Anycasting in Adversarial Systems: Routing and Admission Control

Baruch Awerbuch⋆1, André Brinkmann⋆⋆2, and Christian Scheideler[1]

[1] Department of Computer Science, Johns Hopkins University, 3400 N. Charles Street, Baltimore, MD 21218, USA, {baruch,scheideler}@cs.jhu.edu

[2] Heinz Nixdorf Institute and Department of Electrical Engineering, University of Paderborn, 33102 Paderborn, Germany, brinkman@hni.upb.de

Abstract. In this paper we consider the problem of routing packets in dynamically changing networks, using the anycast mode. In anycasting, a packet may have a set of destinations but only has to reach any one of them. This set of destinations may just be given implicitly by some anycast address. For example, each service (such as DNS) may be given a specific anycast address identifying it, and computers offering this service will associate themselves with this address. This allows communication to be made transparent from node addresses, which makes anycasting particularly interesting for dynamic networks, in which redundancy and transparency are vital to cope with a dynamically changing set of nodes. However, so far not much is known from a theoretical point of view about how to efficiently support anycasting in dynamic networks. This paper formalizes the anycast routing and admission control problem for arbitrary traffic in arbitrary dynamic networks, and provides first competitive solutions. In particular, we show that a simple local load balancing approach allows to achieve a near-optimal throughput if the available buffer space is sufficiently large compared to an optimal algorithm. Furthermore, we show via lower bounds and instability results that allowing admission control (i.e. dropping some of the injected packets) tremendously helps in keeping the buffer resources necessary to compete with optimal algorithms low.

Keywords: Adversarial routing, anycasting, online algorithms, load balancing, dynamic networks

1 Introduction

This paper studies the problem of supporting anycasting in adversarial networks. The notion of anycasting was first standardized in RFC 1546 [16]. 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.

⋆ 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).

The user can simply use the anycast address to directly download the file from the nearest server. In order to aid host auto-configuration, all DNS servers may be given the same anycast address. In this case, a host that is moved to a new network need not be reconfigured with the local DNS address. The host can simply use the global anycast address to access a local DNS server. Service discovery and auto-configuration are seen as vital components of protocols for dynamic networks, and therefore anycasting is seen as a crucial mechanism to ensure robust support for networking services in mobile networks. Since its introduction, anycasting has received considerable attention in the systems community and it has been adopted by all proposed successors of IPv4 (e.g. Pip, SIPP, and IPv6). However, to our surprise, it seems that anycasting has not been investigated by the theory community so far.

Since in highly dynamic networks it may be very hard to predict which may be the nearest server belonging to some anycast address, it seems to be a formidable problem to efficiently support anycasting in dynamic networks, especially for those that are under adversarial control. However, we demonstrate in this paper that even if both the network and the packet injections are under adversarial control, distributed routing strategies can be found for anycasting with a close to optimal throughput. Thus, in principle, anycasting can even be supported in such networks as mobile ad-hoc 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 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?
- *Queueing*: 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?

The study of adversarial models was initiated, in the context of *queueing alone*, by Borodin et al. [12]. Other work on queueing includes [6, 13–15, 17, 18]. 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 so that they do not overload the system. Hence, it remains to find the right queueing discipline (such as furthest-to-go) to ensure that the number of packets in the system (resp. the time needed by packets to reach their destination) is bounded. However, the bounds on the buffer size given in these papers to avoid dropping any packet usually depend on the network size and are sometimes unrealistically high. This motivated Aiello et al. [5] to study the throughput performance of queueing disciplines under the assumption that the routing buffers have a fixed size (i.e. that is independent of network parameters), using an adversary that can inject an unbounded number of packets. In this case, of course, a queueing discipline cannot guarantee the delivery of every injected packet. So the goal is rather to find a queueing strategy whose throughput is as close as possible to a best possible throughput. Aiello et al. show among other results that there are queueing disciplines that are guaranteed to achieve an $\Omega(1/(d \cdot m))$ fraction of the best possible throughput achievable with the same buffer size, where m is the number of edges and d is the longest path injected by the adversary. This upper bound and their lower bound of $O(\sqrt{m})$ for the line that holds for arbitrary greedy protocols seem to indicate that online protocols cannot compete well with best possible protocols when using the same buffer size.

The study of adversarial models was initiated, in the context of *routing*, by Awerbuch, Mansour and Shavit [11] and further refined by [4, 7, 9, 10, 14]. 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. [4] 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. [7] 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.

With the exception of [5], the weakness of the adversarial models above is that they assume that the adversary never overloads the system with packets. In static networks this may be 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 can handle not only the routing and queueing part but also packet-level admission control, i.e. dropping packets from either input or intermediate buffers.

Also, we note that all of the above work on adversarial queueing and routing only considered the *unicasting* mode (every packet has a single destination). We consider the more general anycasting mode, using a very general adversarial model that gets rid of somewhat artificial restrictions of previously suggested models for dynamic networks. In fact, the only limiting assumptions left in our model are that packets are of atomic nature (i.e. they cannot be split or compressed) and that packets cannot be killed by the adversary. Thus, our upper bounds also apply to other adversarial routing and queueing models suggested so far.

Finally, we 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 [1–4, 7, 9, 10] for various routing purposes. Our achievement is to demonstrate that balancing even works for anycasting. Also, we use a much more general adversarial network model then was used in previous papers, and we consider the admission control problem.

In order to state our analytical results, we need some notation.

1.2 The anycast routing and admission control model

First, we describe the basics of 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.

The adversary does not only control the topology of the network but also the injection of packets. Each anycast packet is 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. Each time an 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. 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. We assume that both the optimal and the online algorithm are allowed to allocate one buffer in each node for each type of packet. Thus, if there are b different anycast addresses, then a node can allocate up to b different buffers. This will simplify the comparison. However, our competitive results also work if every node only has a single buffer (or a fixed number of buffers). In this case, the buffer overhead for our online algorithm has to be multiplied by b.

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 (i.e. each buffer can store up to B packets), 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.

In the following, B will always mean the buffer size of an optimal routing algorithm.

1.3 New results

Our new results are arranged in two sections. In Section 2, we 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*, that achieves the following result:

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. For $B \geq \Delta$ and $T = O(B)$, this boils down to a competitive ratio of $O(L/\epsilon)$. The result is sharp up to a constant factor.

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 can be delivered 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.

Some of the proofs are only sketched due to space limitations. Please see [8] for details.

2 Adversarial Anycasting

Let $h_{v,a,t}$ denote the amount of packets in the buffer for anycast address a in node v at the beginning of time step t. $h_{v,a,t}$ will also be called the *height* of the corresponding buffer. The maximum height a buffer can have is denoted by H.

We now present a simple balancing strategy that extends the balancing strategies used by Aiello et al. [4] and Awerbuch et al. [7] by a rule for deleting packets. In every time step $t \geq 1$ the T-balancing algorithm performs the following operations.

1. For every edge (v, w), determine the anycast address a with maximum $h_{v,a,t} - h_{w,a,t}$ and check whether $h_{v,a,t} - h_{w,a,t} > T$. If so, send a packet for a 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 buffer 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 buffers 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.

Let L 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 edges leaving or leading to a node that can be active at any time. 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.

Theorem 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. To simplify the analysis, we prove the competitive ratio for a more general model than our anycast model, called *option set model*.

In the option set model, we have a set of nodes V' with a single buffer each, and all injected packets want to go to the same destination $d \in V'$. The adversary can inject an arbitrary number of packets in each time step. Also, it can activate an arbitrary collection of edge sets $E_1, \ldots, E_k \subseteq V' \times V'$, called *option sets*, in each step as long as every node $v \in V' \setminus \{d\}$ has an incoming or outgoing edge in at most Δ many sets. For each option set E_i, the algorithm is allowed to use only one edge in E_i for the transmission of a packet.

This model is indeed more general than our anycast model.

Lemma 1. *Any algorithm that is c-competitive in the option set model is also c-competitive in the anycast model.*

Proof. For this it suffices to show how to transform the anycast model into the option set model. Suppose that A is the set of all anycast addresses. Then we define $V' = V \times A$, i.e. each buffer in the original model represents a node in the option set model. Each edge $e = (v, w)$ that is activated in the anycast model can then be represented as option set $E_e = \{((v, a), (w, a)) \mid a \in A\}$. Since all packets reaching their destination buffers in the anycast model are absorbed, we can view all of these buffers as a single node in the option set model without affecting the throughput. □

Hence, in the following we only work with the option set model.

Let N be the number of non-destination nodes in the option set model, and let node 0 represent the destination node. 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.

For each successful packet in an optimal algorithm, a schedule can be identified. A schedule $S = (t_0, (e_1, t_1), \ldots, (e_\ell, t_\ell))$ consists of a sequence of movements by which the injected packet P is sent from its source node to the destination. 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 was available in some option set at time t_i for all $1 \leq i \leq \ell$. Certainly, no two schedules are allowed to use the same option set at the same time. 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 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 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 has a schedule (i.e. that will be delivered by the optimal algorithm) will initially be a representative. Every other injected packet will initially be a zombie. The goal of a representative is to stay with its schedule, and the goal of a zombie is to stay at a slot of height more than $H - B$. Whenever this cannot be fulfilled, the packet is transformed into a loser. Together with Fact 1, this implies the following fact.

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

If a packet is injected into a full node, then the highest available loser will be selected to take over its role (Fact 2 implies that this is always possible if $H > B$).

Our goal for the analysis is to ensure that a representative always stays with its schedule as long as this is possible. That is, each time the schedule moves, the representative tries to move with it, and otherwise it tries to stay at the current position of the schedule. This implies the following rules for a representative R when the adversary offers an option set containing one of its schedule edges $e = (v, w)$:

1. A packet is sent along e: Then we always select R to be moved along e.
2. No packet is sent along edge 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.

Let $h_{v,t}$ be the *height* of node v (i.e. the number of packets stored in the buffer represented by v) 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, every option set contains either 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. Also, 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 2. *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 3. *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, there are two ways for w to lose losers before considering e: either an unused schedule edge to w forced a virtual movement of a representative to w, or a used non-schedule edge from w forced to move a loser out of w. Let s be the number of edges with the former property and ℓ be the number of edges with the latter property. If w had r representatives (and zombies) at the beginning of t, then it must hold that $r + s - (\Delta - \ell) + 1 \leq B$ to ensure that at the end of step t, w has at most B representatives (the $+1$ is due to e). Thus, $r + s + \ell \leq B + \Delta - 1$. Hence, if there is still a loser left in w when considering e, the highest of these must have a height of at least $h_{w,t} - (B + \Delta - 1)$. Therefore, if $h_{w,t} \geq B + \Delta$, then it is possible to exchange places between R and a loser in w so that the potential increases by at most

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

If $\delta_e \leq T$, this is at most $T + B + \Delta$. If $h_{w,t} < B + \Delta$, then it may be necessary to convert R into a loser. However, since $h_{v,t} - h_{w,t} \leq T$, this increases the potential also by at most $T + B + \Delta$.

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

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

Subtracting (2) from (1), 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 4. *Every deletion of a newly injected packet decreases the potential by at least $H - B$.*

Proof. According to Fact 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 5. *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 6. *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 2 option sets without a schedule edge do not increase the potential,
- due to Lemma 3 every option set with a schedule edge increases the potential by at most $T + B + \Delta$,
- due to Lemma 4 every deletion of a newly injected packet decreases the potential by at least $H - B$, and
- due to Lemma 5 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 6 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. □

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

Theorem 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 (i.e. slot number) 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 $1-\epsilon$. □

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.

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 buffer pair (h_1, h_2) for each 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 demonstrating 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. 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 never to deliver a 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. Hence, the adversary can prevent packets from reaching the good destination, although there may be plenty of opportunities, had the good packets been chosen. On the other side, the adversary will never offer an edge directly to the dead destination. Hence, no packet will ever get delivered. □

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 so that when using an optimal algorithm, only a bounded number of packets are in transit at any time without deleting any? We show that also in this case some natural algorithms have severe problems if packets cannot be dropped.

Theorem 3. *If the adversary is allowed to inject packets for more than one destination, then the adversary can force the T-balancing algorithm to store by a factor of $\Theta(2^{n/4})$ more packets in a buffer than for an optimal algorithm.*

Proof. (Sketch) For the proof it is sufficient to use two destinations, a and b, and to set $B = 1$. 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 $h_b(v)$.

We show the theorem by complete induction. Suppose that we can construct a scheme using $2(5+2i)$ nodes with two nodes $v_i^{(a)}$ and $v_i^{(b)}$ so that $h_a(v_i^{(a)}) \geq H_i$ and $h_b(v_i^{(a)}) = 0$, and $h_a(v_i^{(b)}) = 0$ and $h_b(v_i^{(b)}) \geq H_i$, where $H_i = 2^i \max\{4T, 3\} - (2^i - 1)(2T + 1)$. Then we can show that 4 more nodes suffice to identify nodes so that the hypothesis above also holds for $i + 1$. The basic idea is to create "copies" u_a and u_b of $v_i^{(a)}$ and $v_i^{(b)}$ and then to inject schedules for a-packets (resp. b-packets) with path $(v_{i+1}^{(a)}, u_b, u_a, a)$ (resp. $(v_{i+1}^{(b)}, u_a, u_b, b)$). □

The theorem implies together with the results in [7] that only for the case that we have a single destination, the T-balancing algorithm without a dropping rule can be space-efficient under friendly adversaries.

What about other natural algorithms studied in the literature such as algorithms based on exponential priority functions (e.g. [10])? A routing algorithm is called *stable* if the number of packets in transit does not grow unboundedly with time. 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 (suggested in the proof of Theorem 1) instead of the original unicast model, which is much more difficult to handle.

Theorem 4. *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.*

Proof. (Sketch) We only show how to get from the anycast to the unicast model. See [8] for details.

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. 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. Then, 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. □

The theorem allows us to show the following result.

Theorem 5. *Natural routing algorithms which are based on exponential priority functions are not stable.*

Proof. By algorithms with exponential priority function we mean algorithms using the potential drop $f(h_1, h_2) = (\phi(h_1) + \phi(h_2)) - (\phi(h_1 - 1)) + \phi(h_2 + 1))$ with $\phi(h) = \sum_{i=1}^{h} e^{\alpha \cdot i}$ for some $\alpha > 0$ to determine the priority of a packet movement.

We will show that this rule can cause packets not to be delivered under certain circumstances, and that these situations can be generated arbitrarily often. We assume that the nodes are sorted according to their heights with $h_{N-1} \geq h_{N-2} \geq ... \geq h_1 \geq h_0$ and that node 0 is the destination node.

Lemma 7. *If the height difference between node $(N-1)$ and node 1 is $h_{N-1} - h_1 \geq \frac{\ln(e+1)}{\alpha}$, 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 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, however, will choose link $(N-1, 1)$ if $e^{\alpha \cdot h_{N-1}} - e^{\alpha \cdot (h_1+1)} > e^{\alpha \cdot h_1} - e^{\alpha}$, which is true if $h_{N-1} - h_1 > \frac{\ln(e+1)}{\alpha}$. □

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 nodes. 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 8. *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. Pick a set S of Δ non-destination nodes, and consider the following strategy: 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 $\frac{\ln(e+1)}{\alpha} + 1$ nodes. From Lemma 8 we know that in this case a height difference of at least $\frac{\ln(e+1)}{\alpha}$ can be created.

After this, the adversary repeats the strategies in Lemma 7 and Lemma 8 again and again. With every iteration, the number of packets in the system will increase by one, which proves the theorem. □

The proof of the theorem immediately implies the following result.

Corollary 1. *Natural routing algorithms which always prefer the buffer with the largest number of packets are not stable.*

We conjecture that any (natural) online algorithm is either instable or requires an exponential buffer size to be stable under friendly adversaries, which would imply together with Theorem 1 that the ability to drop packets can tremendously improve the performance of routing algorithms.

4 Conclusions and Open Problems

In this paper we presented a simple balancing algorithm for anycasting 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 other protocols that can achieve a lower space overhead in general?

References

1. 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.
2. Y. Afek and E. Gafni. End-to-end communication in unreliable networks. In *PODC '88*, pages 131–148, 1988.
3. W. Aiello, B. Awerbuch, B. Maggs, and S. Rao. Approximate load balancing on dynamic and synchronous networks. In *STOC '93*, pages 632–641, 1993.
4. W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosén. Adaptive packet routing for bursty adversarial traffic. In *STOC '98*, pages 359–368, 1998.
5. W. Aiello, R. Ostrovsky, E. Kushilevitz, and A. Rosén. Dynamic routing on networks with fixed-size buffers. In *SODA '03*, 2003.
6. M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, T. Leighton, and Z. Liu. Universal stability results for greedy contention-resolution protocols. In *FOCS '96*, pages 380–389, 1996.
7. B. Awerbuch, P. Berenbrink, A. Brinkmann, and C. Scheideler. Simple routing strategies for adversarial systems. In *FOCS '01*, pages 158–167, 2001.
8. B. Awerbuch, A. Brinkmann, and C. Scheideler. Anycasting and multicasting in adversarial systems. Technical report, Dept. of Computer Science, Johns Hopkins University, March 2002. See http://www.cs.jhu.edu/∼scheideler.
9. B. Awerbuch and F. Leighton. A simple local-control approximation algorithm for multicommodity flow. In *FOCS '93*, 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 *STOC '94*, pages 487–496, 1994.
11. B. Awerbuch, Y. Mansour, and N. Shavit. End-to-end communication with polynomial overhead. In *FOCS '89*, pages 358–363, 1989.
12. A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queueing theory. In *STOC '96*, pages 376–385, 1996.
13. D. Gamarnik. Stability of adversarial queues via fluid models. In *FOCS '98*, pages 60–70, 1998.
14. D. Gamarnik. Stability of adaptive and non-adaptive packet routing policies in adversarial queueing networks. In *STOC '99*, pages 206–214, 1999.
15. A. Goel. Stability of networks and protocols in the adversarial queueing model for packet routing. In *SODA '99*, pages 911–912, 1999.
16. C. Partridge, T. Mendez, and W. Milliken. Rfc 1546: Host anycasting service, November 1993.
17. C. Scheideler and B. Vöcking. From static to dynamic routing: Efficient transformations of store-and-forward protocols. In *STOC '99*, pages 215–224, 1999.
18. P. Tsaparas. Stability in adversarial queueing theory. Master's thesis, Dept. of Computer Science, University of Toronto, 1997.