Anycasting and Multicasting in Adversarial Systems: Routing and Admission Control

Baruch Awerbuch*
Department of Computer Science
Johns Hopkins University
3400 N. Charles Street
Baltimore, MD 21218, USA
baruch@cs.jhu.edu

André Brinkmann†
Heinz Nixdorf Institute and
Department of Electrical Engineering
University of Paderborn
33102 Paderborn, Germany
brinkman@hni.upb.de

Christian Scheideler
Department of Computer Science
Johns Hopkins University
3400 N. Charles Street
Baltimore, MD 21218, USA
scheideler@cs.jhu.edu

May 14, 2002

Abstract

In this paper we consider the problem of routing packets in dynamically changing networks, concentrating on two different modes: *anycasting* and *multicasting*. In anycasting, a packet has a set of destinations but only has to reach any one of them, whereas in multicasting, a packet has a set of destinations and has to reach all of them. Both communication modes have a tremendous number of applications, but so far not much is known from a theoretical point of view about how to efficiently support these communication modes even in static networks. We demonstrate in this paper that even if both the network and the packet injections are under adversarial control, efficient distributed routing strategies can be found for both modes. This even holds if the adversary is not bounded in the number of injections, and therefore packets may have to be dropped from overfull buffers. In order to study the performance of our protocols, we use competitive analysis. The performance is measured in terms of communication throughput (i.e. the number of successful packet deliveries) and space overhead. Our results demonstrate that, in principle, efficient communication is possible even in such highly dynamic networks as mobile ad hoc networks and peer-to-peer networks.

*Supported by DARPA grant F306020020550 "A Cost Benefit Approach to Fault Tolerant Communication" and DARPA grant F30602000-2-0526 "High Performance, Robust and Secure Group Communication for Dynamic Coalitions".

†Supported in part by the DFG-Sonderforschungsbereich 376 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen". Part of the research was done while visiting the Johns Hopkins University, supported by a scholarship from the German Academic Exchange Service (DAAD Doktorandenstipendium im Rahmen des gemeinsamen Hochschulsonderprogramms III von Bund und Ländern).

1 Introduction

This paper considers the problem of designing distributed algorithms for the delivery of packets in dynamic networks with arbitrarily changing topology. We consider two different communication modes: *anycasting* and *multicasting*. In anycasting, a packet has a set of destinations but only has to reach any one of them, whereas in multicasting, a packet has a set of destinations and has to reach all of them. Both modes have numerous applications (please see the appendix for concrete examples). However, to our surprise, it seems that anycasting has not been studied by the theory community so far, whereas numerous papers have already been written on the multicasting problem (see Section A in the appendix for a survey). These papers, however, only concentrate on the multicast tree selection and admission control problem, but not on how to actually route multicast packets in an efficient way. The approximation ratios and competitive ratios shown for the algorithms and the corresponding lower bounds do not look very promising in this regard (particularly, since we actually allow adversaries to be adaptive). However, we demonstrate in this paper that even if both the network and the packet injections are under adversarial control, distributed routing strategies with a close to optimal throughput can be found for both anycasting and multicasting. Thus, in principle, anycasting and multicasting can even be supported in such networks as mobile ad hoc networks or peer-to-peer networks, where connections between users may change quickly and unpredictably.

1.1 Our approach and related results

We measure the performance of our protocols by comparing them with the a best possible strategy that knows all actions of the adversary in advance. The performance is measured in terms of communication throughput and space overhead. In order to ensure a high throughput efficiency in dynamic networks, several challenging tasks have to be solved:

- *Routing*: What is the next edge to be traversed by a packet?
- *Switching*: What is the next packet to be transmitted on an edge? In particular, which destination should be preferred?
- *Admission control*: What is the packet to be dropped if a buffer is full?

This seems to be difficult to decide without knowing the future, especially if both the topology and the packet injections are under adversarial control: choosing the wrong edge may lead a packet further away from its destination, and preferring packets with a destination that will be isolated in the future or dropping the wrong packets in case of overfull buffers may tremendously decrease the number of successfully delivered packets. However, we will show that the seemingly impossible task is possible, both for anycasting and multicasting.

The study of adversarial models was initiated, in the context of *switching alone*, by Borodin et al. [16]. Other work on switching includes [6, 22, 23, 24, 42, 45]. In these papers it is assumed that the adversary has to provide a path for every injected packet and reveals these paths to the system. The paths have to be selected in a way that they do not overload the system. Hence, it only remains to find the right queueing discipline (such as furthest-to-go) to ensure that all of the packets can reach their destinations.

The study of adversarial models was initiated, in the context of *routing*, by Awerbuch, Mansour and Shavit [11] and further refined by [5, 8, 9, 10, 23]. In these papers the model is used that the adversary *does not* reveal the paths to the system, and therefore the routing protocol has to figure out paths for the packets by itself. Based on work by Awerbuch and Leighton [10], Aiello et al. [5] show that there is a simple distributed routing protocol that keeps the number of packets in transit bounded in a dynamic network if, roughly speaking, in each window of time the paths selected for the injected packets require a capacity that is below what the available network capacities can handle in the same window of time. Awerbuch et al. [8] generalize this to an adversarial model in which the adversary is allowed to control the network topology and packet injections as it likes, as long as for every injected packet it can provide a schedule to reach its destination. They show that even for the case that the network capacity is fully exploited, if all packets have the same destination, the number of packets in transit is bounded at any time

We note that all of the above work on adversarial switching and routing only considered the *unicasting* mode (every packet has a single destination). Furthermore, none of them considers the problem of *admission control* inside the network. We also note that all approaches in the adversarial routing area, including this current paper, are based on simple load balancing schemes first pioneered by Awerbuch, Mansour and Shavit [11], and refined in [2, 3, 4, 9, 10, 5, 8] for various routing purposes. Our achievement is to demonstrate that balancing even works for anycasting and, more importantly, even multicasting in highly dynamic networks. We also extend it to the admission control scenario.

1.2 Main contributions of this paper

The weakness of the adversarial models above is that all of them assume that the adversary never overloads the system with packets. In static networks this is a reasonable restriction, since one can imagine that in principle it is possible to perform some kind of admission control *before* injecting a packet into the system. However, in highly dynamic networks such as mobile ad-hoc networks, this may not be possible without being too conservative and therefore wasting too much of the already scarce bandwidth. Hence, for dynamic networks it would be highly desirable to have protocols that manage to handle not only the route selection and switching part but also packet-level admission control, i.e. dropping packets from either input or intermediate buffers.

Also, all papers in the adversarial routing literature we know of do not allow the algorithm to drop packets. In the case that the adversary uses a maximum possible injection rate (i.e. rate $r = 1$), the best known upper bound on the number of packets in a buffer when using an online algorithm is exponential [23] in the context of switching and still unknown in the context of routing.

Admission control: We show that *dropping some packets significantly improves routing* and show how to drop packets to ensure a high throughput and low buffer space. In fact, we show that a very simple dropping rule works: if a packet enters an already full buffer, drop it. Our upper bounds show that by simply dropping a small fraction of the packets, the buffer sizes necessary to successfully deliver the rest can be brought down to *linear* (or better) in the number of nodes in the system. We also prove exponential lower bounds and instability results showing that admission control is necessary.

New anycasting and multicasting problems: This paper formalizes two new problems: the anycast and multicast routing and admission control problem for arbitrary (bursty) traffic in arbitrary dynamic networks, and provides first competitive solutions. These problems occur naturally in many settings in a mobile ad hoc network, e.g., multicasting occurs in case of dissemination of contents, and anycasting occurs when such contents (which may be cached in the network) is being searched for. (See the appendix for some more examples.)

Local multicast routing: It is far from obvious whether it is at all *possible* to route multicast traffic *locally* with a *near-optimal throughput*. Besides stating the problem for the first time, we provide the first competitive solution, which extends previous load balancing approaches by scaling the priorities based on the number of remaining destinations of a packet.

Option set model: We introduce a new model for the study of algorithms in adversarial systems, called *option set model*, that allows to reduce arbitrary anycasting problems to gathering problems (i.e. all packets have the same destination). This tremendously simplifies the analysis of algorithms for anycasting.

New analytical techniques: None of the analytical techniques used in the papers cited above, including [8], seem to be extendable in a straightforward way to analyze the performance of our algorithms under an unbounded adversary. Delay-based techniques certainly fail, since packets can have an unbounded delay in our models. Moreover, potential-based techniques concentrated in the past on showing that beyond a certain point, the potential of the system cannot increase any more. This does not give much information in our case, where nodes have bounded buffers, since the potential is naturally limited. Instead, we found a new potential method that allows to use the *average* length of paths used by successful packets (instead of the maximum path length or the number of nodes in the system, as was previously done) to measure the performance of our routing protocols.

In order to state our results in a more precise way, we need some notation.

1.3 Models

First, we describe the basics or our network model and injection model. We assume that $V = \{1, \ldots, n\}$ represents the set of nodes in the system. The selection of the edges is under adversarial control and can change from one time step to the next. We assume that all edges are directed. This does not exclude the undirected edge case, since an undirected edge can be viewed as consisting of two directed edges, one in each direction. Each edge can forward at most one packet in a time step. Each node can have at most Δ incoming and at most Δ outgoing edges at any time. Δ can be seen as the maximum number of active (logical or physical) connections a node can handle at the same time (due to, for example, its hardware restrictions). Apart from this restriction, the adversary can interconnect the nodes in an arbitrary way in each time step. This includes the possibility of connecting the same pair of nodes via several edges.

Each node has buffers to store packets, and each buffer is responsible for a specific type of packet. For example, a buffer may be responsible for storing all packets for a specific destination. We assume that each buffer has a size of H, i.e. it can store up to H packets.

This paper concentrates on anycasting and multicasting. $\mathcal{D} \subseteq 2^V$ denotes the set of all (anycast of multicast) destination groups used by the adversary. Every injected packet P is assigned to one of the groups $D \subseteq \mathcal{D}$. We say in this case that P is *of type* D. In anycasting, if P reaches any one of the nodes in D, it can be absorbed. In multicasting, P has to reach all of the nodes. As an example, in the standard unicast situation, $\mathcal{D} = \{\{1\}, \{2\}, \ldots, \{n\}\}$.

The adversary does not only control the topology of the network but also the injection of packets. We distinguish between two adversarial injection models: one for anycast injections and one for multicast injections.

The adversarial anycasting model.

Each anycast packet will be given a fixed anycast group at the time of its injection. We allow this group just to be specified implicitly (for example, by an anycast address). Note that for implicitly specified groups, the nodes in the network may have no knowledge about their size. It may even be possible that the group is empty. Thus, our anycast algorithm has to cope with this situation.

The adversary can inject an arbitrary number of packets and can activate an arbitrary number of edges in each time step as long as the number of incoming or outgoing edges at a node does not exceed Δ. In this case, only some of the injected packets may be able to reach their destination, even when using a best possible strategy. For each of the successful packets a *schedule* can be specified. A schedule $S = (t_0, (e_1, t_1), \ldots, (e_\ell, t_\ell))$ consists of a sequence of movements by which the injected packet P can be sent from its source node to one of its destination nodes. It has the property that P is injected at time t_0, the edges $e_1, \ldots, e_\ell$ form a connected path, with the starting point of e_1 being the source of P and the endpoint of e_ℓ being the destination of P, the time steps have the ordering $t_0 < t_1 < \ldots < t_\ell$, and edge e_i is active at time t_i for all $1 \le i \le \ell$.

Schedules of successful packets must fulfill the property that no two schedules *intersect*. That is, no two schedules are allowed to use the same edge (resp. the same copy if several edges connect the same pair of nodes) in the same time step. When speaking about schedules in the following, we always mean a delivery strategy chosen by a best possible routing algorithm.

The adversarial multicasting model

In the multicast model, a packet may have to be sent to several destinations. We assume here in contrast to the anycast case that all destinations of a multicast packet are explicitly given. Otherwise, it does not seem to be possible to provide efficient algorithms for multicasting within our network and multicast model, because once multiple copies of a packet exist, it does not seem to be possible with implicit multicast addresses to avoid multiple deliveries and to decide when to drop a packet in a distributed way.

In order to allow a higher efficiency than simply sending out one packet for each destination, we adopt the standard multicast model, which assumes that a multicast packet only needs one time step to cross an edge and that a multicast packet can split itself while in transit. Requiring an efficient use of this property makes the multicasting problem considerably harder than the unicasting or anycasting problem. Especially when using dynamic networks, it seems to be a formidable problem to decide when and where to split a packet.

As in the anycasting model, the adversary is allowed to inject an unlimited number of multicast packets. However, only some of these packets may be able to reach their destinations, even when using a best possible strategy. For each multicast packet P that can reach k of its destinations, a schedule can be identified. Any such schedule can be reduced to a directed tree with k leaves representing the destinations. Hence, we can view all schedules to be of the form $S = (t_0, (e_1, t_1), \ldots, (e_\ell, t_\ell))$, where t_0 specifies the injection time of P, the edges $e_1, \ldots, e_\ell$ form an directed tree with the source as the root and the k destinations as leaves, for every directed path $((e_{i_1}, t_{i_1}), \ldots, (e_{i_k}, t_{i_k}))$ the time steps have the ordering $t_{i_1} < t_{i_2} \ldots < t_{i_k}$, and edge e_i is active at time t_i for all $1 \leq i \leq \ell$. As in the anycasting model, the schedules must fulfill the property that they do not overlap.

Analytical approach

Each time a multicast or anycast packet reaches one of its destinations, we count it as one *delivery*. The number of deliveries that is achieved by an algorithm is called its *throughput*. We are interested in maximizing the throughput. (As we will see, this also allows to maximize the number of multicast packets that reach *all* of their destinations if all multicast groups are of the same size.) Since the adversary is allowed to inject an unbounded number of packets, we will allow routing algorithms to drop packets so that a high throughput can be achieved with a buffer size that is as small as possible.

In order to compare the performance of a best possible strategy with our online strategies, we will use competitive analysis. To have a fair comparison, both the optimal and the online strategy must be based on the same type of resources. In our case, we will assume that both have the same number of buffers in each node and each buffer is reserved for a specific type of packet (in the anycasting case, for example, each buffer will be responsible for holding packets for a specific anycast group).

Given any sequence of edge activations and packet injections σ, let $\mathrm{OPT}_B(\sigma)$ be the maximum possible throughput (i.e. the maximum number of deliveries) when using a buffer size of B, and let $A_{B'}(\sigma)$ be the throughput achieved by some given online algorithm A with buffer size B'. We call an online algorithm A (c, s)-*competitive* if for all σ and all B, A can guarantee that

$$A_{s' \cdot B}(\sigma) \geq c \cdot \mathrm{OPT}_B(\sigma) - r$$

for any $s' \geq s$, where $r \geq 0$ is some value that is independent of σ (but may depend on s, B and n). $c \in [0, 1]$ denotes here the fraction of the best possible throughput that can be achieved by A and s denotes the space overhead necessary to achieve this. If c can be brought arbitrarily close to 1, A is also called $s(\epsilon)$-*competitive* or simply *competitive*, where $s(\epsilon)$ reflects the relationship between s and ϵ with $c = 1 - \epsilon$. Obviously, it always holds that $s(\epsilon) \geq 1$, and the smaller $s(\epsilon)$, the better is the algorithm A.

1.4 New results

A routing algorithm is called *pure* if it does not produce copies of packets or violate their atomic property (in a sense that it cuts packets into pieces or compresses them in some way). We will only consider pure algorithms in this paper. In the following, B will always mean the buffer size of an optimal routing algorithm.

Our new results are arranged in three sections. In Section 3, we demonstrate with the help of lower bounds and instability results that even if the adversary is friendly (i.e. it only injects packets that have a schedule when using a buffer size of B), routing without the ability to drop packets may have a poor performance both with respect to throughput and space overhead. In the following two sections, we then demonstrate that if it is allowed to drop packets, a near-optimal throughput can be achieved with a low space overhead.

In particular, we present a simple algorithm for anycasting, called *T-balancing algorithm*, and an extension of it for multicasting, that achieves the following results:

- *Anycasting*: For every $T \geq B + 2(\Delta - 1)$, the T-balancing algorithm is $1 + (1 + (T + \Delta)/B)L/\epsilon$-competitive, where L is the *average* path length used by successful packets in an optimal solution. The result is sharp up to a constant factor. We also show how to extend this result to bus networks and networks based on virtual circuit switching.

- *Multicasting*: For every $T \geq B + 4D\Delta$, the multicast variant of the T-balancing algorithm is $1 + (1 + (T + D\Delta)/B)D \cdot L/\epsilon$-competitive, where L is defined as above and D is the maximum number of destinations a packet can have. This also implies that for multicast groups of uniform size, the T-balancing algorithm is $1 + (1 + (T + D\Delta)/B)D^2 \cdot L/\epsilon$-competitive concerning the number of packets that can reach *all* of their destinations.

As indicated by their names, our algorithms are based on simple balancing methods that have already been shown to be effective in previous works [9, 5, 8]. New ingredients of our algorithms are rules for dropping packets and rules for splitting packets in the case of multicasting. In fact, we show that a very simple dropping rule works: if a packet enters an already full buffer, drop it.

Our results demonstrate that there is hope that efficient and flexible communication strategies can be constructed that can even handle such unpredictable networks like mobile ad-hoc networks or peer-to-peer networks.

Finally, we note that all of our results also hold for faulty networks as long as faulty edges can be detected and faulty nodes either remain faulty forever or can recover completely without a loss of information (i.e. packets cannot get lost due to faults).

2 A Simple Model and Algorithm for Anycasting

In this section we present a model that is even more general than the adversarial anycasting model but much simpler to use for our upper and lower bounds. Afterwards, we present a simple routing algorithm that will be the basis of our algorithmic investigations.

2.1 The option set model

Next we show how to transform the adversarial anycasting model into a more general model called *option set model* that is easier to handle. All of our upper bounds will be based on this model (or its extensions).

If a connection is offered between two nodes, an online algorithm is faced with two problems: it has to decide whether to send a packet along the connection and if so, which type of packet to prefer. Since there may be many different types of packets stored in a node, a connection offered by the adversary can be interpreted as a set of virtual edges, one for each type of packet, and out of this set only one edge can be used. Viewing each buffer as a separate, virtual node, each edge e offered by the adversary can therefore be seen as offering a subset of virtual edges $S_e = \{e'_1, \ldots, e'_s\}$, where each edge e'_i connects two virtual nodes representing two buffers for the same type of packet. For each set S_e, at most one edge can be chosen for the transmission of a packet.

With the approach of using virtual nodes and virtual edges, we do not have to distinguish between different types of packets any more. Since they can be distinguished by their virtual node numbers and they are naturally separated from each other as long as there is no virtual edge connecting two virtual nodes representing buffers for different types of packets, we can view all packets as being of the same type. Furthermore, since every packet is absorbed once it reaches one of its virtual destination nodes, we can combine all destination nodes into a single, virtual destination node, allowing us to view all the packets in the system as having the same destination.

Thus, arbitrary anycasting problems in the original model can simply be modelled as gathering problems in the option set model.

The only difference compared to gathering in a standard network model is that instead of just having a set of edges $E = \{e_1, \ldots, e_k\}$ at some time step in which each edge can be used independently (which corresponds to the model used in [8]), we now have a set $\mathcal{E}$ of *option sets* representing subsets of edges, and for each of these option sets the problem is to select "the right" edge for the transmission of a packet. This will allow us in the following to tremendously simplify the presentation and analysis of communication protocols.

Taking the arguments above into account, it is easy to check that any upper bound shown for routing in the option set model also implies an upper bound for the anycast model. The reverse statement, however, is not necessarily true, since in the unrestricted option set model no difference is made between different types of nodes, and therefore it is more general than the anycast model.

Finally, we note that the Δ in the original model now means the maximum number of incoming or outgoing edges a virtual node can have. For the destination node, we will not require any limitation on the number of incoming edges. The option set model can be generalized so that it can also cover multicasting situations. However, we will wait with this until Section 5.

2.2 The T-balancing algorithm

We will present our algorithm in the option set model. Thus, when we speak about nodes and edges, we mean virtual nodes and virtual edges. Let $h_{v,t}$ denote the amount of packets in node v at the beginning of time step t. For the destination node d, $h_{d,t} = 0$ at any time. $h_{v,t}$ will also be called the *height* of node v at step t. The maximum height a node can have is denoted by H.

We now present a simple balancing strategy that extends the balancing strategies used by Aiello et al. [5] and Awerbuch et al. [8] by a rule for deleting packets. In every time step $t \geq 1$ the T-balancing algorithm performs the following operations.

1. For every option set S_i, determine the edge $e = (v, w) \in S_i$ with maximum $h_{v,t} - h_{w,t}$ and check whether $h_{v,t} - h_{w,t} > T$. If so, send a packet from v to w (otherwise do nothing).
2. Receive all incoming packets and absorb all packets that reached the destination. Afterwards, receive all newly injected packets. If a packet cannot be stored in a node because its height is already H, delete it.

Note that if T is large enough compared to Δ, then packets are guaranteed never to be deleted at intermediate nodes but only at the source. This provides the sources with a very easy rule to perform admission control: if a packet cannot be stored because its buffer is already full, delete it.

3 Unicasting without Admission Control

In this section we demonstrate that routing without admission control mechanisms seems to be very difficult if not impossible, even in the adversarial unicast setting, and even if an unbounded (or extremely high) amount of resources for the buffering is available. Due to space limitations, most of the proof can be found in the appendix.

We will start by defining some properties of online routing algorithms which intuitively seem to be necessary for the successful online delivery of packets. A *priority function* $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$ gets as arguments two buffer heights and outputs a number determining the priority with which a packet should be sent from one to the other buffer. In a balancing algorithm that uses a priority function f, the pair with the highest priority wins. We call f *monotonic* if for all $h_1, h_2 \in \mathbb{N}_0$, $f(h_1 + 1, h_2) > f(h_1, h_2)$ and $f(h_1 + 1, h_2 + 1) \geq f(h_1, h_2)$.

Consider a routing algorithm that uses a monotonic priority function to determine a winning pair (h_1, h_2) for each option set resp. activated edge in the unicast model. If $h_1 \leq h_2$, no packet is allowed to be sent. Otherwise a packet for that pair (or none) may be sent, but if the buffer corresponding to h_2 is a destination buffer, a packet has to be sent for that pair. Intuitively, these rules seem to be reasonable to ensure a high throughput, and we will therefore call this class of routing algorithms *natural* algorithms.

We start with an observation that demonstrates that for adversaries that are unbounded in their injections it is necessary to drop packets in a natural routing algorithm in order to make sure that *any* of the injected packets can be delivered, even if only two different destinations are used. Note that when we speak about algorithms that do not drop packets, this implies that they must have sufficient space to accommodate all injected packets.

Claim 3.1 *For every natural algorithm that does not drop packets, there is an adversary for unicast injections using just two different destinations that can force the algorithm to deliver no packet, no matter how high the throughput of an optimal strategy can be.*

Proof. The adversary will simply pick one destination as the so-called *dead destination* and will inject so many packets into the system that whenever an edge is offered, a packet will be sent for the dead destination. However, the adversary will never offer an edge directly to the dead destination so that none of these packets can ever be delivered. For the other destination, the adversary injects packets that could all be sent to that destination when using an optimal strategy. □

Thus, unbounded adversaries seem to be difficult to handle without allowing packets to get dropped. However, what about "friendly" adversaries, i.e. adversaries that only inject packets that have a schedule? We show that also in this case some natural algorithms have severe problems if packets cannot be dropped.

A routing algorithm is called *stable* if the number of packets in transit does not grow unboundedly with time. Under the assumption that we have a friendly adversary that ensures that no more than S schedules are active at any point of time, it is in principle possible to route packets so that the number of packets in transit is bounded (namely, by S). In order to investigate the stability of natural algorithms, we start with an important property of natural algorithms that allows us to study instability in the option set model.

Theorem 3.2 *For any natural deterministic algorithm it holds: If it is not stable in the option set model, it is also not stable in the adversarial unicast model.*

Thus, we only have to prove instability in the option set model, which is much easier than in the original model. We start with algorithms that use exponential priority functions.

Theorem 3.3 *Natural routing algorithms which are based on exponential priority functions are not stable in the option set model.*

The proof of the theorem immediately implies the following result.

Corollary 3.4 *Natural routing algorithms which always prefer the buffer with the largest number of packets are not stable in the option set model.*

Next we show that also the T-balancing algorithm without the dropping rule has severe problems. n represents the number of nodes in the original unicasting model.

Theorem 3.5 *If the adversary is allowed to inject packets for more than one destination, then the adversary can force the T-balancing algorithm to have a buffer size of at least $\Theta(B \cdot 2^{\lfloor n/4-2 \rfloor})$.*

The theorems together with the results in [8] imply that only for the case that we have a single destination and that all option sets are of size 1, the T-balancing algorithm without a dropping rule can be space-efficient under friendly adversaries.

How about arbitrary routing algorithms? The next result proves a general lower bound for them in the option set model. The proof can be found in the appendix.

Theorem 3.6 *Even when using an oblivious adversary and even if the maximum path length necessary for an optimal strategy with buffer size B is at most 2, any pure $(1, s)$-competitive algorithm must fulfill that $s \geq (N-2)^2(B-1)/(2B)$.*

Comparing the results in this section with the results we will show in the next sections, it becomes clear that the ability to drop packets can tremendously improve the performance of routing algorithms.

4 Adversarial Anycasting

In this section we present essentially matching upper and lower bounds for the T-balancing algorithm in the option set model. As mentioned in Section 2, this also immediately provides an upper bound on the performance of the T-balancing algorithm in the adversarial anycasting model.

Let $L \leq n-1$ denote an upper bound on the (best possible) average path length used by the successful packets in an optimal algorithm with buffer size B, and let Δ denote the maximum number of (virtual) edges leaving or leading to a (virtual) node that can be active at any time (excluding the destination). We do not demand that these edges have to connect different pairs of nodes. Hence, the result below also extends to dynamic networks with non-uniform edge capacities. The theorem can also be generalized to more complex network models than networks based on point-to-point connections. More details can be found in Section D.

Theorem 4.1 *For any $\epsilon > 0$ and any $T \geq B+2(\Delta-1)$, the T-balancing algorithm is $1+(1+(T+\Delta)/B)L/\epsilon$-competitive.*

Proof. A detailed proof of the theorem can be found in the appendix. The basic idea is to define three types of packets: representatives, zombies, and losers. As long as a packet is still with its schedule, it is a representative. Newly injected packets that do not have a schedule are declared zombies. If a zombie enters a buffer slot that has a height of at most $H-B$ or a representative cannot follow its schedule any more, it is transformed into a loser. Using a potential function based on the number of losers in the system, we can show that any option set not containing a schedule edge never increases the potential, and any option set containing a schedule edge can increase the potential by at most $T+B+\Delta$. Combining this with other properties, the theorem can be shown. $\square$

Next we demonstrate that the analysis of the T-balancing algorithm is essentially tight, even when using just a single destination.

Theorem 4.2 *For any $\epsilon > 0$, $T > 0$, and $L \geq 1$, the T-balancing algorithm requires a buffer size of at least $T \cdot (L-1)/\epsilon$ to achieve a more than $1-\epsilon$ fraction of the best possible throughput.*

Proof. Consider a source node s that is connected to a destination d via two paths: one of length 1 and one of length $(L-1)/\epsilon$. Further suppose packets are injected at s so that $1-\epsilon$ of the injected packets have a schedule along the short path and ϵ of the packets have a schedule along the long path. Then the average path length is $1(1-\epsilon) + ((L-1)/\epsilon) \cdot \epsilon \leq L$. Since each time a packet is moved forward along a node its height must decrease by at least T, a packet can only reach the destination along the long path if s has a buffer of size $H \geq T \cdot (L-1)/\epsilon$. Hence, such a buffer size is necessary to achieve a throughput of more than $1-\epsilon$. □

5 Adversarial Multicasting

Next we show how to extend the T-balancing algorithm to an algorithm that even handles multicasting. For this we first extend the option set model so that it can also be used for multicasting. Afterwards, we describe and analyze an extended form of our T-balancing algorithm.

5.1 The extended option set model

Next we show that the multicasting scenario can be reduced to a model that is similar to the one given in Section 2.1. Suppose that a multicast packet can have up to D different destinations. Then we give every node $\sum_{d=1}^{D} \binom{n}{d}$ *multicast buffers* M_U, one for each possible set of destinations U a packet might still have to visit. A buffer M_U with $|U| = k$ is also called a *k-buffer*, and packets stored in it are called *k-packets*.

Suppose now that we have a multicast packet with destination set U that is currently at node v. Then it is stored in the buffer M_U of v. In contrast to an anycast packet, there is now more than one way of transmitting a multicast packet. We distinguish between *regular* and *split* transmissions.

If a multicast packet is sent to some node w in its entirety, then we talk about a regular transmission. If $w \notin U$, then the packet will be moved to M_U in w. Otherwise, it will be moved to $M_{U\setminus\{w\}}$ in w. For the special case that $U = \{w\}$, the multicast packet will be absorbed.

If a multicast packet is split into several pieces that are sent out along different edges, we speak about a split transmission. In our model we will only consider pairwise split operations in a sense that given an edge (v, w), a multicast packet is only sent with a part of its destinations, say $U' \subset U$, to w, and the rest is moved to buffer $M_{U\setminus U'}$ in node v. We will motivate later why this restriction is possible without us having to restrict a best possible strategy in any way.

We can again view the buffers as virtual nodes and the activation of a communication link as the creation of an option set that consists of several virtual edges connecting different virtual nodes. For every node v associated with a k-buffer, we call it a k-node and set $\mu(v) = k$. Virtual edges associated with regular transmissions are called *regular* edges and denoted by (v, w), where v is the virtual node representing the source buffer and w is the virtual node representing the destination buffer. Since the number of destinations in a multicast packet can never grow, every regular edge (v, w) fulfills $\mu(v) \geq \mu(w)$. Virtual edges associated with split transmissions are called *split* edges and denoted by (v, w, w'), where v is the source buffer, w is the destination buffer of one part and w' is the destination buffer of the other part. Since a split operation can never increase the total number of destinations of a multicast packet, every split edge (v, w, w') fulfills $\mu(v) \geq \mu(w) + \mu(w')$.

Combining all destination buffers in a single buffer, we are essentially back to the model formulated in Section 2.1 with the difference that the use of split edges creates two packets out of one.

It remains to motivate, why we only have to study regular and split transmissions as they are defined above. Two issues have to be addressed for this:

1. An optimal solution might split a packet into packets with overlapping sets of destinations.
2. An optimal solution might split a packet into more than two parts and send them along several edges in the given time step.

Concerning the first case, any optimal solution can easily be reduced to an optimal solution that does not create overlapping sets of destinations, since multiple deliveries of a multicast packet to the same destination only count in our model as a single successful delivery. Concerning the second case, any situation with multiple splits can be replaced by a sequence of splits and a regular transmission in which a packet is only split into two packets without changing the outcome. For example, an event in which a packet with destination set $\{w_1, w_2, w_3\}$ at node u is split into 3 packets $\{w_1\}$, $\{w_2\}$, and $\{w_3\}$ that are sent to the nodes v_1, v_2, and v_3 respectively, can also be represented as a sequence of events in which first $\{w_1\}$ is sent to v_1 and $\{w_2, w_3\}$ to the corresponding buffer in u, then $\{w_2\}$ is sent to v_2 and $\{w_3\}$ to the corresponding buffer in u, and finally $\{w_3\}$ is sent to v_3. (We just have to make sure in our analysis that the corresponding option sets are considered in the same order as the sequence of pairwise splits. This ordering does not cause problems with other split operations, since each option set can only be used by at most one multicast schedule.) Hence, any optimal solution can be reduced to an optimal solution with pairwise splits. Therefore, concentrating only on pairwise split operations in our algorithm below does *not* mean that it cannot be made competitive any more against algorithms that allow arbitrary split operations. Concentrating only on pairwise split operations has several advantages: it significantly improves the time to determine the best operation, and it simplifies the analysis.

5.2 The multicast T-balancing algorithm

Let us view each multicast packet with k destinations as consisting of k *packet units*, and let the *maximum height* H be now defined as the maximum number of packet units that can be stored in a node. This means for a k-node v that v can store at most $\lfloor H/k \rfloor$ multicast packets. Hence, the smaller k, the more multicast packets are allowed to be stored in a node. For any v and t, the *height* $h_{v,t}$ denotes now the number of packet units in v at step t. The height of a multicast packet P and in some node v is defined as the height of its highest packet unit. In every time step t the multicast T-balancing algorithm performs the following operations:

1. For every option set $E_{i,t}$ do the following: Let $E^r_{i,t}$ be the set of regular edges and $E^s_{i,t}$ be the set of split edges in $E_{i,t}$. Let $e_r = (x, y) \in E^r_{i,t}$ be the edge with maximum $\delta_{e_r} = \mu(x)h_{x,t} - \mu(y)h_{y,t}$ for which $\delta_{e_r} > \mu(x) \cdot T$, and let $e_s = (v, w, w') \in E^s_{i,t}$ be the edge with maximum $\delta_{e_s} = \mu(v)h_{v,t} - \mu(w)h_{w,t} - \mu(w')h_{w',t}$ for which $\delta_{e_s} > \mu(v) \cdot T$. Choose among these two edges the edge e with maximum δ_e and send a packet along that edge (if there is no such edge, do nothing).
2. Receive incoming packets and absorb all packets that reached the destination. Afterwards, receive all newly injected packets. If a packet cannot be stored in a k-node because its height is already H, then delete the new packet.

The most important differences between this algorithm and the T-balancing algorithm for anycasting are that even if T is large, packets may now get dropped at both source nodes and intermediate nodes and that different buffer heights are weighted in different ways. Giving buffers representing multicast groups of larger size a larger weight makes sense, because forwarding a packet for a large multicast group intuitively has more benefit than forwarding a packet for a small multicast group.

Our aim will be to show that if H is sufficiently high, the balancing algorithm can achieve a competitive ratio that is arbitrarily close to 1. For this we first describe how to extend the parameters B and L to the multicast situation.

Here, the buffer size B represents the maximum number of packet units that can be stored in a buffer by the optimal routing strategy. Defining B via packet units instead of multicast packets makes more sense, since during the course of the time a multicast packet with k destinations may be split into k packets with a single destination. In order to allow an optimal routing strategy to efficiently deliver multicast packets in this situation, it is reasonable to allow 1-buffers to have k times more space for multicast packets than in k-buffers.

Analogously, L now denotes an upper bound on the average number of edges used for the successful delivery of a single packet unit in an optimal solution. Thus, if the total number of edges used by the schedules is p and the total number of successful deliveries is s, then $L = p/s$. Hence, the more compact the multicast tree (i.e. the fewer edges it has), the better for L.

As before, Δ denotes the maximum number of edges leaving or leading to a single node that can be active at any time. Again, we do not demand that these edges have to connect different pairs of nodes. Hence, the result below also extends to networks with non-uniform edge capacities.

Theorem 5.1 *For any $\epsilon > 0$ and any $T \geq B + 4D\Delta$, the multicast T-balancing algorithm is $1 + (1 + (T + D\Delta)/B)D \cdot L/\epsilon$-competitive.*

The proof of the theorem is basically an extension of the proof of Theorem 4.1 and quite technical. The details can be found in the appendix.

6 Conclusions and Open Problems

In this paper we presented simple balancing algorithms for anycasting and multicasting in adversarial systems. Many open questions remain. Although our space overhead is already reasonably low (essentially, $O(L/\epsilon)$), the question is whether it can still be reduced. For example, could knowledge about the location of a destination or structural properties of the network (for instance, it has to form a planar graph) help to get better bounds? Or are there certain non-pure protocols that can achieve a lower space overhead? In the worst case, our multicasting result may need an exponentially large number of buffers per node. Can this be reduced to a polynomial number without restricting a best possible strategy? We suppose that this is not possible in general. But under certain circumstances (such as certain restrictions on the network topology and the movement of packets), this should be possible.

References

[1] A. Adams, J. Nicholas, and W. Siadak. Protocol Independent Multicast – dense mode (PIM-DM). Internet draft, February 2002.

[2] Y. Afek, B. Awerbuch, E. Gafni, Y. Mansour, A. Rosen, and N. Shavit. Slide – the key to polynomial end-to-end communication. *Journal of Algorithms*, 22(1):158–186, 1997.

[3] Y. Afek and E. Gafni. End-to-end communication in unreliable networks. In *Proc. of the 7th IEEE Symp. on Principles of Distributed Computing (PODC)*, pages 131–148, 1988.

[4] W. Aiello, B. Awerbuch, B. Maggs, and S. Rao. Approximate load balancing on dynamic and synchronous networks. In *Proc. of the 25th ACM Symp. on Theory of Computing (STOC)*, pages 632–641, 1993.

[5] W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosén. Adaptive packet routing for bursty adversarial traffic. In *Proc. of the 30th ACM Symp. on Theory of Computing (STOC)*, pages 359–368, 1998.

[6] M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, T. Leighton, and Z. Liu. Universal stability results for greedy contention-resolution protocols. In *Proc. of the 37th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 380–389, 1996.

[7] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput competitive online routing. In *Proc. of the 34th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 32–40, 1993.

[8] B. Awerbuch, P. Berenbrink, A. Brinkmann, and C. Scheideler. Simple routing strategies for adversarial systems. In *Proc. of the 42nd IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 158–167, 2001.

[9] B. Awerbuch and F. Leighton. A simple local-control approximation algorithm for multicommodity flow. In *Proc. of the 34th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 459–468, 1993.

[10] B. Awerbuch and F. Leighton. Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. In *Proc. of the 26th ACM Symp. on Theory of Computing (STOC)*, pages 487–496, 1994.

[11] B. Awerbuch, Y. Mansour, and N. Shavit. End-to-end communication with polynomial overhead. In *Proc. of the 30th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 358–363, 1989.

[12] B. Awerbuch and T. Singh. Online algorithms for selective multicast and maximal dense trees. In *Proc. of the 29 ACM Symp. on Theory of Computing (STOC)*, pages 354–362, 1997.

[13] T. Ballardie, P. Francis, and J. Crowcroft. Core Based Trees (CBT): An architecture for scalable inter-domain multicast routing. In *Proc. of the ACM SIGCOMM '93*, pages 85–95, 1993.

[14] E. Basturk, R. Engel, R. Haas, V. Peris, and D. Saha. Using network layer anycast for load distribution in the Internet. Technical Report RC 20938, IBM T.J. Watson Research Center, 1997.

[15] S. Bhattacharjee, M. Ammar, E. Zegura, V. Shah, and Z. Fei. Application-layer anycasting. In *Proc. of the IEEE INFOCOM '97*, 1997.

[16] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queueing theory. In *Proc. of the 28th ACM Symp. on Theory of Computing (STOC)*, pages 376–385, 1996.

[17] R. Carr and S. Vempala. Randomized metarounding. In *Proc. of the 32nd ACM Symp. on Theory of Computing (STOC)*, pages 58–62, 2000.

[18] S. Deering. Multicast routing in internetworks and extended LANs. In *Proc. of the ACM SIGCOMM '88*, pages 55–64, 1988.

[19] S. Deering. Host extensions for IP multicasting. RFC 1112, August 1989.

[20] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast – Sparse Mode (PIM-SM): protocol specification. RFC 2362, June 1998.

[21] P. Francis. Pip near-term architecture, 1994.

[22] D. Gamarnik. Stability of adversarial queues via fluid models. In *Proc. of the 29th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 60–70, 1998.

[23] D. Gamarnik. Stability of adaptive and non-adaptive packet routing policies in adversarial queueing networks. In *Proc. of the 31st ACM Symp. on Theory of Computing (STOC)*, pages 206–214, 1999.

[24] A. Goel. Stability of networks and protocols in the adversarial queueing model for packet routing. In *Proc. of the 10th ACM Symp. on Discrete Algorithms (SODA)*, pages 911–912, 1999.

[25] A. Goel, M. Henzinger, and S. Plotkin. Online throughput-competitive algorithm for multicast routing and admission control. In *Proc. of the 9th ACM Symp. on Discrete Algorithms (SODA)*, pages 97–106, 1998.

[26] J. Guyton and M. Schwartz. Locating nearby copies of replicated Internet servers. In *Proc. of the ACM SIGCOMM '95*, 1995.

[27] M. Henzinger and S. Leonardi. Scheduling multicasts on unit-capacity trees and meshes. In *Proc. of the 10th ACM Symp. on Discrete Algorithms (SODA)*, pages 438–447, 1999.

[28] R. Hinden. Simple internet protocol plus. RFC 1710, 1994.

[29] R. Hinden and S. Deering. Ip version 6 addressing architecture. RFC 1884, December 1995.

[30] R. Hinden and S. Deering. Internet protocol version 6 (IPv6) specification. RFC 2460, December 1998.

[31] J. Jetcheva and D. Johnson. Adaptive demand-driven multicast routing in multi-hop wireless ad hoc networks. In *Proc. of the ACM MobiHOC '01*, pages 33–44, 2001.

[32] W. Jia, D. Xuan, and W. Zhao. Integrated routing algorithms for anycast messages. *IEEE Communications Magazine*, 48:48–53, 2000.

[33] D. Katabi. The use of IP-anycast to construct efficient multicast trees. In *Proc. of the IEEE Global Internet'99*, 1999.

[34] D. Katabi and J. Wroclawski. A framework for scalable global IP-Anycast (GIA). In *Proc. of the ACM SIGCOMM*, pages 3–15, 2000.

[35] D. Kim, D. Meyer, H. Kilmer, and D. Farinacci. Anycast RP mechanism using PIM and MSDP. Internet draft, 2000.

[36] S.-J. Lee, W. Su, and M. Gerla. Wireless ad hoc multicast routing with mobility prediction. *Mobile Networks and Applications*, 6:351–360, 2001.

[37] J. Moy. Multicast extensions to OSPF. RFC 1584, March 1994.

[38] V. Park and J. Macker. Anycast routing for mobile services. In *Proc. of the 33rd Annual Conference on Information Sciences and Systems (CISS '99)*, 1999.

[39] C. Partridge, T. Mendez, and W. Milliken. Rfc 1546: Host anycasting service, November 1993.

[40] G. Robins and A. Zelikovsky. Improved Steiner tree approximation in graphs. In *Proc. of the 11th ACM Symp. on Discrete Algorithms (SODA)*, pages 770–779, 2000.

[41] E. Royer and C. Perkins. Multicast operation of the ad-hoc on-demand distance vector routing protocol. In *Proc. of the Mobicom '99*, pages 207–218, 1999.

[42] C. Scheideler and B. Vöcking. From static to dynamic routing: Efficient transformations of store-and-forward protocols. In *Proc. of the 31st ACM Symp. on Theory of Computing (STOC)*, pages 215–224, 1999.

[43] M. Schwartz, A. Emtage, B. Kahle, and B. Neuman. A comparison of Internet resource discovery approaches. *Computing Systems: The Journal of the USENIX Association*, 5(4):461–493, 1992.

[44] C. Tan and S. Pink. MobiCast: A multicast scheme for wireless networks. *Mobile Networks and Applications*, 5:259–271, 2000.

[45] P. Tsaparas. Stability in adversarial queueing theory. Master's thesis, Dept. of Computer Science, University of Toronto, 1997.

[46] B. Vöcking and S. Vempala. Approximating multicast congestion. In *Proc. of the ISAAC '99*, 1999.

[47] D. Waitzman, C. Partridge, and S. Deering. Distance vector multicast routing protocol. RFC 1075, November 1988.

[48] J. Wieselthier, G. Nguyen, and A. Ephremides. Algorithms for energy-efficient multicasting in static ad hoc wireless networks. *Mobile Networks and Applications*, 6:251–263, 2001.

A Bibliographical references

The notion of anycasting in the Internet was first introduced by RFC 1546 [39]. In this RFC, IP anycast is defined as a network service that allows a sender to access the nearest of a group of receivers that share the same anycast address, where "nearest" is defined according to the routing system's measure of distance. Usually, the receivers in the anycast group are replicas, able to support the same service (e.g. mirrored web servers). RFC 1546 proposes anycast as a means to discover a service location and provide host auto-configuration. For example, by assigning the same anycast address to a set of replicated FTP servers, a user downloading a file need not choose the best server manually from the list of mirrors. The user can use the anycast address to directly download the file from the nearest replica. The application of anycast to host auto-configuration, on the other hand, is exemplified in the assignment of the same anycast address to all DNS servers. In this case, a host that is moved to a new network need not be reconfigured with the local DNS address. The host can use the global anycast address to access the local DNS server anywhere. Since service discovery and auto-configuration are vital for dynamic networks, anycasting is also seen as an important mechanism to ensure robust support for networking services in mobile networks. Since its introduction, anycasting has received considerable attention in the systems community (see, e.g., [34, 33, 35, 32, 15, 38, 26, 14] and the references therein) and it has been adopted by all proposed successors of IPv4: Pip [21], SIPP [28], and IPv6 [29, 30]. However, to our surprise, it seems that anycasting has not been investigated by the theory community so far.

Multicasting is increasingly used by a number of applications to disseminate data to a group of receivers over networks, including multimedia services, software distribution, distributed interactive simulation and resource discovery [43]. Internet multicast protocols were first introduced in RFC 1112 [19]. Each multicast group is identified by a group address. Members join and leave multicast groups as they wish using the Internet Group Management Protocol (IGMP). IP packets addressed to a group address are delivered over a multicast tree to all group members. The sender does not need to know the membership of the group. Descriptions of current IP multicast protocols can be found in [13, 18, 1, 20, 37, 47]. Recently, a number of protocols have been suggested that allow to perform multicasting also in mobile ad hoc networks [41, 48, 44, 31, 36].

In the theory community, multicasting has mostly been studied in the context of selecting a least-cost tree for a single multicast group, also known as the Steiner tree problem. The currently best approximation ratio for general, weighted graphs is 1.55 [40].

Also the problem of minimizing the congestion of connecting several multicast groups via trees has been studied. Consider the situation that we are given a graph $G = (V, E)$ and subsets of vertices $S_1, \ldots, S_m \subseteq V$ that represent the multicast requests. The goal is to find a set of m trees so that the ith tree spans the vertices of the ith subset and the maximum number of trees that use any single edge is minimized. Vöcking and Vempala found a polynomial time algorithm for this problem with approximation ratio $O(\log n)$ [46]. Using convex decomposition techniques, this was improved by Carr and Vempala to a constant approximation ratio [17].

In another model, used for the development of online algorithms for multicasting, a request to join a multicast group specifies the group, the node that wants to join, the amount of the requested bandwidth, and the profit of being accepted. The multicast routing and admission algorithm can either reject this request or accept it and allocate the requested bandwidth along some path from the new node to the current tree associated with the requested multicast group. The algorithm is not allowed to allocate above link capacity. It is usually assumed that the number of multicast groups M is known in advance and that once a multicast group is established, it never leaves. The goal is to maximize the profit of the accepted requests.

Awerbuch, Azar and Plotkin show that if all the participants of a multicast group arrive at the same time, the accept/reject decision is for the whole multicast group, and the bandwidth needed by the group is only a logarithmic fraction of the link capacity, it is possible to achieve an $O(\log n)$ competitive ratio [7]. Their technique was extended by Awerbuch and Singh [12] to achieve a polylog competitive ratio for the case where there is no restriction on the bandwidth for a request and members of each multicast group arrive sequentially, i.e. the size and membership of the group is unknown upon its creation. Their algorithm can deal only with the non-interleaved case, i.e. when all the members of a particular multicast group arrive before a new group is created. Goel, Henzinger and Plotkin [25] generalize the algorithm in [12] to a randomized algorithm that can also handle the case that multicast groups are interleaved. It achieves a competitive ratio of $O((\log n + \log\log M)(\log n + \log M)\log n)$. They also show that this is close to optimum by presenting an $\Omega(\log n \log M)$ lower bound that holds for any randomized online algorithm against an oblivious adversary when M is much larger than the link capacities. It even holds for the case that the link capacities are much larger than the bandwidth requested by a single multicast. In addition, they present a simple proof showing that it is impossible to be competitive against an adaptive online adversary. Henzinger and Leonardi [27] further improved the techniques in [25] for the special case of trees and meshes.

B Proofs in Section 3

B.1 Proof of Theorem 3.2

First, we show that instability in the option set model implies instability in the anycast model, and then we show that instability in the anycast model implies instability in the unicast model.

Lemma B.1 *Let A be a natural, deterministic algorithm. Suppose that there is a network of N nodes and an adversary in the option set model so that A is not stable. Then there is a network of $N!$ nodes and an adversary in the anycast model so that A is also not stable here.*

Proof. To be able to simulate the option set model in the original adversarial model, we have to transform the adversarial strategy in the option model in a certain way.

First, we will remove all options from the option sets which are neither taken by the algorithm nor part of an active schedule. This will not change the distribution of the packets nor change the behavior of the algorithm. The remaining option sets only consist of one or two edges. We will perform the following additional transformations on these sets:

- If the adversary offers an option set with an edge (v, w) with $h(v) - h(w) < 0$ (which has to belong to an active schedule s in this case), we will split the time step into three substeps t_1, t_2, t_3. In time step t_1, the adversary offers the option set $\{(v, w), (w, v)\}$. Since we assume the algorithm to be natural, it is not allowed to send a packet from node v to node w. If it sends a packet from w to v, we will offer an edge from v to the destination at time step t_2, which has to be taken by the algorithm. At the end of step t_2, we will inject a new packet into node w that will take over the rest of schedule s. At time step t_3 the adversary offers the remaining edge (if any) of the option set to which (v, w) belonged.

  It is not difficult to check that this does not change the distribution of the packets, and that every packet still has a valid schedule. (Note that we are considering friendly adversaries.)

- If the adversary offers an option set of the form $\{(u, v), (v, w)\}$ with $h(u) \geq h(v) \geq h(w)$, we will insert the additional edge (w, u) into the option set. This edge will not be taken by the algorithm and does not change the distribution of the packets nor the competitive ratio of the algorithm.

Hence, at the end we are left with option sets of the form $\{(u, v)\}$ with $h(u) \geq h(v)$, or $\{(u, v), (v, u)\}$, or $\{(u, v), (v, w), (w, u)\}$ with $h(u) \geq h(v) \geq h(w)$.

Now we will transform the network in the option set model into a network in the original adversarial model. In the following, node 1 in the option set model denotes the destination. Let Π be the set of all permutations on $(1, 2, \ldots, N)$ and let $\mathcal{D} = \{D_1, \ldots, D_N\}$ with $D_i = \{\pi \in \Pi \mid \pi(i) = 1\}$. Π represents the node set of our new network and $\mathcal{D}$ represents the set of all possible anycast destination sets. Each node in Π has a buffer for every anycast set.

For each node $\pi \in \Pi$, its buffer for type i packets will be responsible for simulating the virtual node $\pi(i)$ in the option set model. In fact, we will design an adversarial strategy so that at the beginning of each step the heights of the packet buffers in the original adversarial model are the same as the heights of the virtual nodes they simulate in the option set model. (Thus, for example, for every node π the type i buffer with $\pi(i) = 1$ will always be empty.) This assumption is certainly true at the beginning of step 1, when all buffers are empty. Now we will show that if it is true at the beginning of step t, it will also be true at the beginning of time $t + 1$ when using the following strategy:

For every option set of the form

- $\{(u, v)\}$ or $\{(u, v), (v, u)\}$ with $h(u) \geq h(v)$: activate the edge (π_1, π_2) for all $\pi_1, \pi_2 \in \Pi$ in which the positions of u and v are permuted and everything else is identical. For example, edge $((x, u, y, v), (x, v, y, u))$ would be activated. Certainly, every node in Π is the starting point and endpoint of exactly one edge for each such option set. Consider any edge (π_1, π_2). Since everything is identical apart from the positions of u and v, all type i buffers in π_1 and π_2 with $\pi_1(i), \pi_2(i) \notin \{u, v\}$ have an identical height and therefore a difference in height of 0. If the algorithm in the option set model sends a packet from u to v, the algorithm in the adversarial anycast model would send a packet from the buffer in π_1 representing u to the buffer in π_2 representing v. Therefore, the corresponding heights in the option set model and in the adversarial anycast model remain equal after the edge activations.

- $\{(u, v), (v, w), (w, u)\}$: activate the edge (π_1, π_2) for all $\pi_1, \pi_2 \in \Pi$ in which v in π_2 is at the position of u in π_1, w in π_2 is at the position of v in π_1, and u in π_2 is at the position of w in π_1. Again, if the algorithm in the option set model sends a packet from, say, u to v, the algorithm in the adversarial anycast model would send a packet for the corresponding buffers. Hence, also here the corresponding heights in the option set model and in the adversarial anycast model remain equal after the edge activations.

It remains to consider packet injections and absorptions.

- If a packet is newly injected in node i in the option set model, we will inject a new packet in every packet buffer simulating i in every node in the adversarial model.
- Every edge use in the option set model in which a packet is absorbed causes packets to be absorbed for all corresponding edges in the adversarial anycast model, since all of them will be forwarded to a buffer simulating node 1 and therefore to one of their possible destinations.

Therefore, every action in the option set model can be simulated by a set of corresponding actions in the adversarial anycast model so that the heights of the virtual nodes and the corresponding packet buffers do not differ. This completes the proof of the lemma. □

Lemma B.2 *For any natural deterministic algorithm it holds: If it is not stable in the anycast model, then it is also not stable in the unicast model.*

Proof. Consider any natural deterministic algorithm A that is instable in the anycast setting. Let V be the set of nodes and let $\mathcal{D} = (D_1, \ldots, D_N)$ be the set of anycast sets used for this. To prove instability for A in the unicast model, we extend V to $V \cup \{d_1, \ldots, d_N\}$, where d_i is the new and only destination node for packets originally having destination set D_i. Let S be the strategy that caused instability for A in the anycast model. We simulate S until a packet of type i is supposed to reach one of its destination nodes D_i. Instead, we will offer now an edge to d_i. If this edge is taken by a packet of type i, we continue with the simulation. Otherwise, it follows from the definition of a natural algorithm that another packet must have been sent to d_i. This causes the total number of packets stored in the buffers in $\{d_1, \ldots, d_N\}$ to increase by one. Afterwards, we remove all packets from V by offering again and again edges to destinations d_i and start from the beginning with the simulation of S.

Thus, either we obtain a perfect simulation of S for the unicast case, in which case A will be instable, or we increase the number of packets in the buffers in $\{d_1, \ldots, d_N\}$ in every failed simulation attempt, which will also cause A to become instable. This completes the proof. □

B.2 Proof of Theorem 3.3

Algorithms which base their routing decision on exponential priority functions are very similar to the T-balancing algorithm. The main difference is that the potential of a node v is not equal to $\phi_v = \sum_{i=0}^{h_v} i$ but has the form $\phi_v = \sum_{i=0}^{h_v} \mathrm{e}^{\alpha \cdot i}$ with $\alpha > 0$. If an adversary offers an option set to the algorithm, the algorithm tries to optimize the potential based on its local information.

During the proof we will show that these decisions can involve the generation of new packets under certain circumstances and that these situations can be generated arbitrarily often. We assume that the nodes are sorted according to their heights with $h_{N-1} \geq h_{N-2} \geq \ldots \geq h_1 \geq h_0$ after every execution of a schedule and that node 0 is the destination node.

Lemma B.3 *If the height difference between node $(N-1)$ and node 1 is $h_{N-1} - h_1 \geq \frac{\ln(\mathrm{e}+1)}{\alpha} + 1$, a new packet can be injected without any packet leaving the system.*

Proof. We assume that the adversary injects a new packet into node 1, which stores the lowest number of packets and has the height h_1 before the injection of the packet. Then the adversary offers an option set with the two links $\{(N-1, 1), (1, 0)\}$. This option set is a valid schedule for the newly injected packet. The algorithm will choose link $(N-1, 1)$, if

$$\begin{array}{rrcl} & \mathrm{e}^{\alpha \cdot h_{N-1}} - \mathrm{e}^{\alpha \cdot (h_1+2)} & > & \mathrm{e}^{\alpha \cdot (h_1+1)} \\ \Rightarrow & \mathrm{e}^{\alpha \cdot h_{N-1}} & > & \mathrm{e}^{\alpha \cdot (h_1+2)} + \mathrm{e}^{\alpha \cdot (h_1+1)} \\ \Rightarrow & \mathrm{e}^{\alpha \cdot h_{N-1}} & > & (\mathrm{e}+1) \cdot \mathrm{e}^{\alpha \cdot (h_1+1)} \\ \Rightarrow & \alpha \cdot h_{N-1} & > & \ln(\mathrm{e}+1) + \alpha \cdot (h_1+1) \\ \Rightarrow & h_{N-1} - h_1 & > & \frac{\ln(\mathrm{e}+1)}{\alpha} + 1 \end{array} \qquad (1)$$

The lemma follows. □

The important observation from the previous lemma is that the necessary height difference between the two nodes does not depend on the actual height of these node. If it is always possible to create this fixed height difference for a given algorithm and a given number of packets in the system, then the algorithm is not stable. In the next lemma we will show that this is the case.

Lemma B.4 *Given a network with at least $\Delta + 1$ nodes it is possible to achieve a difference in height of at least Δ packets between the node with the highest number of packets and the non-destination node with the lowest number of packets without reducing the number of packets, or the algorithm is instable.*

Proof. Suppose we have a set S of Δ non-destination nodes with any distribution of packets. Then we will show how to design schedules so that the difference in height between the node with the highest number of packets and the node with the lowest number of packets is at least Δ without reducing the total number of packets in these nodes, or the algorithm is instable.

The strategy works as follows: Suppose that there are two nodes in S of equal height, say v and w. Then we inject a packet in v and offer the option set $\{(v, w)\}$. If the algorithm sends a packet, we offer the option set $\{(v, 0)\}$ and otherwise the option set $\{(w, 0)\}$. This ensures that the injected packet will have a schedule in any case and that the number of packets in S does not change. Furthermore, using the potential function $\phi_u = \sum_{i=1}^{h_u} i$, one can show that this operation increases the potential in S.

Hence, either the number of packets in S goes to infinity or there cannot be two nodes in S of the same height any more. In the latter case, this means that the highest and lowest node in S must have a difference of at least Δ. □

We now assume that we have a network with at least $\frac{\ln(e+1)}{\alpha} + 2$ nodes. From Lemma B.4 we know that in this case a height difference of at least $\frac{\ln(e+1)}{\alpha} + 1$ can be created.

After this, the adversary repeats the strategies in Lemma B.3 and Lemma B.4 again and again. With every iteration, the number of packets in the system will increase by one, which proves the theorem.

B.3 Proof of Theorem 3.5

We only prove the theorem for $B = 1$. For the proof it is sufficient to use two destinations, a and b. Packets destined for a (resp. b) will be called a-packets (resp. b-packets) and their buffers a-buffers (resp. b-buffers). Given a node v, the height of its a-buffer is denoted by $h_a(v)$ and the height of its b-buffer is denoted by $h_b(v)$.

According to [ABBS01] we know that for any $T \geq 0$ three nodes together with node a suffice to create an a-buffer in one of the nodes of height 2 (without using any b-packets). These three nodes can then easily be used to create in any one of the remaining non-destination nodes an a-buffer of height 2.

Consider the following assumption to be true: Suppose that we have a set U of n' nodes not used so far. Further, suppose we have a scheme that ensures that any one of these nodes can be taken and its a-buffer (resp. b-buffer) increased to a height of θ without using any one of the other unused nodes. Then it suffices to take two nodes out of U and use them so that any one of the remaining nodes can be taken and its b-buffer (resp. a-buffer) increased to a height of $2\theta - 1$ without using any other of the remaining nodes. This would lead to an increase in the buffer size reflected by the following recursive formula:

$$\theta_{i+1} = 2\theta_i - 1$$

Solving this formula results in $\theta_i = 2^i\theta_0 - (2^i - 1)$. Using three non-destination nodes to set $\theta_0 = 2$, it follows that $2(1 + 3 + 2i)$ nodes (for a and b each: 1 destination node, 3 non-destination nodes for lowest level, and $2i$ for further levels) would be sufficient to create a buffer of height $2^i \cdot 2 - 2(2^i - 1) \geq 2^i$. Choosing the largest possible i so that $2(4 + 2i) \leq n$ results in a buffer of height at least $2^{\lfloor n/4-2 \rfloor}$, which would prove the theorem.

Hence, it remains to prove the assumption above. Suppose that for any node in U, its a-buffer of b-buffer can be given a height of θ. For our construction we need the destinations a and b and two nodes $u_1, u_2 \in U$. Furthermore, let $U' = U \setminus \{u_1, u_2\}$ and let $v \in U'$ be the node in which we want to create a b-buffer of height $2(\theta - 1)$. Our construction repeats the following steps sufficiently often:

- Step 1: repeat
  - remove all packets from u_1 (if any) by offering edges to their destinations, use the hypothesis to set $h_a(u_1) = \theta$ ($h_b(u_1)$ will remain 0), and activate the edge (u_1, u_2)

until $h_a(u_2) = \theta - T$ (note that this will not affect the b-packets currently at u_2)

- Step 2: remove all packets from u_1 and use the hypothesis to set $h_b(u_1) = \theta$ ($h_a(u_1)$ will remain 0)
- Step 3: inject a b-packet in v and offer a schedule along the path (v, u_2, u_1, b)

It is not difficult to check that no b-packet injected in step 3 will ever reach its destination until the b-buffers have (at least) the following heights:

- $h_b(u_1) = \theta$
- $h_b(u_2) = 2\theta - T - 1$ (recall that $h_a(u_1) = 0$ after step 2 and $h_a(u_2) = \theta - T$)
- $h_b(v) = 2\theta - 1$ (recall that $h_a(v) - h_a(u_2) > T$ for a b-packet to be moved)

This completes the proof of the assumption and therefore the proof of the theorem.

B.4 Proof of Theorem 3.6

We will restrict our attention to deterministic algorithms. This allows us to deal with an adaptive adversary without losing the property that one can also use an oblivious adversary. However, the proof can also be generalized to randomized algorithms.

We will first do the proof for $B = 2$ and consider the case $B > 2$ afterwards. Suppose we have some deterministic algorithm A. For any node v and time step t, let $h_{v,t}$ denote the number of packets in v at step t. Furthermore, let the potential of a node v at step t by defined as $\phi_{v,t} = \sum_{i=1}^{h_{v,t}} i = \binom{h_{v,t}+1}{2}$ and the potential of the system at step t be defined as $\Phi_t = \sum_v \phi_{v,t}$.

Suppose that at some time step t there are two pairs of nodes, (v, w) and (x, y), representing at least three different nodes, with $h_{v,t} \geq h_{w,t}$ and $h_{v,t} - h_{w,t} = h_{x,t} - h_{y,t}$ Then the adversary chooses the strategy to inject a packet in v and a packet in x (if $v = x$, then it will only inject one packet in v). Afterwards, the adversary offers the option set $\{(v, w), (x, y)\}$. If A selects none of these edges, the adversary offers afterwards the option sets $\{(w, d)\}$ and $\{(x, y)\}$ and after that the option set $\{(y, d)\}$, where d is the destination. Then it would have been possible to send both injected packets to their destinations. However, no matter what A does with the last three option sets, it is easy to check that at the end the potential will be strictly higher than before.

So suppose w.l.o.g. that A selected (v, w) from the first option set to transmit a packet (the analysis for (x, y) is similar). Then the adversary offers the option sets $\{(v, d)\}$ and $\{(y, d)\}$. In this case, it would have been possible to send both injected packets to their destinations, had A selected the edge (x, y) first. However, no matter what A does with the last two option sets, it is easy to check that also here at the end the potential will be strictly higher than before.

Hence, as long as there are two pairs (v, w) and (x, y) of equal difference in height, the potential (and therefore ultimately also the number of packets in the system) strictly increases. If A is stable, then it must arrive at a state where there are no such pairs any more. Suppose that the highest height of a node is H in this case. Since the number of pairs of non-destination nodes is $\binom{N-1}{2}$, H must fulfill that

$$H + 1 \geq \binom{N-1}{2}$$

to make sure that there cannot be two of the pairs with height differences in $\{0, \ldots, H\}$. Hence, $H \geq (N - 2)^2/2$ for $B = 2$.

The bound for $B > 2$ follows from the fact that $B - 2$ packets can be injected to make the height difference of two pairs the same. Hence, the height difference of pairs has to be at least by a $B-1$ apart to make sure that this cannot happen, resulting in $H + 1 \geq (B - 1)\binom{N-1}{2}$.

C Proof of Theorem 4.1

Let N be the number of non-destination nodes in the option set model, and let node 0 represent the destination node. As mentioned previously, the height of node 0 is always 0, since any packet reaching 0 will be absorbed. For each of the remaining nodes we assume that it has H slots to store packets. The slots are numbered in a consecutive way starting from below with 1. Every slot can store at most one packet. After every step of the balancing algorithm we assume that if a node holds h packets, then its first h slots are occupied. The *height* of a packet is defined as the number of the slot in which it is currently stored. If a new packet is injected, it will obtain the lowest slot that is available after all packets that are moved to that node from another node have been placed.

Recall that for every successful packet in an optimal algorithm a schedule can be identified. A schedule $S = (t_0, (e_1, t_1), \ldots, (e_\ell, t_\ell))$ is called *active* at time t if $t_0 \leq t \leq t_\ell$. The *position* of a schedule at time t is the node at which its corresponding packet would be at that time if it is moved according to S. An edge in an option set is called a *schedule edge* if it belongs to a schedule of a packet. Suppose that we want to compare the performance of the balancing algorithm with an optimal algorithm that uses a buffer size of B. Then the following fact obviously holds.

Fact C.1 *At every time step, at most B schedules can have their current position at some node v.*

Next we introduce some further notation. We will distinguish between three kinds of packets: *representatives*, *zombies*, and *losers*. During their lifetime, the packets have to fulfill certain rules. (These rules will be crucial for our analysis. The balancing algorithm, of course, cannot and does not distinguish between these types of packets.) Every injected packet that does not have a schedule will initially be a zombie. Every other packet will initially be a representative. If a packet is injected into a full node, then the highest available loser will be selected to take over its role.

We want to ensure that a representative always stays with its schedule as long as this is possible. Two cases have to be considered for this when the adversary offers an option set with a schedule edge $e = (v, w)$:

1. A packet is sent along e: Then we always make sure that this packet is the representative belonging to e.
2. No packet is sent along e: If w has a loser, then the representative exchanges its role with the highest available loser in w. In this case we will also talk about a *virtual* movement. Otherwise, the representative is simply transformed into a loser. In this case, we will disregard the rest of the schedule (i.e. we will not select a representative for it afterwards and the rest of the schedule edges will simply be treated as non-schedule edges).

Furthermore, if a packet is sent along a non-schedule edge $e = (v, w)$, then we always make sure that none of the representatives is moved out of v but only a loser (which always exists if T is large enough).

The three types of packets are stored in the slots in a particular order. The lowest slots are always occupied by the losers, followed by the zombies and finally the representatives. Every zombie that happens to be placed in a slot of height at most $H - B$ will be immediately converted into a loser. Together with Fact C.1 these rules immediately imply the following fact.

Fact C.2 *At any time, the number of zombies and representatives stored in a node is at most B.*

Let $h_{v,t}$ be the *height* of node v (i.e. the number of packets stored in it) at the beginning of time step t, and let $h'_{v,t}$ be its height when considering only the losers. The *potential* of node v at step t is defined as $\phi_{v,t} = \sum_{j=1}^{h'_{v,t}} j = \binom{h'_{v,t}+1}{2}$ and the potential of the system at step t is defined as $\Phi_t = \sum_v \phi_{v,t}$. First, we study how the potential can change in a single step. Since schedules are not allowed to overlap, we arrive at the following fact.

Fact C.3 *Every option set contains at most one schedule edge.*

Hence, an option set can only have one or no schedule edge. To simplify the consideration of these two cases, we consider the option sets given in a time step one by one, starting with option sets without a schedule edge and always assuming the worst case concerning previously considered option sets. When processing these option sets, we always use the (worst case) rule that if a loser is moved to some node w, it will for the moment be put on top of all old packets in w. This will simplify the consideration of option sets with a schedule edge. At the end, we then move all losers down to fulfill the ordering condition for the representatives, zombies, and losers. This will certainly only decrease the potential. Using this strategy, we can show the following result.

Lemma C.4 *If $T \geq B + 2(\Delta - 1)$, then any option set that does not contain a schedule edge does not increase the potential of the system.*

Proof. Consider any fixed option set without a schedule edge. If no edge in the given option set is used by a packet, the lemma is certainly true. Otherwise, let $e = (v, w)$ be the edge along which a packet is sent. Note that in this case, $h_{v,t} - h_{w,t} > T$. If $T \geq B + 2(\Delta - 1)$, then even after $\Delta - 1$ removals of packets from v and the arrival of $\Delta - 1$ packets at w, there are still losers left in v, and the height of the highest of these is higher than the height of w. Hence, we can avoid moving any representative away from the position of its schedule and instead move a loser from v to w without increasing the potential. □

For option sets with a schedule edge (i.e. an edge that still has a representative associated with it), only a slight increase in the potential is caused.

Lemma C.5 *If $T \geq B + 2(\Delta - 1)$, then every option set that contains a schedule edge increases the potential of the system by at most $T + B + \Delta$.*

Proof. Consider some fixed option set with a schedule edge $e = (v, w)$. If e is selected for the transmission of a packet, then we can send the corresponding representative along e, which has no effect on the potential.

Otherwise, it must be that either $\delta_e = h_{v,t} - h_{w,t} \leq T$ or $\delta_e > T$ and another edge was preferred. In both cases, the representative R for e has to be moved virtually or transformed into a loser.

First of all, note that our rule of placing new losers on top of the old packets makes sure that the height of the representative in v does not increase. Furthermore, w can have at most $B + \Delta - 1$ representatives before moving R to w, and the lowest of these must have a height of at least $h_{w,t} - (B + \Delta - 1) + 1 = h_{w,t} - (B + \Delta) + 2$ (otherwise option sets have been taken that violate Fact C.2). Hence, if $h_{w,t} > B + \Delta - 1$, then it is possible to exchange places between R and a loser in w so that the potential increases by at most

$$h_{v,t} - (h_{w,t} - (B + \Delta) + 1) = \delta_e + B + \Delta - 1 \; . \tag{2}$$

If $\delta_e \leq T$, this is at most $T + B + \Delta$. This is also an upper bound for the case that there is no loser left in w, since this can only happen if $h_{w,t} \leq B + \Delta - 1$ and therefore $h_{v,t} \leq T + B + \Delta - 1$.

Otherwise δ_e might be quite big, but in this case there must be some other edge $e' = (v', w')$ that won against e because $\delta_{e'} \geq \delta_e$. Since $\delta_{e'} > T$, v' must have a loser even if $\Delta - 1$ losers already left v' and the maximum possible number of losers in v' was converted into representatives. In fact, similar to w above, the height of the highest of the remaining losers in v' must be at least $h_{v',t} - (B + \Delta) + 1$. On the other hand, w' can receive at most $\Delta - 1$ other packets before receiving the packet sent by e'. So the potential drop due to moving the highest available loser in v' to w' is at least

$$(h_{v',t} - B - \Delta + 1) - (h_{w',t} + \Delta) = (h_{v',t} - h_{w',t}) - B - 2\Delta + 1 \geq \delta_e - B - 2\Delta + 1 \; . \tag{3}$$

Subtracting (3) from (2), the increase in potential due to the given option set is at most

$$(\delta_e + B + \Delta - 1) - (\delta_e - B - 2\Delta + 1) = 2B + 3\Delta - 2 \; .$$

If $T \geq B + 2(\Delta - 1)$, this is at most $T + B + \Delta$. □

In addition to option sets, also injection events and the transformation of a zombie into a loser can influence the potential. This will be considered in the next two lemmata.

Lemma C.6 *Every deletion of a newly injected packet decreases the potential by at least $H - B$.*

Proof. According to Fact C.2, the highest available loser in a full node must have a height of at least $H - B$. Since the deletion of a newly injected packet causes this loser to be transformed into a representative or zombie, this decreases the potential by at least $H - B$. (Note that in case of a zombie, it might be directly afterwards converted back into a loser, but this will be considered in the next lemma.) □

If an injected packet is not deleted, this will initially not affect the potential, since it will either become a representative or a zombie. However, a zombie may be converted into a loser.

Lemma C.7 *Every zombie can increase the potential by at most $H - B$.*

Proof. Note that zombies do not count for the potential. Hence, the only time when a zombie influences the potential is the time when it is transformed into a loser. Since we allow this only to happen if the height of a zombie is at most $H - B$, the lemma follows. ◻

Now we are ready to prove an upper bound on the number of packets that are deleted by the balancing algorithm.

Lemma C.8 *Let σ be an arbitrary sequence of edge activations and packet injections. Suppose that in an optimal strategy, s of the injected packets have schedules and the other z packets do not. Let L be the average length of the schedules. If $H \geq B + 2(\Delta - 1)$, then the number of packets that are deleted by the balancing algorithm is at most*

$$s \cdot \frac{L(T + B + \Delta)}{H - B} + z \ .$$

Proof. First of all, note that only newly injected packets get deleted. Let p denote the number of option sets with a schedule edge and d denote the number of packets that are deleted by the balancing algorithm. Since

- due to Lemma C.4 option sets without a schedule edge do not increase the potential,
- due to Lemma C.5 every option set with a schedule edge increases the potential by at most $T + B + \Delta$,
- due to Lemma C.6 every deletion of a newly injected packet decreases the potential by at least $H - B$, and
- due to Lemma C.7 every zombie increases the potential by at most $H - B$,

it holds for the potential Φ after executing σ that

$$\Phi \leq p \cdot (T + B + \Delta) + z \cdot (H - B) - d \cdot (H - B) \ .$$

Since on the other hand $\Phi \geq 0$, it follows that

$$d \leq \frac{p \cdot (T + B + \Delta)}{H - B} + z \ .$$

Using in this inequality the fact that the average number of edges used by successful packets is at most L, and therefore the number of injected packets with a schedule, s, satisfies $s \geq p/L$, concludes the proof of the lemma. ◻

From Lemma C.8 it follows that the number of packets that are successfully delivered to their destination by the balancing algorithm must be at least

$$s + z - \left(s \cdot \frac{L(T + B + \Delta)}{H - B} + z \right) - H \cdot N = s \cdot \left(1 - \frac{L(T + B + \Delta)}{H - B} \right) - H \cdot N \ ,$$

where N is the number of (virtual) non-destination nodes. For $H \geq L(T + B + \Delta)/\epsilon + B$ this is at least

$$(1 - \epsilon)s - r$$

for some value r independent of the number of packets successful in an optimal schedule.

D Further applications of the T-balancing algorithm

In this section we describe how the T-balancing protocol could be applied to handle communication in more complex networks than adversarial networks consisting of direct point-to-point connections.

D.1 Wireless LANs

A wireless LAN (mobile users with fixed base stations) can be modelled as a bus network.

A *bus network* is a network in which nodes may not just be connected in a point-to-point fashion but several subsets of nodes share a single bus. In this case, the topology of a network can be modelled as a hypergraph $H = (V, E)$, i.e. each edge $e \in E$ can now be an arbitrary subset of E.

Suppose that in one step each hyperedge (or bus) can transport at most one packet from any endpoint to any other endpoint contained in it. Further, suppose that now the adversary can select an arbitrary hypergraph in each step. For each of these hyperedges, the problem would then be to decide which of the packets stored in its endpoints to send, and where to send it. This problem can again be modelled as an option set: we view each buffer as a virtual node, and the option to send a packet from buffer B to buffer B' can be modelled as a virtual edge between their virtual nodes. Thus, we arrive at exactly the same model as the one used above for routing in standard dynamic networks. This immediately yields the following result.

Corollary D.1 *Also for bus networks it holds: for any $\epsilon > 0$ and any $T \geq B + 2(\Delta - 1)$, the T-balancing algorithm is $1 + (1 + (T + \Delta)/B)L/\epsilon$-competitive.*

D.2 Virtual circuit networks

A *virtual circuit network* is a network in which every edge may now represent a virtual circuit in some underlying network. A example of a network standard based on virtual circuit switching are ATM networks. Also a peer-to-peer network can be viewed as a virtual circuit network if based on reliable in-order packet transfer which is guaranteed, for example, by TCP.

Virtual circuits must fulfill certain properties to give online algorithms the chance to get arbitrarily close to an optimal throughput. In order to motivate these properties, we start with a situation in which no online algorithm that does not duplicate packets can even get close to an optimal throughput.

A virtual circuit is said to have a *capacity of c* if it can absorb and hold up to c packets. Packets are delivered in FIFO order. That is, if $c + 1$ packets are given to that circuit, then the first of these packets must surface at the end of that link before the $(c + 1)$st packet can be absorbed by it. These rules give the following adversarial strategy to activate virtual circuits. Given a circuit of capacity c, the adversary has the right to determine when the link can absorb or deliver a packet under the condition that no more than c packets are kept by the link at any time.

Claim D.2 *If virtual circuits are allowed to have an unbounded capacity, then no pure online algorithm that does not duplicate packets manages to deliver an $\omega(1/n)$ (expected) fraction of the optimal throughput, even if all of them have the same destination.*

Proof. Consider n nodes numbered from v_1 to v_n. The adversarial strategy works as follows. First, it injects s packets with destination v_n into node v_1. Then it offers s times to absorb a packet for each circuit (v_1, v_i), $i \in \{2, \ldots, n-1\}$. No matter which online algorithm is chosen, there must be a circuit (v_1, v_j), $j \in \{2, \ldots, n-1\}$, for which at most $s/(n-2)$ offers are used by the algorithm (either with certainty or in the expected sense) to deliver a packet to that link. Afterwards, none of the circuits (v_1, v_i) will be offered any more to v_1. The adversary will only allow those packets to surface that were given to link (v_1, v_j). Finally, it offers s circuits to node v_n. Hence, if the online algorithm had sent all packets along (v_1, v_j) and then along (v_j, v_n), all packets could have reached their destination. However, at most $s/(n-2)$ (with certainty or in the expected sense) will manage to do so. $\square$

Hence, there has to be an upper bound on the capacity of the circuits. Let us call this parameter C. We need some further assumptions for our methods to work: whereas it might take a while to send a data packet from one node to another, we assume that control information can be exchanged instantly. In our case, this will be a reasonable assumption, since the information we need to exchange only consists of a few bits. Under this assumption we can extend the balancing algorithm in the following way.

For each possible virtual circuit (v, w) and each destination d, we introduce a buffer of size C, $B_{(v,w),d}$, at w. All packets stored in these buffers and all packets that are in transit in (v, w) are considered to have a potential of 0. Every time (v, w) is willing to absorb a packet, we treat this as an edge activation in our standard network model: If no packet is selected by the T-balancing algorithm to cross (v, w), then nothing is done also here. If a packet P with destination d is selected by the T-balancing algorithm to cross (v, w), then we pass P on to (v, w) and inform w about this (here we need the ability to exchange control information instantly) so that w can move one packet out of $B_{(v,w),d}$ (if available) to the buffer that P is supposed to reach in w. This ensures that P can be treated as if instantly moving from v to w, which is important to guarantee that P will be sooner in w then the packet that may have been possibly sent by a best possible strategy.

Since the capacities of the circuits are bounded by C, it is easy to see that none of the buffers $B_{(v,w),d}$ will ever overflow. Hence, our simulation technique above does not cause deletions of packets. Suppose now that we use the same analytical technique as for the standard model. We distinguish between several cases to make clear that the new model does not cause any problem for this technique. For this we consider any time point in which a circuit is willing to accept a packet as an option set representing an edge activation:

- Suppose that we have an option set without a schedule edge. If no packet is selected for a transmission, there will be no potential increase. Otherwise, suppose the selected packet, P, has destination d. If the corresponding buffer $B_{(v,w),d}$ is not empty, then our simulation technique simulates exactly a standard edge activation, and therefore we will experience also here no potential increase. Otherwise, the potential of P is simply taken away from the system, which also causes the potential not to increase.
- Suppose that we have an option set with a schedule edge. If the representative is selected for a transmission and there is a packet in $B_{(v,w),d}$, then we can simulate its traversal. Otherwise, we simply convert the representative into a loser and send it along (v, w). Since this loser will have a potential of 0, this will not increase the potential. If the representative is not selected, then we move it virtually, using the potential drop caused by the selected packet (or if no packet is moved, the fact that the difference in height is at most T) to compensate the virtual movement.
- If a representative or zombie is deleted or a zombie is transformed into a loser, the same results apply as in the standard model.

Hence, we arrive at the following result.

Theorem D.3 *For virtual networks it holds: for any $\epsilon > 0$ and any $T \geq B + 2(\Delta - 1)$, the T-balancing algorithm is $1+(1+(T+\Delta)/B)L/\epsilon$-competitive if an additional space of $C(n-1)$ is available in each node to perform our simulation strategy.*

E Proof of Theorem 5.1

For the proof, we will slightly modify the definition of the height of a packet or node. The height of a multicast packet is now the height of its highest slot plus o_v, where $o_v = (H - B) - \lfloor (H - B)/\mu(v) \rfloor \cdot \mu(v)$. The height of a node is the height of its highest multicast packet.

Obviously, $o_v \in [0, \mu(v)]$ for all v. Hence, when comparing the new height $h_{v,t}$ with the original height $h^o_{v,t}$, it holds that $h_{v,t} \in [h^o_{v,t}, h^o_{v,t}+\mu(v))$. Given a regular edge (u, v), let $\delta_{(u,v)} = \mu(u)h_{u,t} - \mu(v)h_{v,t}$ and given a split edge (u, v, w), let $\delta_{(u,v,w)} = \mu(u)h_{u,t} - \mu(v)h_{v,t} - \mu(w)h_{w,t}$. Then we obtain the following lemma.

Lemma E.1 *Consider the multicast T-balancing algorithm. If a packet is sent along a regular edge (u, v), then $\delta_{(u,v)} > \mu(u)(T - \mu(u))$. If a packet is send along a split edge (u, v, w), then also $\delta_{(u,v,w)} > \mu(u)(T - \mu(u))$.*

Proof. Consider a regular edge (u, v). If a packet is sent along that edge, then $\mu(u) \cdot h^o_{u,t} - \mu(v) \cdot h^o_{v,t} > \mu(v) \cdot T$. Since $\mu(u) \geq \mu(v)$ and $h_{x,t} \in [h^o_{x,t}, h^o_{x,t} + \mu(x))$ for all nodes x, $\delta_{(u,v)} > \mu(u)(T - \mu(u))$. The proof for a split edge follows the same reasoning. ☐

Recall the notation used in the proof of Theorem 4.1. Suppose that we want to compare the performance of the balancing algorithm with an optimal algorithm that uses a buffer size of B. Recall that every packet unit is associated with at most one successful delivery. Hence, the following fact obviously holds again.

Fact E.2 *For every time step and every node v, at most $B/\mu(v)$ schedules can have their current position at v.*

We will again distinguish between three kinds of packets: *representatives*, *zombies*, and *losers*. Every newly injected multicast packet that has a schedule for k of its d destinations will originally be called a (d, k)-representative. If $k \geq 1$, it will also simply be called a *representative*, and otherwise a *zombie*. If a (d, k)-representative is injected into a buffer that is able to store that packet, then it will be stored on top of the other packets. Those k of its packet units representing deliverable destinations are denoted *representative units* and the rest are *zombie units*. If the buffer is already full, then the highest available multicast packet representing a loser will be taken to transform it into a (d, k)-representative.

If a zombie happens to have a height of at most $H - B$, it will be converted into a loser unit. (Note that our choice of o_v ensures that in all nodes v a packet can have a height of *exactly* $H - B$. This will be important for our potential analysis below.) Throughout the routing we will make sure that representatives are always at the position prescribed by their schedule. Since a schedule can have the form of a tree, a representative may split into many representatives (and zombies) during this process. To ensure that representatives are always at the position prescribed by their schedule, we will apply the same virtual movement trick as for our anycast approach: if a representative at some node v is not moved or split along its schedule edge, then it exchanges its role with a loser at each endpoint of the edge. If for one of the endpoints this is not possible (i.e. there is no loser with which to exchange roles), then nothing changes for that endpoint and the rest of the schedule for that endpoint will be disregarded (i.e. we will not select representatives for it afterwards). This means that we will simply convert the corresponding packet units in v into loser units without converting packet units at the new position into representative units. Note that it can happen that at some point some representative becomes a $(d, 0)$-representative or zombie. In this case, it will remain at its current node until it is transformed into a loser.

There are special rules about how packets are stored in nodes. Representatives always occupy the highest slots in the nodes, followed by the zombies and the losers. Recall that $h_{v,t}$ is the height of node v (i.e. the number of packet units in v plus o_v) at the beginning of time step t, and $h'_{v,t}$ is its height when considering only the losers. The height of a multicast packet P (and each of its packet units) in some node v is defined as the height of its highest packet unit. The weight of P is the sum of the heights of its packet units and therefore $\mu(v)$ times its height. The *potential* of node v at step t is defined as the sum of the weights of all losers in it, or formally,

$$\phi_{v,t} = \sum_{j=1}^{(h'_{v,t}-o_v)/\mu(v)} \mu(v) \cdot (j \cdot \mu(v) + o_v)$$

and the potential of the system at step t is defined as $\Phi_t = \sum_v \phi_{v,t}$. First, we study how the potential can change in a single step. Since schedules are not allowed to overlap, we arrive at the following fact.

Fact E.3 *Every option set contains at most one schedule edge.*

Hence, an option set can only have one or no schedule edge. To simplify the consideration of these two cases, we consider the changes in the option sets that take place in a time step one by one, starting with option sets without a schedule edge and always assuming the worst case concerning previously considered option sets. For option sets without a schedule edge, the following lemma holds.

Lemma E.4 *If $T \geq B + 4D\Delta$, then any option set that does not contain a schedule edge does not increase the potential of the system.*

Proof. Consider some fixed option set. First, note that the amount of multicast packets that arrive at a node via an edge during a time step can be up to 2Δ, since both incoming packets and split events can contribute Δ new packets to a buffer. The maximum number of multicast packets that can leave a node is still at most Δ.

If no edge in the given option set is used by a packet, the lemma is certainly true. Otherwise, we distinguish between regular and split edges.

Suppose that the selected edge $e = (v, w)$ is a regular edge. Note that $\mu(v) \geq \mu(w)$. In the worst case, v might send up to $\Delta - 1$ multicast packets and w might receive up to $2\Delta - 1$ multicast packets before considering e. In this case, the height of v might be reduced to $h_v - \mu(v) \cdot (\Delta - 1)$ and the height of w might be increased to $h_w + \mu(w) \cdot (2\Delta - 1)$ before considering e. Since there can be only up to B packet units in v belonging to representatives or zombies, the highest loser in v must have a height of at least $h_v - (B + \mu(v)(\Delta - 1))$, which is greater than o_v since $T \geq B + 4\mu(v)\Delta$. Thus, taking the highest loser away from v reduces its potential by at least $\mu(v) \cdot (h_v - (B + \mu(v)(\Delta - 1)))$. On the other hand, placing the loser in w increases its potential by at most $\mu(w) \cdot (h_w + \mu(w)(2\Delta - 1))$ (which is also an upper bound for the case that the loser may be deleted due to a full node). Hence, the overall potential increase is at most

$$\begin{aligned} -\mu(v) \cdot (h_v - B - \mu(v)(\Delta - 1)) + \mu(w) \cdot (h_w + \mu(w) \cdot 2\Delta) \\ \leq \quad -\delta_e + \mu(v) \cdot (B + \mu(v)(3\Delta - 1)) \end{aligned}$$

where δ_e is the difference in the weights of v and w. Since, according to Lemma E.1, $\delta_e > \mu(v)(T - \mu(v))$ and $T \geq B + 4D\Delta$, this is at most 0.

Otherwise, suppose that $e = (v, w, w')$ is a split edge. Note that $\mu(v) \geq \mu(w) + \mu(w')$. Analogous to the case when e is regular, even after $\Delta - 1$ removals of packets from v there are still losers in v. The height of the highest of these is at least $h_v - B - \mu(v)(\Delta - 1)$, and when moving it to w and w' the resulting losers will have a height of at most $h_w + \mu(w) \cdot 2\Delta$

and $h_{w'} + \mu(w') \cdot 2\Delta$, where h_v, h_w and $h_{w'}$ are the original heights of the nodes. Hence, the overall potential increase is at most

$$\begin{aligned} -\mu(v) \cdot (h_v - B - \mu(v)(\Delta - 1)) + \mu(w) \cdot (h_w + \mu(w) \cdot 2\Delta) + \mu(w') \cdot (h_{w'} + \mu(w') \cdot 2\Delta) \\ \leq \quad -\delta_e + \mu(v) \cdot (B + \mu(v)(3\Delta - 1)) \ . \end{aligned}$$

Therefore, also in this case no increase in the potential will occur. □

Next we consider option sets that contain a schedule edge.

Lemma E.5 *If $T \geq B + 4D\Delta$, then every option set that contains a schedule edge increases the potential of the system by at most $D(T + B + D \cdot \Delta)$.*

Proof. Consider some fixed option set E with a schedule edge e. We distinguish between two cases. If e is selected, then we can send the corresponding representative along e, which has no effect on the potential.

Otherwise, the representative units in a representative have to be moved virtually. Suppose first that e is a regular edge (v, w). On the one hand we can assume that previous executions of option sets do not increase the height of the representative in v, because (virtual) movements of representatives do not affect the old representatives in v and when moving a loser we always consider the worst case that it is placed above v. Since on the other hand at most Δ packets can leave w and the representative will be stored in one of the B highest positions of w at the end of the step (if it is possible to exchange it with a loser in w), the virtual movement causes an increase in the potential by at most

$$\begin{aligned} \mu(v) \cdot h_{v,t} - \mu(w) \cdot (h_{w,t} - B - \mu(w) \cdot \Delta) &= \mu(v)(h^o_{v,t} + o_v) - \mu(w)((h^o_{w,t} + o_w) - B - \mu(w)\Delta) \\ &\leq \delta^o_e + \mu(v)^2 + \mu(v)(B + \mu(v) \cdot \Delta) \end{aligned}$$

where δ^o_e is the original δ_e used in the multicast T-balancing algorithm. We have here $B + \mu(w)\Delta$ instead of $B + \mu(w)(\Delta - 1)$ for the anycast case, since we do not allow here to exchange the representative with a zombie in w. If e is a split edge (v, w, w'), a similar argument implies that the virtual movement causes an increase in the potential by at most

$$\begin{aligned} &\mu(v)h_{v,t} - \mu(w)(h_{w,t} - B - \mu(w)\Delta)) - \mu(w')(h_{w',t} - B - \mu(w')\Delta) \\ &\quad \leq \quad \mu(v)(h^o_{v,t} + o_v) - \mu(w)(h^o_{w,t} + o_w - B - \mu(w)\Delta) - \mu(w')(h^o_{w',t} + o_{w'} - B - \mu(w')\Delta) \\ &\quad \leq \quad \delta^o_e + \mu(v)^2 + \mu(v)(B + \mu(v) \cdot \Delta) \ . \end{aligned}$$

Hence, in both cases the virtual movement increases the potential by at most

$$\delta_e + \mu(v)^2 + \mu(v)(B + \mu(v)\Delta) \ . \tag{4}$$

If $\delta^o_e \leq \mu(v)T$, this is at most $\mu(v)(T + B + \mu(v)(\Delta + 1))$. A similar upper bound also holds for the case that there is no loser left in w and/or w', since this can only happen if $h^o_{w,t}$ (resp. $h^o_{w',t}$) is at most $B + \mu(w)(\Delta + 1)$ (resp. $B + \mu(w')(\Delta + 1)$). Otherwise δ^o_e might be quite big, but in this case there must have been some other edge e' that won against e. For this edge e', it must hold that $\delta^o_{e'} \geq \delta_e$. If e' is a regular edge (x, y), then the potential drop by moving the highest available loser in x to y is at least

$$\begin{aligned} &\mu(x)(h_{x,t} - B - \mu(x)(\Delta - 1)) - \mu(y)(h_{y,t} + \mu(y) \cdot 2\Delta) \\ &\quad = \quad \mu(x)(h^o_{x,t} + o_x - B - \mu(x)(\Delta - 1)) - \mu(y)(h^o_{y,t} + o_y + \mu(y) \cdot 2\Delta) \\ &\quad \geq \quad (\mu(x) \cdot h_{x,t} - \mu(y) \cdot h_{y,t}) - \mu(x)^2 - \mu(x)(B + \mu(x)(3\Delta - 1)) \\ &\quad \geq \quad \delta^o_e - \mu(x)^2 - \mu(x)(B + \mu(x)(3\Delta - 1)) \ . \end{aligned} \tag{5}$$

If e' is a split edge (x, y, y'), then the potential drop by moving the highest available loser in x to y and y' is at least

$$\begin{aligned} &\mu(x)(h_{x,t} - B - \mu(x)(\Delta - 1)) - \mu(y)(h_{y,t} + \mu(y) \cdot 2\Delta) - \mu(y')(h_{y',t} + \mu(y') \cdot 2\Delta) \\ &\quad \geq \quad (\mu(x) \cdot h^o_{x,t} - \mu(y) \cdot h^o_{y,t} - \mu(y') \cdot h^o_{y',t}) - \mu(x)^2 - \mu(x)(B + \mu(x)(3\Delta - 1)) \\ &\quad \geq \quad \delta^o_e - \mu(x)^2 - \mu(x)(B + \mu(x)(3\Delta - 1)) \ . \end{aligned} \tag{6}$$

Subtracting (6) from (4), the increase in potential due to E is at most

$$(\delta_e^o + D(B + D \cdot (\Delta + 1))) - (\delta_e - D(B + D \cdot 3\Delta)) \leq D(2B + D(4\Delta + 1)) \ .$$

If $T \geq B + 4D\Delta$, this is at most $D(T + B + D \cdot \Delta)$. □

The only cases we omitted so far is the fact that in addition to option sets also the deletion of a non-loser and the transformation of a zombie into a loser can influence the potential. This will be considered in the next two lemmata.

Lemma E.6 *Every deletion of a non-loser unit will decrease the potential by at least $H - B$.*

Proof. Deletions of multicast packets only occur if all slots are occupied. If this happens to a non-loser in some node v, always the highest available loser in v is used to convert it into the deleted non-loser. From the definition of B and o_v we know that the height of this loser must be at least $H - B$. Hence, the potential decrease caused by the deletion of a non-loser is at least $H - B$ for each of its units. (Note that in the case of a zombie, it might be directly afterwards converted back into a loser, but this will be considered in the next lemma.) □

Lemma E.7 *Every zombie unit can increase the potential by at most $H - B$.*

Proof. Since a zombie (unit) is only transformed into a loser (unit) if it has a height of at most $H - B$, the lemma follows. □

Now we are ready to prove an upper bound on the number of packet units that are deleted by the balancing algorithm.

Lemma E.8 *Let σ be an arbitrary sequence of edge activations and packet injections. Suppose that in an optimal strategy, s of the requested deliveries can be achieved and z can not. Let L be the average number of edges used for the deliveries. If $H > B$, then the number of packet units that are deleted by the balancing algorithm is at most*

$$s \cdot \frac{L \cdot D(T + B + D\Delta)}{H - B} + z \ .$$

Proof. Let p denote the number of option sets with a schedule edge and d denote the number of packet units corresponding to a successful delivery in the optimal solution that are deleted by the balancing algorithm. Since

- due to Lemma E.4 option sets without a schedule edge do not increase the potential,
- due to Lemma E.5 every option set with a schedule edge increases the potential by at most $D(T + B + D \cdot \Delta)$,
- due to Lemma E.6 every deletion of a non-loser unit decreases the potential by at least $H - B$, and
- due to Lemma C.7 every zombie unit increases the potential by at most $H - B$,

it holds for the potential Φ after executing σ that

$$\Phi \leq p \cdot D(T + B + D\Delta) + z \cdot (H - B) - d \cdot (H - B) \ .$$

Since on the other hand $\Phi \geq 0$, it follows that

$$d \leq \frac{p \cdot D(T + B + D\Delta)}{H - B} + z \ .$$

Using in this inequality the fact that the average number of edges used by successful deliveries is most L, and therefore the number successful deliveries, s, satisfies $s \geq p/L$, concludes the proof of the lemma. □

From Lemma E.8 it follows that the number of successful deliveries achieved by the balancing algorithm must be at least

$$s + z - \left(s \cdot \frac{L \cdot D(T + B + D\Delta)}{H - B} + z\right) - H \cdot N = s \cdot \left(1 - \frac{L \cdot D(T + B + D\Delta)}{H - B}\right) - H \cdot N \ ,$$

where N is the number of (virtual) non-destination nodes. For $H \geq L \cdot D(T + B + D\Delta)/\epsilon + B$ this is at least

$$(1 - \epsilon)s - r$$

for some value r independent of the number of packets successful in an optimal schedule.