$.  
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 $\frac{f_j}{B} \geq \frac{2m(d+B)}{2m(d+B)}$, which is our claim.

Now, applying Claims 6 and 7 we have: $\frac{2m(d+B)}{B} \cdot f_j \geq c_j$.

We can now conclude the proof of Theorem 1. For any time $t$, we now 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 $t$. Let $s$ be such that $(s-1)(d+B) \leq t < s(d+B)$. First observe that  
$$\sum_{j=1}^{s} d_j \geq \sum_{j=1}^{s} d_j + d_{j+1} \geq \frac{2m(d+B)}{B} \sum_{j=1}^{s} c_j.$$  
In addition, $ADV(t) \leq \sum_{j=1}^{s-1} c_j + 2m(d+B)\alpha$, 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 $2m(d+B)\alpha$.

By applying the above inequalities, we get:

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

**3.2 Upper Bound in terms of Network Structure**

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, butterflies, etc.

**Definition 8:** For every two edge $e, e' \in E$ (not necessarily distinct), and value $k$ 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 $e \in E$:  
$$f(e) \triangleq \sum_{k=1}^{d} \sum_{e' \in E} f'(e', k, e).$$

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 different lengths of paths that are in the network, which start with $e'$ and end with $e$.

**Theorem 9:** For any DAG $G$ the protocol MAXW 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 [6]. This property will enable us to use the zero-one principle for switching networks (also defined in [6]) 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, the following is the definition of a comparison based protocol:

**Definition 10:** [6] Given a function $f: \mathbb{R} \rightarrow \mathbb{R}$ we denote by $f(\sigma)$ the packet sequence $\sigma$ with the modified weight function $f(w(p))$ for each packet $p \in \sigma$. For a given algorithm $A$ we let $A(\sigma)$ be the set of all possible outputs of $A$ on $\sigma$ (i.e., all possible sets of packets that reach their destinations). Let $A$ be a switching algorithm that may break ties arbitrarily between packets with equal weight. $A$ is called comparison-based if and only if for every monotonically increasing function $f: \mathbb{R} \rightarrow \mathbb{R}$ we have $A(\sigma) \subseteq A(f(\sigma))$.

The zero-one principle is defined in [6] 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 $0/1$.

We start by proving the following:

**Claim 11:** MAXW is a comparison-based protocol.

**Proof.** Let $f: \mathbb{R} \rightarrow \mathbb{R}$ be a monotonically increasing function, and $\sigma$ an input packet sequence. Let $MAXW_1(\sigma)$ be a possible output of $MAXW$ on the input sequence $\sigma$ (i.e., a set of packets which $MAXW$ delivers to their destinations). The monotonically increasing property of $f$ ensures that the relative order between any two packets in $\sigma$ also holds in $f(\sigma)$. Therefore, any contention-resolution and scheduling decision made by $MAXW$ for $\sigma$ is a legal decision for $f(\sigma)$. Thus, there exists an execution of $MAXW$ on $f(\sigma)$ which makes the same decisions and therefore has the same output sequence as $MAXW_1(\sigma)$. Therefore $MAXW_1(\sigma) \subseteq A(f(\sigma))$. □

**Proof of Theorem 9.** 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 [2]. 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 (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 $\tau$ 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 $\tau$, respectively. We also denote by HOLDING the number of virtual packets in the virtual holding buffer at the tail of $e$ at time $\tau$.

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

**Claim 12:** For any time $\tau$ 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 over time. Initially, the claim clearly holds, since initially $MAXW_1 = ADV_1 = HOLDING = 0$. We continue with the proof for any $\tau$, 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 $\tau$ as the end of a forwarding sub-step, and as the end of a switching sub-step.

We start with the forwarding sub-step. If $HOLDING = 0$ in 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 assumed 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 $\delta$...
be the number of D-packets injected by the adversary in the forwarding sub-step which preceded the switching sub-step, requiring edge \( e \). \( \text{ADV}_1 \) and \( \text{HOLDING} \) both increase by \( \delta \), 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 \( \text{MAXW} \), either \( \text{MAXW}_1 \) grows by \( \delta \) or more, or the buffer of \( \text{MAXW} \) is full with 1-packets, i.e., \( \text{MAXW}_1 = B \). 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 \( \text{MAXW}_1 = B \geq \text{ADV}_1 \geq \text{HOLDING} \) which proves part 2.

**Corollary 13:** A virtual packet is piggybacked only on 1-packets forwarded by \( \text{MAXW} \) and never on 0-packets.

**Proof.** By induction over 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 \( p_{\text{virt}} \) be a virtual packet piggybacked on a real packet \( p \) over an edge \( e \) emanating from a node \( v \). \( p_{\text{virt}} \) is taken out of a virtual holding buffer or out of a virtual transit buffer. If \( p_{\text{virt}} \) is taken out of a virtual holding buffer, then by Claim 12 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 \( \text{MAXW} \) forwards is a 1-packet. Otherwise, \( p_{\text{virt}} \) 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 the claim holds up to \( t - 1 \), \( q \) is a 1-packet. Therefore, at least one 1-packet was placed by \( \text{MAXW} \) in the real buffer of \( e \) in the switching sub-step of \( t - 1 \), so \( p \) which is forwarded by \( \text{MAXW} \) 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 14:** If a virtual packet \( p_{\text{virt}} \) 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, \( p_{\text{virt}} \) is piggybacked on a real packet.

**Proof.** By induction on time steps, starting from \( t \). For \( t' = t \) \( p_{\text{virt}} \) is piggybacked on a real packet and so the claim holds. For \( t' > t \), if \( p_{\text{virt}} \) 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 \( t' - 1 \) \( \text{MAXW} \) places at least one packet in the buffer at the tail of the next edge \( e \) on the path of \( p \), and \( p_{\text{virt}} \) is added to the virtual transit buffer at the tail of \( e \). Therefore, in \( t' \), \( \text{MAXW} \) forwards a real packet on \( e \), and this packet piggybacks all packets in the virtual transit buffer, including \( p_{\text{virt}} \).

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 15:** 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 \( PV_{\text{IRT}} \), the set of virtual packets piggybacked on a real packet in a certain time step \( t \) over an edge \( e \). For each virtual packet \( p_{\text{virt}} \in PV_{\text{IRT}} \) 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 14, 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 \( p_{\text{virt}} \) 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 the holding buffer at any given time step, \( p_{\text{virt}} \) is the only packet in \( PV_{\text{IRT}} \) which was taken out of the virtual holding buffer at the tail of \( e' \) at time \( t' \). Thus, we may associate each packet in \( PV_{\text{IRT}} \) 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 \( PV_{\text{IRT}} \) we count the number of pairs \((e', t')\) which satisfy this property. This number is the definition of \( f(e) \). Therefore, \( f(G) \), the maximum value of \( f(e) \) in the graph \( G \), is an upper bound for the size of \( PV_{\text{IRT}} \) for any time \( t \) and any edge \( e \).

**Corollary 16:** 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 9. 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 12 and Corollary 16 bound the size of the holding and transit buffers at any given time, to be \( B \) and \( f(G) \), respectively. Claim 15 and Corollary 13 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 \( \text{MAXW} \) by time \( t \). Thus, we have:

\[ \text{ADV}(t) \leq f(G) \cdot \text{MAXW}(t) + m(B + f(G)). \]

### 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 [2]. The first lower bound matches up to a constant factor the upper bound of \( \frac{4\text{max}(d,B)}{B} \) given in Theorem 1. The second lower bound matches the upper bound of \( f(G) \), given in Theorem 9, 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.

**Theorem 17:** There exist a DAG \( G \) such that the competitive ratio of FTG is \( \Omega\left(\frac{n^2}{(n+B)}\right) \).

FTG prioritizes packets according to their remaining path lengths, preferring packets with longer remaining paths to packets with shorter remaining paths. In our proof we show how this property reduces the throughput of FTG, in comparison with the throughput of an adversary.

**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:

- \( V_1 = \{v_1, v_2, \ldots, v_n\} \)
- \( V_2 = \{v_n+1, v_n+2, \ldots, v_{2n}\} \)
Figure 2: Graph used in the $\Omega \left( \frac{n^2(n+B)}{B} \right)$ lower bound for FTG

- $V_3 = \{v_{2n+1}, v_{2n+2}, \ldots, v_n\}$.
- $V_4 = \{v_{n+1}, v_{n+2}\}$.

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

- $E_1 = \{(v_i, v_j) \mid v_i \in V_1, v_j \in V_2\}$. $E_1$ creates a complete uni-directional bi-partite graph between $V_1$ and $V_2$.
- $E_2 = \{(v_i, v_{2n+1}) \mid v_i \in V_2\}$. Note that $v_{2n+1}$ is the “first” node in $V_3$.
- $E_3 = \{(v_i, v_{n+1}) \mid v_i \in V_4\}$. $E_3$ is the set of edges from $V_4$ to $V_5$.
- $E_4 = \{(v_i, v_{n+1}) \mid v_i \in V_5 \setminus \{v_{2n+1}\}\}$. $E_4$ is the “first” node in $V_4$.
- $E_5 = \{(v_{n+1}, v_{n+2})\}$.

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 $(v_i, v_j)$. We refer to these packets as short packets.
  2. $B$ packets into $v_i$ with a path of $(v_i, v_j, v_{2n+1}, v_{2n+2}, v_{n+1}, v_{n+2})$. We refer to these packets as medium packets.
- One packet into $v_i$ with a path of $(v_i, v_{2n+1}, v_{2n+2}, v_{2n+3}, \ldots, v_n, v_{n+1}, v_{n+2})$. We refer to this packet as a long packet.

In time steps $t = 1$ through $t = \frac{n}{2} - 3$ 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 $(v_i, v_j)$. These are the short packets.
  2. One packet into $v_i$ with a path of $(v_i, v_{2n+1}, v_{2n+2}, v_{2n+3}, \ldots, v_{n-t}, v_{n+1}, v_{n+2})$. These are the medium packets.
- One packet into $v_i$ with a path of $(v_i, v_{2n+1}, v_{2n+2}, v_{2n+3}, \ldots, v_{n-t}, v_{n+1}, v_{n+2})$. 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. FTG prioritizes long over medium packets. The situation at time $t = \frac{n}{2} + 2$ will be as follows: (a) None of the packets are delivered yet. (b) For each time $t \in [0, \frac{n}{2} - 3]$, the long packet injected at $t$ will be at node $n-t$, with $v_{n+1}$ as its next node. (c) All medium packets will be in one of the nodes of $V_1$, $V_2$ or in the first two nodes of $V_3$ which are $v_{2n+1}$ and $v_{2n+2}$. Starting from time $t = 3$, a packet will cross edge $(v_{n+1}, v_{n+2})$ for at most $3B + 3$ consecutive time steps. After that, no packet will be in the network. The throughput of FTG is therefore bounded from above by $3B + 3$ for this series of injections.

The adversary can drop the following packets upon their injection in each time step $t \in [0, \frac{n}{2} - 3]$:

- All long packets, and
- 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 step time after their injection. At the beginning of time $t = \frac{n}{2} - 1$, 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 $(\frac{n}{2})^2(\frac{n}{2} - 2)$ short packets, plus $(\frac{n}{2})^2(B-1)$ medium packets. In all, FTG delivers at most $3B + 3$ packets and the adversary delivers $(\frac{n}{2})^2(\frac{n}{2} - 2) + (\frac{n}{2})^2(B-1)$ packets.

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

**Theorem 18:** There exists a DAG $G$, such that the competitive ratio of any greedy protocol is $\Omega(n^2)$.

This lower bound illustrates the inherent drawback of Greedy protocols which may flood a bottle neck of the network with packets arriving from different paths. Since no coordination is done between the routing 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.

**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:

- $E_1 = \{(v_i, v_j) \mid v_i \in V_1, v_j \in V_2\}$. $E_1$ creates a complete uni-directional bi-partite graph between $V_1$ and $V_2$.
- $E_2 = \{(v_i, v_{n+1}) \mid v_i \in V_2\}$. Note that $v_{n+1}$ is the “first” node in $V_3$.
- $E_3 = \{(v_{n+1}, v_{n+2})\}$.

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 with a path of $(v_i, v_j, v_n, v_{n+1})$ into node $v_i$. Any greedy protocol will forward...
Figure 3: Graph used in the $\Omega(n^2)$ lower bound for any greedy protocol

one packet to its destination on $e = (v_n, v_{n+1})$ in every time step starting from time $t = 3$ for $3B$ consecutive time steps. After that, no packets will remain in the network. This gives a throughput of $3B$. The adversary will drain each buffer, one after the other, such that no packet is dropped. The throughput of the adversary will be $B(\frac{n}{2})^2$.

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.

5. CONCLUSIONS

In this paper we extend the Competitive Network Throughput (CNT) model introduced in [2] 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 [2] for the unweighted case, and we thus give strong (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 [2]. Providing a non-greedy online centralized algorithm that beats this lower would be an important first step.

6. REFERENCES