Computer Communications 146 (2019) 201–218

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Design & analysis of a distributed routing algorithm towards Internet-wide geocast

Bernd Meijerink [a,*], Mitra Baratchi [b], Geert Heijenk [a]

[a] University of Twente, Enschede, The Netherlands
[b] Leiden Institute of Advanced Computer Science (LIACS), Leiden University, Leiden, The Netherlands

ARTICLE INFO

Keywords:
Geocast
Routing
Geographic routing

ABSTRACT

Geocast is the concept of sending data packets to nodes in a specified geographical area instead of nodes with a specific address. This forwarding method is valuable in situations where any number of nodes inside a geographic area need to be reached, such as vehicular networking scenarios. To facilitate large scale geocast, a wired network geographic routing algorithm is needed that can route packets efficiently towards a destination area. Our goal is to design an algorithm that can deliver shortest path tree like geographic forwarding while relying purely on distributed data without central knowledge. In this paper we present and implement two algorithms for geographic routing. One algorithm is based purely on distance-vector data. Another, more complicated algorithm is based on path data. We show that our purely distance-vector-based algorithm can come close to the number of links used by a shortest path tree when a small number of routers are present in the destination area. We also show that our path-based algorithm can come close to the link usage of a shortest path tree in almost all geocast situations. We also show that the algorithms converge relatively quickly following link drops.

1. Introduction

Due to the increasing use of connected, possibly autonomous, vehicles and 'smart' devices there is an increasing need for Internet-wide geographically scoped communications [1]. Geographically scoped communications, or geocast, would allow devices in a specific area to be addressed without the need for tracking IP addresses or administration concerning multicast groups.

Geocast was first introduced by Navas and Imielinski [2]. It is a transmission method where packets are sent to a location or area rather than an IP-address. It can be seen as a one-to-many or many-to-many system like multicast with the main difference that devices receive packets based on their location rather than a subscription model.

While geocast might seem similar to multicast in some ways, multicast-like routing will not be sufficient in a geocast environment for scalability reasons. Multicast routing algorithms are mostly designed to route packets towards predefined multicast groups that are relatively static. Receivers also need to subscribe to specific groups, which can prove problematic in situations where either the sender(s) or receivers have a high rate of change. Geocast packets on the other hand will have to be routed to a set of routers based on an sender-specified destination area that could contain several or even zero routers. The destination area can take any shape or size which in turn makes predefined areas problematic.

Our work mostly focuses on the possible applications of geocast in the vehicular network domain. Possible applications would be location dependent weather updates, traffic alerts and information to assist autonomous driving. In the vehicular networking scenario a functioning geocast solution would at least require location-aware base stations, or Road Side Units (RSUs) that are aware of the area they serve. Location aware-systems are important for many safety applications related to vehicular networks [3]. One example of a vehicular networking geocast scenario would be notifying cars on a specific highway of an accident or traffic jam, using geocast to only send the message to that street. An example of such a situation is depicted in Fig. 1. A RSU forwards a notification of a traffic accident to multiple vehicles on the road the accident took place on. Other examples would be a central traffic control centre that needs to send messages to all vehicles in a specific area, or a location-specific weather warning for road users in a specific region.

Previous proposals for geographic routing are mostly application layer based, an example being extended DNS [4]. There are two main downsides to such an approach. They have high overhead due to lookup operations and are less resilient to change. We propose an alternative approach to the problem: Implementing geocast on the network layer. A network layer implementation would allow us to

* Corresponding author.
E-mail addresses: bernd.meijerink@utwente.nl (B. Meijerink), m.baratchi@liacs.leidenuniv.nl (M. Baratchi), geert.heijenk@utwente.nl (G. Heijenk).

https://doi.org/10.1016/j.comcom.2019.07.025
Received 19 November 2018; Received in revised form 22 July 2019; Accepted 27 July 2019
Available online 30 July 2019

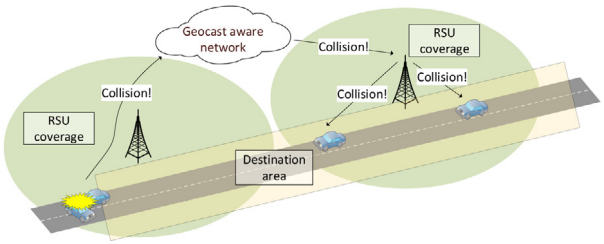


Fig. 1. Geocast traffic accident example.

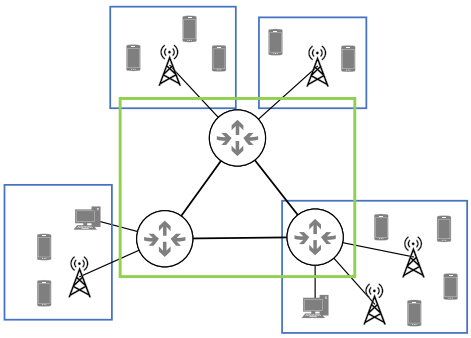


Fig. 2. Area of interest. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

use information already available due to unicast routing. The system would also be more resilient due to not relying on the availability of certain servers. Embedding geocast in the network itself will also allow it to route around problems in the network. It would also enable such a system to possibly scale to the entire Internet. Enabling Internet-wide geocast could potentially allow fine grained geographically scoped message transmission for everyone. The main benefit would be that sending hosts on the network do not require any sort of geographical information, they can just send a geocast packet to the router serving them. Possible use cases of network layer geocast range from localized weather reports without clients reporting their location to safety information transmitted to vehicles.

To provide an efficient geocast solution, the underlying routing protocol will need to take geographic information into account. Traditional routing methods such as unicast or multicast routing have drawbacks in the geocast scenario in that additional signalling or even over the top solutions are needed to enable geographic routing. Unicast routing has the obvious drawback of sending one packet per destination. This would lead to communications overhead when a large number of devices is present in the destination area. On the other hand, the per-packet processing overhead is minimal as unicast routing is well understood and optimized. Multicast routing seems like a better fit as it already supports one-to-many communications. The main drawback for geographically scoped communication is that most multicast routing solutions depend on subscription messages. In a geocast solution, routers will need to report their coverage area and evaluate which routers cover the destination area of a packet. Another drawback would be the requirement of predefined destination areas, as it would have to be known which multicast group covers which area.

We set out to design a distributed algorithm, as the communication overhead needed for a centralized approach would lead to scalability problems for the system. Routers should not depend on some central authority or need full network knowledge. This last requirement prevents us from constructing a least cost (Steiner) tree, as this would require full network knowledge. We will cover this last point in greater detail in Section 2.3.

For efficient geographic routing we need a routing algorithm in which geographical areas are central to packet routing. A geographic routing algorithm will need to efficiently route packets that have a geographic destination to all routers that (partially) cover that area. We specifically refer to coverage instead ‘being in the area’, as the important thing is that devices connected to the router are in the destination area. The most important aspect is the ability to route a packet to multiple destinations using the lowest number of hops possible, without sending duplicate packets over the same link.

We use two area definitions in our geocast system: Coverage area and destination area. *Coverage Area* defines the geographic area that is covered by a router, devices in this area can be reached through this router. Coverage areas of routers may overlap or even be identical, for example multiple providers servicing the same area. *Destination area* refers to the geographic area to which a packet is sent. This area does not need to be identical to the coverage area of a router, instead routers should calculate if the destination overlaps with their coverage area.

In this paper we specifically focus on routing between routers (as enclosed by the green rectangle in Fig. 2), we do not consider end-hosts. We consider the router that initially receives a packet as the source. The actual source might be an end-host connected to that router or some other network. We consider a router a destination if the intersection of one or more of its coverage areas and the destination area is not empty. A router’s coverage area (as enclosed by the blue rectangles in Fig. 2) is defined by one or more rectangles that enclose all end-hosts connected to the router and the area covered by all wireless access devices that are connected to the router.

The main research question we answer in this paper is: How can we efficiently route geocast packets within a network? The main contribution of our work is threefold:

- We design and implement a geographical routing algorithm using purely distance-vector based information,
- We design and implement an efficient geographical routing algorithm based on path information,
- We validate and evaluate the proposed algorithms and their implementation on a large set of real world network topologies, both in terms of link cost and convergence aspects.

Whereas the first routing algorithm mentioned above is very efficient with respect to link cost, the second algorithm is computationally very simple, at the cost of a somewhat higher link cost for the resulting distribution tree and can be used in cases where network capacity is less important than router resources.

The remainder of this paper is structured as follows: In the next section we will describe previous work in the area, including our own work on geographic addressing. Section 3 will describe the algorithms we have designed to perform geographic routing. We evaluate our algorithms in Section 4. In Section 5 we describe implementations of our algorithms, of which we evaluate the convergence properties during link loss in Section 6. Finally we draw our conclusions in Section 7.

2. Previous work

In this section, we will describe previous work done on geocast and geographic routing. We start with describing related work in the wireless domain, moving towards work on geocast in a wired setting. We will conclude the section with an overview of our own work on geographic addressing and forwarding tree evaluation, in which we will explain our choice for a shortest path tree.

2.1. Related work

Geocast was initially introduced by Navas and Imielinski for wired networks [2]. Their approach relied on special routers that know their location and forward packets based on the destination point, circle or

polygon. Routers are connected hierarchically: A router that covers a certain area connects to 'lower' routers that cover smaller areas within that coverage area. Routers can calculate the intersect of destination and their coverage using the GPS coordinates. Downsides of this approach are the hierarchical router requirement, the need for routers to perform area intersection calculations and the variable length of the addressing (points, circles, or polygons).

In later work from the same authors they studied improved routing cost [5] by approximating the destination/coverage area intersection. They have also studied alternate approaches based on addressing predefined locations [6].

Most work on the topic of geocast has been done in the wireless ad-hoc network context, and especially the VANET case. Overviews of such routing protocols and underlying mechanism can be found in [7,8] and [9]. In most of these protocols the location of forwarding nodes is tightly coupled with the destination of a packet, a next hop node will generally be in the direction of the destination area. The correlation between the position of the next hop node and the location of the destination area does not necessarily exist in a fixed wired network situation. Especially in situations were a network serves several access networks, there might be very little correlation between the forwarding routers and the actual destination area. On the other hand, the fixed wired environment is usually mostly static, this enables route distribution to be effective over long distances.

A well-known example of a geographic routing protocol for ad-hoc networks is GeoTORA [10]. When a node in the network needs to geocast a message it broadcasts a query with the request for the destination nodes. The destination nodes send a message back, allowing the original requesting node to know the forwarding hop towards the geocast area. The mobile nature of these ad-hoc networks makes this kind of signalling a necessity to reach any sort of efficiency. We do not have this problem in wired networks, allowing for the possibility of route distribution beforehand.

Another example is Greedy Perimeter Stateless Routing [11]. In this algorithm, traffic is routed to nodes that are located closer to the destination area than the transmitting node. This approach is seen often in geographic routing solutions for ad hoc networks. The downside for a wired environment is that the location of routers is not necessarily correlated with the direction a packet needs to travel to reach its destination.

An interesting grid based ad-hoc routing system is the Grid Location Service (GLS) [12]. This system has nodes keep track of each others location in a distributed manner, where nodes closer to a node are more likely to know its location. The geographic routing layer of GLS addresses nodes based on their current location and uses a distance-vector protocol with two hop knowledge to route packets to their destination. This protocol allows nodes to lookup the location and send packets to a specific other node. This requires the location of all possible destinations to be knows somewhere in the network, while we would like to address any number of nodes in a given area preferably without any knowledge (of the potentially large number) of nodes in this area.

There are off-course algorithms for multicast routing such as Protocol Independent Multicast (PIM). These could be used in some capacity for geocast routing but they do have some drawbacks. PIM Dense Mode (PIM-DM) relies on an initial flooding stage where routers that are not subscribers send a prune message back to their forwarding neighbour [13]. We would ideally like to not have this behaviour in our geocast system as we believe the number of destination areas that might be addressed in a short time could be very large. Alternatively, PIM Sparse Mode (PIM-SM) relies on an initial Rendezvous Router that routes packets before a shortest path tree is established [14]. Due to the large number of varying geocast destinations and the overhead caused by the Rendezvous Router we believe this approach would not be feasible.

Another, application layer, approach to geocast is to use DNS to resolve geographical areas to an IP addresses by extending the DNS [4].

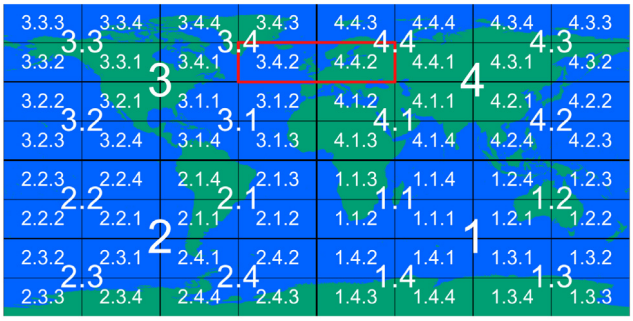


Fig. 3. Geographic addressing up to level 3.

When the eDNS server is queried for a certain area, it returns the IP addresses of all entries in that region. The eDNS was designed for VANET scenarios, so it would only have to return a list of RSUs in the target area. Scaling the system to track the movements of all vehicles to also allow geocasting from multiple networks at the same time was later found to be somewhat feasible [15]. For a truly Internet-wide deployment such a system would need to scale significantly. DNS delegation would also be complicated if updates are to be distributed through the network in a relatively short time.

2.2. Geographic addressing

The addressing system our routing system is based on is described in our previous work [16]. Our addressing scheme transforms a location bounding box defined by its maximum and minimum latitude and longitude into a format that can fit inside an IPv6 address. This is achieved by dividing the area of the world into 4 rectangles, and in turn subdividing these rectangles until the desired area size is reached. We number the rectangles in such a way that neighbouring rectangles with different 'parents' share the same number. An example of this method mapped to a world map with a depth of 3 levels can be seen in Fig. 3. Using this hierarchical structure we can address increasingly smaller blocks for more precision down to areas with a size of 7 by 3.5 cm at the equator.

An example of this scheme applied to the world with rectangles up to a depth of 3 can be seen in Fig. 3. In this figure we have highlighted an area encompassing much of northern Europe and the UK. Individually these can be addressed as 3.4.2 and 4.4.2, but taken together we can address them as [3,4].4.2. We can map this representation to a binary format by using 4 bits per depth level, where we set the bits in the order 1,2,3,4 with a bit set to 1 indicating that rectangle is included. The example used before would translate to 0011.0001.0100. By using 4 bits per level we can combine any neighbouring rectangles into the area we wish to describe. Using this representation we could use this geographic address as a destination IPv6 address. Given that we use 4 bits per level and assuming the first 2 bytes of the IPv6 address are needed to distinguish geocast from unicast or multicast, we are left with $112/4 = 28$ levels. This gives us a worst case rectangle size of 7 by 3.5 cm on the equator.

This addressing scheme effectively decouples the actual geographic location and the address itself. Routers or other forwarding systems do not need to have any knowledge of their geographic location, they forward based on the underlying addressing system. Our addressing system allows the matching of a destination area and a coverage area in a manner similar to prefix matching. This is achieved by encoding each level in a block of 4 bits. We can perform a bitwise AND operation on the destination and coverage area to find if there is overlap. Overlap is found if there is at least one bit shared in each 4 bit group, up until the length of the shortest address (which corresponds to the largest area).

This addressing method also allows multiple overlapping and/or neighbouring areas to be aggregated into one or more larger area(s).

Potentially this could allow a network that covers a single region to aggregate its area into a single address.

The addressed area does have to be symmetrical. This might cause the addressed area to be greater than the actual destination area. We have, however, shown that our addressing scheme will allow a packet to efficiently get close to its destination with minimal overhead [16]. Once the packet reaches the final router in its destination, a more accurate distribution system might have to take over (for example, one specific for vehicular networks).

2.3. Shortest path vs. Steiner tree

In an ideal world we would always transmit packets using the least cost tree (Steiner tree) from source to destinations. By definition this is the best forwarding tree that can be established based on chosen metrics such as cost or delay. There are however several drawbacks to such an approach that have real world implications. Some of these drawbacks are: The requirement of full network knowledge and high computational overhead.

A least cost tree routing method might work in smaller networks where the cost of maintaining full network knowledge in each router is not too high. In larger networks or even on an Internet-wide scale this approach is infeasible due to the communications and processing overhead involved in maintaining a full network graph and establishing or maintaining a Steiner tree for every (source, destination) area combination. Another problem is the before-mentioned computational overhead of the Steiner tree. The Steiner tree problem is NP-complete [17], and the cost grows exponentially with the number of routers in a network.

The downsides of the Steiner tree make it infeasible for larger networks. A shortest path tree (a tree consisting of all shortest paths from the source to all destinations) from the source to the destination area does not have these limitations. A single shortest path can be computed using a distance-vector algorithm that does not require full network knowledge and has significantly less computational overhead compared to a Steiner tree.

For multicast it has been shown that shortest path trees can be preferable to Steiner trees in both fixed [18] and wireless ad-hoc [19] networks. In our previous work, we have evaluated which type of forwarding tree would be most efficient for the geocast scenario [20]. We have shown that a shortest path tree has minimal additional cost in overall link usage compared to a perfect Steiner tree in a situation where destinations are geographically close.

3. Algorithm design

In this section, we will describe the process we have followed to design our geographic routing algorithms. We will start by describing the path notation we will use in the rest of the paper. We will then discuss the simplest algorithm possible that will achieve our stated goal: flooding. In the following subsections we will add conditions to build increasingly complex forwarding rules, resulting in our distance-vector-based algorithm. Following that, we will briefly analyse the performance of this algorithm. We continue by describing our path-based forwarding algorithm, followed by short sections on possible link state approaches and hierarchical routing.

We define the primary goal for our geographic routing algorithm as follows: to deliver a message addressed to a certain area to all routers that cover that area with minimal cost. We will use the hop count (which for a tree we define as the total number of transmission used per packet to reach all destinations) as our cost metric, with a lower number of hops being better. To achieve our goals we choose to target a shortest path tree from the source to all routers that cover (advertise) the destination area. We also have the secondary design goals of limiting the processing overhead and using a system where no per destination signalling is needed. Our algorithms are designed around the assumptions that all links in the network are symmetrical in both connectivity and cost.

Our addressing system allows the aggregation of coverage areas into a single area. This allows a network to advertise its coverage to the outside world as a single address. Our path-based proposal might be suitable for inter-network routing like BGP using the aggregated address for a network. However, in this paper we mostly focus on a single network scenario to show that our system can function.

3.1. Path notation

We will use paths in the network to better explain and eventually build our routing algorithm on. We denote a (shortest) path $p_{n,m}$ through the network from a node n to another node m.

$$p_{n,m} = n \rightarrow \ldots \rightarrow m$$

We define the length of a path $l = |p_{n,m}| - 1$ as the number of nodes it contains minus one. A path has a minimum length of 1 as $|p_{n,m}| \geq 2$ (assuming $n \neq m$), and can have an arbitrary number of nodes ($x_a, x_b, \ldots$ where $x_a \neq x_b$, $x \neq n$ and $x \neq m$) between n and m. We denote the kth node in such a path as $p_{n,m}(k)$. For example, $p_{n,m}(1) = n$ in the path above. Note that we will use paths in the description of our distance-vector approaches, even though this approach only uses cost information. The path information used in the algorithm is always limited to the next and previous hop, information that would also be available to a purely distance-vector based algorithm as it can be inferred from the advertised cost information.

3.2. Flooding

The most straightforward approach to geocast is to simply flood the network with all traffic. This will lead to each packet traversing each link in the network at least once.

Flooding would guarantee that packets are delivered to all addressed destinations. The downside is that there would be significant overhead, especially in larger networks with few routers in the addressed destination area. The total per packet transmission cost would be constant, equal to the number of links in the network assuming routers would ignore duplicate packets coming in on different links.

The transmission overhead is large for such an approach, each link in the network would transmit the packet at least once. The processing overhead is limited to checking if a certain packet has already been processed before, or checking if an incoming packet has been received on the shortest path link to the source, provided that unicast routing info is present.

3.3. Distance vector

Using every link in the network is not very efficient; we would like to construct a perfect shortest path tree through the network. We can improve on the flooding algorithm by introducing shortest path knowledge to the routers using a distance-vector approach. A distance-vector algorithm (like those used for unicast) would give all routers knowledge of the shortest-path next hop to all other routers. Coupled with geographic coverage information for these routers, this could enable geocast in a network.

For the following algorithms we assume the routers have the following knowledge:

- Coverage area for every router in the network.
- The cost and next hop for reaching every other router in the network.

A Router receives a packet that has a geocast address in its destination field, as described in Section 2.2. The router then checks this address against the coverage area of the route advertisements it has received. Packets are forwarded to the routers that have overlapping coverage with the destination area. We denote the set of routers with coverage overlap of the destination area as D. We will now describe 4 distance-vector algorithms in order of increasing complexity that use only the cost to other routers to forward packets to their destinations.

3.3.1. DV Algorithm 1

For our first attempt we simply try to limit the flooding in the network to the 'direction' of the destinations. We use the term direction loosely here, as the actual geographical location of links and routers does not necessarily correspond to the area they cover. In this simple approach, each router will forward packets it receives on its shortest path link to each of the destinations, except for the link the packet was received on.

We define our forwarding function as $fn(n, D)$, which returns for a router, n, the set of neighbours to forward the packet to based on the destination set, D.

$$fn(n, D) = \forall m \in neighbours(n) : \exists dst \in D : m = p_{n,dst}(2)$$

In this function the path $p_{n,dst}$ is the shortest path from the current router n to a single destination dst. By definition this path passes through a next hop m in the position $p_{n,dst}(2)$, that may be the same as the destination dst (in which case the path would have a length of 1). With distance-vector information each router is aware of at least two routers on such a path, itself, the destination and the next hop (which could be the same as the destination). The next hop router is simply the router that advertises the destination with the lowest cost (number of hops).

For each router n that receives the packet we choose neighbours m to forward to based on if they are the second entry on the known shortest path to a destination dst in the destination set D of the packet.

While this simple distance-vector approach leads to a shortest path in the case of a single destination, with multiple destinations the performance is worse. As routers cannot know how they fit on a shortest path tree from the source to each destination, forwarding on the best next hop to all destinations would act like a form of limited flooding. This is caused by each router forwarding the packet on its shortest path links to all destinations. While the algorithm floods the packet in the general direction of the destinations, there is still a large overhead in terms of links used compared to a shortest path tree.

In Fig. 4a we can see an example network consisting of 7 routers with a source router 1, and destination routers 3, 4 and 6. The links used by our simple algorithm are coloured green. We can clearly see the 'limited flooding' effect here, especially in router 7. This router is also forwarding to router 3 as it is the shortest path from the point of view of router 7. It is, however, not on the actual shortest path from the source to that destination and router 3 has already received the packet from another router.

3.3.2. DV Algorithm 2

It is obvious that the algorithm we described previously is not very efficient; it uses more links than necessary to reach all destinations. We can improve the performance of the algorithm by ignoring packets that do not arrive on the reverse path interface to the source. The shortest path $p_{n,src}$ should have the previous hop, ph, as the second entry. This reverse path check already slightly reduces the average link usage due to routers not forwarding packets for which they are not on the reverse path, but the 'limited flooding' problem remains. Routers that are on the reverse path to the source and destination routers will still forward the packet to the other destinations in most cases.

To solve this forwarding problem we add a check for the cost towards the destination. We only forward packets if the current routers cost towards the destination is smaller than the cost reported by the previous hop. This check confirms that the current router n is actually closer to the destination dst than the previous router ph.

We extend our forwarding function fn with these extra checks. This gives us the function $fn(n, D, src, ph)$, where we add the source src and previous hop ph of the packet as extra inputs. The output is the set of forwarding next hops as before.

$$fn(\bullet) = \begin{cases} \forall m \in neighbours(n) : & \\ \exists dst \in D : m = p_{n,dst}(2) \wedge & \text{if } ph = p_{n,src}(2) \\ |p_{n,dst}| < |p_{ph,dst}| & \\ & \\ \emptyset & \text{if } ph \neq p_{n,src}(2) \end{cases}$$

In Fig. 4b we can see the effect. Router 7 sees that the cost for the previous hop (router 5) to reach router 3 is 1. The cost for router 7 to reach router 3 is also 1, thus the packet is not forwarded on that link. Note that in this case we might actually prevent two transmission, as the packet might pass each other on that link as router 3 might send a packet for router 6 through 7. We still see router 4 sending a packet it receives from 3 to 6 as the cost 3 reports is 2 while router 4's own cost to 6 is 1.

3.3.3. DV Algorithm 3

We can further improve the algorithm by checking the cost to reach the source. A router should check if the cost to the source as reported by a candidate next hop router is higher than its own cost to reach the source. Logically, if this was not the case, the candidate next hop should have already received this packet via another path. This prevents the packet from propagating 'backwards' in certain situations. This check improves performance due to the fact that a router receives a packet because it is on the shortest path tree for at least one destination, but evaluates its forwarding for all destinations. The 'directed flooding' effect is reduced, but unneeded transmissions are not completely eliminated.

We once again extend our formula fn by adding the next hop nh as an input and checking the path length from the candidate next hop m to the source.

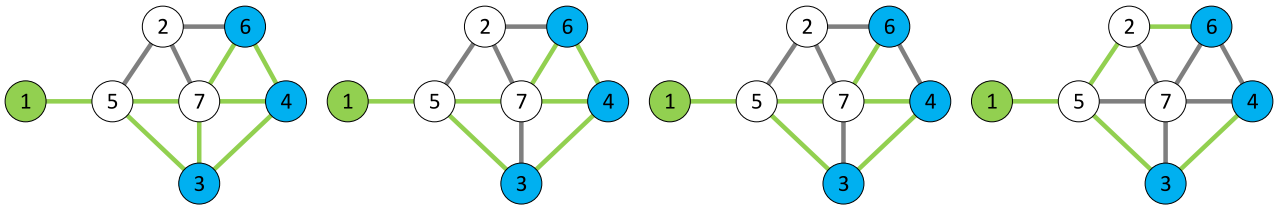
$$fn(\bullet) = \begin{cases} \forall m \in neighbours(n) : & \\ \exists dst \in D : m = p_{n,dst}(2) \wedge & \\ |p_{n,dst}| < |p_{ph,dst}| \wedge & \text{if } ph = p_{n,src}(2) \\ |p_{n,src}| > |p_{m,src}| & \\ & \\ \emptyset & \text{if } ph \neq p_{n,src}(2) \end{cases}$$

We can see the improvement of this addition in Fig. 4c, where the packets router 4 receives from router 3 and 7 are not forwarded to router 6 as the cost for router 4 to reach router 1 is identical to the cost of router 6 to reach router 1.

3.3.4. DV Algorithm 4

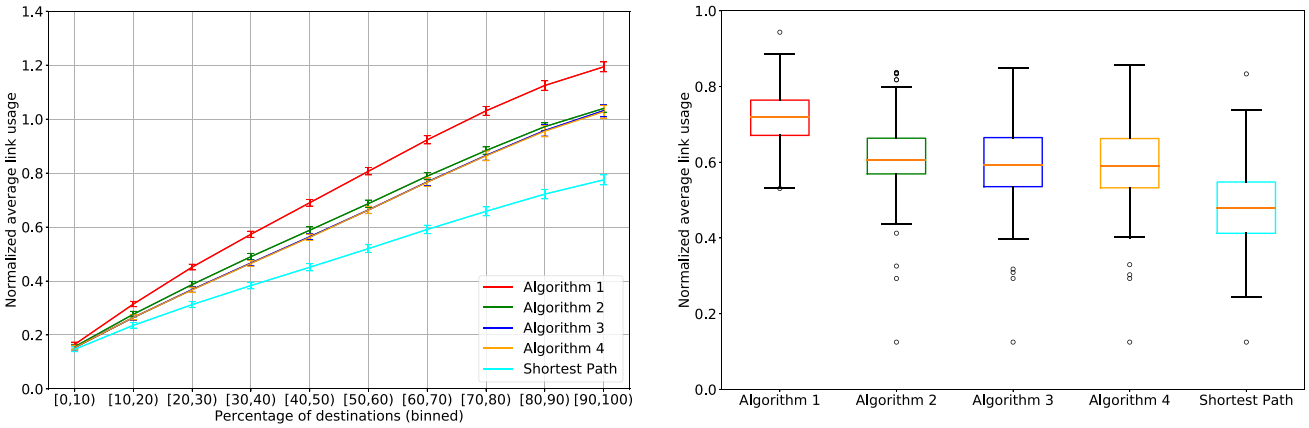
Our final improvement to this algorithm is to prevent random selection of the next hop when there are two equal length paths for a destination router. The path is now selected through a deterministic method. A router will select the next hop router based on its router ID. We chose (arbitrarily) to use the lowest next hop ID for this. This choice forced packets down the same links when there is a choice. This change will force packets with similar destinations over the same link, leading to less overall link usage. Our path-based algorithm will exploit this deterministic behaviour for its forwarding as we will explain in Section 3.5.

The results of this modification are visible in Fig. 4d. Note that had we chosen to use the highest next hop id, the forwarding tree would be



(a) Algorithm 1 (b) Algorithm 2 (c) Algorithm 3 (d) Algorithm 4

Fig. 4. Example forwarding tree of different distance-vector algorithms.



(a) Binned performance (b) Overall performance

Fig. 5. DV algorithm performance.

equal to that shown in Fig. 4c, so this modification does not guarantee a shorter forwarding tree. It does however force packets over somewhat similar paths in cases where shortest paths to multiple destinations share a similar lowest next hop id.

This final distance-vector algorithm can be implemented by looping over all destinations, finding the candidate next hop and evaluating this next hop with the rules given before. This gives us a worst case complexity of $O(|D|log(|V|))$ per forwarding operation, where $|D|$ is the number of destinations and $|V|$ the number of vertices in the network.

3.4. Intermezzo: DV performance

Before we describe our path-based solution we will evaluate the link usage of our DV-based algorithms. We do this to evaluate if our proposed solution satisfies our initial goal of link usage similar to a shortest path tree.

We perform our evaluation over a large set of real world network graphs taken from the Topology Zoo [21]. Our exact evaluation method will be explained in Section 4. For now, we will present the performance of our algorithms as a normalized link usage metric. This is calculated by taking the average link usage for a certain number of destinations in a network and dividing it by the number of links in the network. A value of 1 represents all links being used, a value above 1 shows that one or more links are used multiple times. We then bin the number of destinations per 10% and average these numbers over all networks with more than 10 nodes. We compare the link usage of the algorithms described above with the link usage a shortest path tree (cyan line/box) would have.

The overall results of our evaluation are shown in Fig. 5a. We can see that the improvement between the first and second distance-vector algorithm is relatively large, while further improvements provide only minor benefits. Overall, the link usage of our 4th distance-vector algorithm is around 12% larger than the shortest path tree link usage for a small number of destination to around 32% when (almost) all routers are inside the destination area. On average the link usage is 24% larger than our shortest path tree target. This result implies that for situations where only a small number of routers in the network would be addressed, the simple solution might be viable, but for large destination sets the overhead is relatively large.

Fig. 5b shows box-plots for all algorithms averaged over all numbers of destinations. Using this plot we can compare the overall performance of the different algorithms. We can mainly see that the largest improvement was made with relatively simple additions to our forwarding rules, and that later additions only marginally improve the link cost of the forwarding tree.

After evaluating the performance of the different distance-vector algorithms we conclude that not one comes close to the tree cost of the shortest path tree. In some cases there are even two identical packets traversing the same link when routers forward at the same time, resulting in packets 'crossing' each other on the link. Due to these results we will develop a path-based algorithm that will have link usage closer to the shortest path tree.

3.5. Path-based distance-vector

While the link usage of the 4th distance-vector algorithm presented is identical to the shortest path tree in the case of the network in Fig. 4, this does not hold for larger networks. We can clearly see this by looking at the link usage in Figs. 5a and 5b. The main problem with the distance-vector approach is that routers have no information on their place in the complete forwarding tree in the network. Considering the limited knowledge that is used to calculate forwarding decisions in the DV algorithms we can certainly do better with more information about other paths. As stated before, our aim is to establish a forwarding tree which is as close as possible to the shortest path tree. To prevent the limited flooding effect (unnecessary links are used to reach the destinations) and also keep the amount of information that needs to be distributed in the network relatively low, we have investigated an

approach where routers not only know the next hop to each destination, but also know the complete path to other routers, somewhat like the Border Gateway Protocol (BGP) [22]. This information will allow routers to make decisions that can lead to a forwarding tree that is almost identical to a shortest path tree, at the cost of computational overhead and larger route advertisements.

Our proposed path-based algorithm evaluates forwarding decisions on a per destination router basis. The destination address of a packet arriving at a router is mapped to a set of routers advertising (partial) coverage of this destination. A router can now use its shortest path information for each of these destinations to evaluate its forwarding options, similar to the purely cost based algorithms presented earlier.

We develop our path-based algorithm on the basis of the 4th distance-vector algorithm we presented earlier. The forwarding rules from this previous algorithm are extended to no longer use only the number of hops but the entire path to base the evaluation on.

The main problem we try to fix with this path-based approach is that routers have no knowledge of alternate paths through the network while making their forwarding decisions. This can lead to extra transmissions in some cases where for example destination routers think they need to forward the packet to other destinations, while in reality these have already been reached. We attempt to solve this problem by keeping track of two distinct paths towards each other node in the graph when possible. We will describe our route distribution method later in the paper. For now we will assume each router knows the one or two (when such an alternate path exists) shortest paths towards all other routers.

We will start by explaining our path distribution method. The algorithm calculating the next hop(s) will be described following this, followed by an explanation of the lowest next hop ID rule. We conclude this subsection by describing situations in which the algorithm fails to establish a forwarding tree that resembles a shortest path tree.

3.5.1. Route distribution

Each router or node in the network advertises its coverage area(s) on each of its links. This advertisement contains the path, initially only the advertising router. A router must append its own id to the path it is propagating.

An advertisement packet contains the $Area$, $Cost$ and $Path$ to reach it. The $Path$ is a set of router IDs: $Path = ID_0 \rightarrow ID_1 \rightarrow \ldots \rightarrow ID_n$, where ID_0 is the advertising router and ID_n the previous hop as seen from the receiving router.

The advertising router id should be unique in the network. Using this method, different routers with identical or overlapping coverage areas can be uniquely identified. This allows a router to know which of its links lead to routers that cover the geographic area in the destination field.

A router will transmit the best path to each router it is aware of to all neighbouring routers except if this neighbouring router is contained in the path. In that case the router will transmit an alternative, possibly longer, path that does not contain the other router. If an alternate route is not available, for example because the router has no other links, the path containing the neighbour is returned. A neighbour can detect such a loop due to the path information in the advertisement. This system with two distinct paths ensures that routers have knowledge about the existence or non-existence of alternate routes.

Fig. 6 shows an example of the path for the router with id 1 being distributed through a network with 6 nodes. The paths marked in green are chosen by the receiving routers due to the lowest next hop id rule. The paths marked in red are kept as second best path by the receiving routers.

A router will need to keep track of the advertisements it receives on all its links. Assuming each router has one area it will cover (this could also be zero or multiple areas), there will be an entry per destination per link resulting in $|V| \times deg(n)$ entries for a router where $|V|$ is the number of routers in the network and $deg(n)$ the node degree of the router itself (the number of links this router has). The average node degree for a network is given by $2 \cdot |E|/|V|$, where $|E|$ is the number of links in the network, giving us a average space complexity of $O(|E|)$ for a single router.

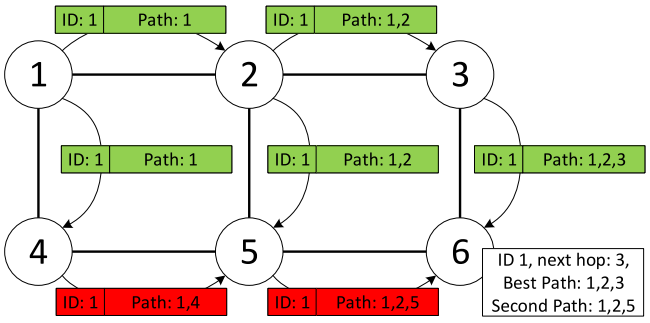


Fig. 6. Example of route distribution.

The resulting routing table has a number of entries that is at least equal to the number of distinct (router, coverage area) pairs in the network. The worst-case scenario is that the number of entries is twice the number of pairs due to the existence of alternate routes. An example would be router 3 in Fig. 6, which will receive an alternate route from router 6 to router 1 with the path $1 \rightarrow 2 \rightarrow 5 \rightarrow 6$, as the best route known to router 6 passes through router 3 itself.

In the event a router detects a link as no longer available, by no longer receiving advertisements on that link, it will stop advertising paths that contain this link to its neighbours. As routers do not propagate paths that contain themselves in the path, eventually all nodes will have updated path information.

3.5.2. Choosing forwarding next hops

Once a router receives a packet, it will evaluate the packet's source and destination. The destination geocast address is translated to all known routers in the network that (partially) cover that area using the method described in our previous paper [16]. After the router generates this list of the covering router ids it can move on to the forwarding step.

A router needs to evaluate each (source, destination) combination using a function $fn(n, D, src, ph)$ which takes the current router n, destination set D, source src and previous hop ph as input values.

$$\begin{aligned} fn(n, D, src, ph) = &\forall m \in neighbours(n) : \\ &\exists dst \in D : dst \neq n \wedge \\ &m = p_{n,dst}(2) \wedge m \neq ph \wedge \\ &fn^*(n, src, dst, ph, m) \end{aligned}$$

A given destination dst from the set of destinations D and the candidate next hop m to that destination are evaluated for forwarding if: (i) the current router n is not the destination, (ii) the next hop for the destination is not the previous hop ph and (iii) we do not already have the next hop in the forwarding hop list because of another destination in the set. This initial step can be seen in Algorithm 1. If a candidate next hop passes these initial steps we compare three possible paths to each other in $fn^*(n, src, dst, ph, m)$. We check the shortest path from source to destination through the current router, the shortest path as seen from the previous hop and the shortest path as seen from the candidate next hop for each candidate next hop that passes the initial checks.

Algorithm 1: Next hop(s) algorithm

```
Input  : Destination routers list D
         Previous hop ph
         Source router src
Output: List result with next hops for the packet
1  List result;                                          // Initialize list
2  if prev_hop == −1 then                                // is entry router
3      foreach dst ∈ D do
4          if self.nextHop(dst) ∉ result then
5              result.add(self.nextHop(dst))
6  else
7      foreach dst ∈ D do
8          nh ← self.nextHop(dst);
9          if dst! = self and nh! = ph and nh ∉ result and
           find_diff(dst, src, ph, nh) then                // Alg.2
10             result.add(nh)
```

Algorithm 2: find_diff (Find path difference)

```
Input  : Candidate next hop next_hop
         The previous hop prev_hop
         Destination dst
         Source scr
Output: Boolean result
1  nh_src = self.pathTo(next_hop, src);
2  nh_dst = self.pathTo(next_hop, dst);
3  ph_src = self.pathTo(prev_hop, src);
4  ph_dst = self.pathTo(prev_hop, dst);
5  ph_dst_removed = ph_dst;
6  foreach n ∈ ph_src do
7      if n ∈ ph_dst_removed then
8          ph_dst_removed.remove(n)
9  ph_src_removed = ph_src;                              // Copy list
10 foreach n ∈ ph_dst do
11     if n ∈ ph_src_removed then
12         ph_src_removed.remove(n)
13 ph_src_dst = ph_src_removed;                          // Copy list
14 ph_src_dst += ph_dst_removed ;                        // Add lists
15 nh_src_removed = nh_src;                              // Copy listp
16 foreach n ∈ nh_dst do
17     if n ∈ nh_src_removed then
18         nh_src_removed.remove(n)
19 nh_dst_removed = nh_dst;                              // Copy list
20 foreach n ∈ nh_src do
21     if n ∈ nh_dst_removed then
22         nh_dst_removed.remove(n)
23 nh_src_dst = nh_src_removed;                          // Copy list
24 nh_src_dst += nh_dst_removed ;                        // Add lists
25 p_src_dst = ph_src;
26 p_src_dst += nh_dst ;                  // Path through this router
27 result = false;
28 if length(ph_src_dst) ≥ length(p_src_dst) then
29     if self ∈ nh_src then
30         result = true;                                // on nh to src
31     else if length(nh_src_dst) > length(p_src_dst) then
32         result = true ;                       // Other path is worse
33     else if length(nh_src_dst) == length(p_src_dst) then
34         temp = [self] + ph_src;
35         for i = 1; i < length(nh_src) + 1); i++ do
36             if nh_src[length(nh_src) − i]! = temp[length(temp) − i]
               then
37                 if nh_src[length(nh_src) − i] > temp[length(temp) − 1]
                   then
38                     result = true;
39                 break;
40 return result;
```

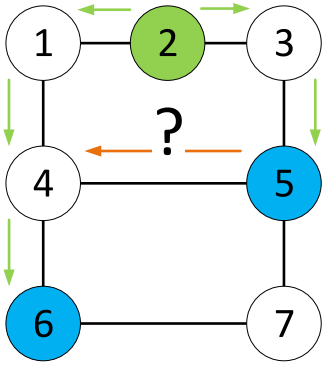


Fig. 7. Forward lookup from node 5 to 4.

We will use Fig. 7 to illustrate our path comparison rules. In this figure, router 2 is the source for a packet that needs to be delivered to routers 5 and 6. We will focus on router 5, which has to decide if it will forward the packet it received towards router 6. There are two shortest paths from 5 to 6, namely $5 \rightarrow 4 \rightarrow 6$ and $5 \rightarrow 7 \rightarrow 6$). Both candidate next hops pass the initial check described in Algorithm 1. We will evaluate the path through router 4 for this example as it has a lower id, the exact reasoning behind this choice will be explained in Section 3.5.3.

We start by constructing the path from the source to the destination as seen from the next hop router $p^{nh}_{src,dst}$. This is the shortest path from the source to the destination that the next hop can know of, given the path information it has.

$$p^{nh}_{src,dst} = p_{src,nh} \triangle p_{nh,dst}$$

Here $p_{src,nh}$ is the shortest path from the *source* to the *next hop* node and $p_{nh,dst}$ the shortest path from the *next hop* to the *destination*. We define the $\triangle$ operation as the concatenation of the two paths excluding duplicate routers except the one that connects the paths. This is done in Algorithm 2 on lines 15 to 24.

In our example $p_{src,nh} = 2 \rightarrow 1 \rightarrow 4$ and $p_{nh,dst} = 4 \rightarrow 6$ which gives us $p^{nh}_{src,dst} = 2 \rightarrow 1 \rightarrow 4 \rightarrow 6$. In some situations both paths could share multiple routers at the start, leading to the exclusion of all but the last of these shared routers in the constructed path. In our example there is only one shared router so it stays on the path as it is also the last.

We now construct a similar path as seen from the previous hop router $p^{ph}_{src,dst}$.

$$p^{ph}_{src,dst} = p_{src,ph} \triangle p_{ph,dst}$$

Where the path is constructed from the shortest path from the *previous hop* to the *source* $p_{src,ph}$ and the shortest path from the *previous hop* to the *destination* $p_{ph,dst}$. This is done in Algorithm 2 on lines 5 to 14.

In our example $p_{src,ph} = 2 \rightarrow 3$ and $p_{ph,dst} = 3 \rightarrow 2 \rightarrow 1 \rightarrow 4 \rightarrow 6$ which gives us $p^{ph}_{src,dst} = 2 \rightarrow 1 \rightarrow 4 \rightarrow 6$. Note that the previous node reports a longer path towards node 6 to node 4 as the shorter path passes through node 4 itself.

Finally we construct the path that the packet will take if we forward it, $p^{n}_{src,dst}$, which is the path as seen from the current router n.

$$p^{n}_{src,dst} = p_{src,ph} \triangle n \triangle p_{nh,dst}$$

The path is constructed from the path from the *previous hop* to the *source*, the *next hop* to the *destination* and the router n itself.

Using the example in Fig. 7, this results in $p_{ph,src} = 3 \rightarrow 2$ and $p_{nh,dst} = 4 \rightarrow 6$ which gives us $p^{n}_{src,dst} = 2 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow 6$. Note that we include the current node in this path! Using all relevant paths, we can compare their lengths in $fn^*(n, src, dst, ph, m)$.

$$fn^*(n, src, dst, ph, m) = |p^{n}_{src,dst}| \leq |p^{ph}_{src,dst}| \wedge |p^{n}_{src,dst}| \leq |p^{m}_{src,dst}|$$

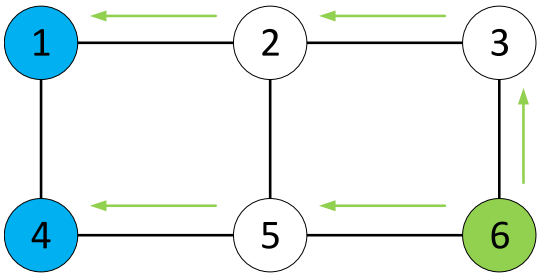


Fig. 8. Forwarding from node 6 to nodes 1 and 4.

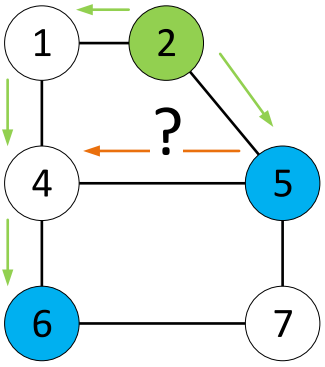


Fig. 9. Forwarding lookup from node 5 to 4.

Where fn^* takes the current router n, source src, destination dst, previous hop ph and the candidate next hop m. fn^* returns true or false.

In our example from Fig. 7, fn^* would check if $|2 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow 6| \leq |2 \rightarrow 1 \rightarrow 4 \rightarrow 6|$. This gives us $(5 \leq 4)$, which is false. The result is that router 5 will not forward the packet to router 4. If the statement would have been true the second evaluation would have also been False (in this case it is even the same path).

Pseudo code for the entire forwarding operation is given in Algorithms 1 and 2. In Algorithm 1 we show the initial checks. The router will always forward on the shortest path to the destinations if it is the entry router for the packet in the network (if $ph == -1$). If the packet is received from another router in the same network the router finds a candidate next hop ($nextHop$), checks if it is not the destination, the packet is not returned on the previous hop and that the next hop is not already included in the forwarding next hops list. If all these checks pass the router will call Algorithm 2 which performs the path comparison checks.

The number of times these path differences have to be calculated per packet is limited by the node degree of the router. Our path difference algorithm combines 4 paths into 2 by looping over their entries. When these 2 resulting paths are equal length another loop over a path is needed leading to a worst case of 5 loops over a path. The worst case for this path length is the diameter of the network giving a worst case time complexity of $O((2|E|/|V|) * |V|) = O(|E|)$ per forwarding operation.

3.5.3. Lowest next hop ID

When a router has the option of two or more paths of identical length to forward on for a certain destination it has a choice. We let routers base this choice on the lowest router id of the next hop. This makes the choice between paths deterministic and by extension allows routers further in the forwarding tree to know for which destination they are part of the forwarding tree.

Consider the network from Fig. 8, here a tree is constructed from node 6 to nodes 1 and 4 based on the rules described previously. We choose router 3 as the forwarding hop to router 1 over router 5 because it has a lower id. In Algorithm 2 (which will be explained fully later on) we can see the lowest id rule implemented in lines 43 to 52.

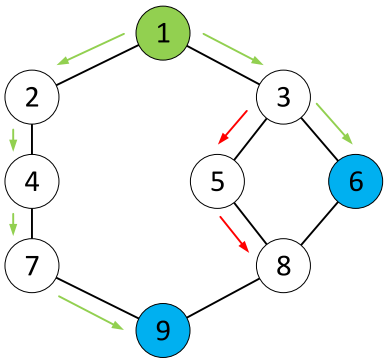


Fig. 10. Shortest example of forwarding error loop. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

A similar choice also needs to be made if a router needs to choose between forwarding or not based on path knowledge of its candidate next hop. The router can compare the two paths and check which of the paths has the lowest ID next hop from where they diverge from each other. Using this method a router can determine where it sits in the forwarding tree and for which destinations it should forward.

To illustrate this, we will use a modified version of the network shown in Fig. 7 with node 2 removed, shown in Fig. 9. The source is router 2 and the destination are routers 5 and 6 as before. Router 5 has to decide if it forwards the packet is has received to router 6. There are two possible paths available to the router $5 \rightarrow 4 \rightarrow 6$ and $5 \rightarrow 7 \rightarrow 6$, where the first path has the lower next hop ID. This path through router 5, $2 \rightarrow 5 \rightarrow 4 \rightarrow 6$, is now compared to the path as seen from the candidate next hop router 4: $2 \rightarrow 1 \rightarrow 4 \rightarrow 6$. These paths diverge from each other after router 2, where the best path as seen from router 4 has the lowest next hop ID. Router 5 now knows it does not have to forward the packet.

This method of choosing one path over the other allows our system to only use one path to each destination. In some cases such as the one shown in Fig. 9 this leads to a forwarding tree that has a slightly higher cost than a Steiner tree, but multiple paths are not used to reach the same destinations.

3.5.4. Deviation from the shortest-path-tree

The algorithm using limited path knowledge described above constructs a forwarding tree that is close to the shortest path tree in terms of link usage. However, the algorithm fails to construct such a tree in some specific situations where the network contains loops within loops. A minimal example of one such network can be seen in Fig. 10. In this figure node 1 represents the source, nodes 6 and 9 are the destinations.

Routers in the small loop will receive advertisements for the source and destination from both their neighbours as they keep track of the two best paths (when multiple paths exists) to the source. Both paths will use the small loop to reach those destinations as the distance is shorter compared to the large loop. The result is that the routers inside the small loop have no knowledge of the alternate path through the larger loop and mistakenly believe they should forward the message for node 9. The router that connects the small loop to the larger loop (router 8 in Fig. 10) does have this knowledge and will correctly not forward the packet.

The maximum cost of the extra link usage for this topology is half the number of links on the small loop. This occurs in the case where the destination in the small loop is on the side of the higher next hop id from the source (like node 6 in Fig. 10). In practice, as our evaluations will show in Section 4, this problem almost never occurs and when it does the extra overhead caused is minimal.

3.6. Link state

For completeness we have to mention the option of using a link state algorithm for geographic routing. Using a link state algorithm, which can provide full network knowledge, each node can determine if it is on the shortest path between a given source and destination and base its forwarding decisions on this information.

The benefit of this approach would be the perfect shortest path tree for any given (source, destinations) combination. The main downside would be computational, storage and communication overhead of the full network graph as compared to the other alternatives presented. Each node would have to compute the shortest paths from the source to all destinations for every incoming geocast packet. The worst case complexity of this operation would be $O(|V|^2 log(|V|))$. It is possible to pre-compute the shortest path between each node in the network (given there are no network changes in the meantime). In that case the router would just need to lookup the next hop for each (source, destination) pair, but this would require storing this information.

Link state algorithms would also allow each router to compute a Steiner tree for a given source and destination set, in theory allowing the network to forward using the lowest cost tree possible.

However, the amount of data that would need to be transmitted and the computational overhead for such an approach would be large and scale with the size of the network. Considering these drawbacks we think a link state approach is not feasible and will not consider it further in the remainder of this paper. We will also show in Section 4 that the path-based distance-vector approach already comes close to a shortest path tree in terms of link usage.

3.7. Hierarchical routing

Due to the ability of our addressing scheme to aggregate the geographic addresses, as described in Section 2.2, it is possible to advertise an entire network as a single coverage area. This enables geographic routing on a large scale, as each network would not be represented by a single or even multiple coverage areas per router, but by a single unified area.

As an example we take a network covering an entire city. This network could aggregate the coverage area of the routers in a single address. While this single address might not be completely identical to the coverage area of the individual routers (it will be slightly larger in reality), it allows the network to advertise its area in a single advertisement. The same holds for any autonomous system, we believe our system could work for inter-network geocast using this method.

4. Algorithm evaluation

In this section, we will describe the evaluation of our proposed path-based routing algorithm. We will start with briefly mentioning the tools we used to perform the evaluation followed by the method for destination selection. We then describe how we measure the path-based algorithm's link usage and describe the method we have used to evaluate our algorithms. We will evaluate how close the forwarding tree constructed by the algorithm is to the shortest path and Steiner tree. Our main metric will be the number of links used to construct the tree.

4.1. Tools

All our evaluations are run over a set of real-world networks taken from the Topologyzoo [21] unless otherwise noted. Using these networks, we hope to more accurately evaluate performance in real-world scenarios as compared to randomly generated ones.

To analyse our algorithms on these networks we use the network library NetworkX [23] for the python programming language. We use this tool to import the Topologyzoo network graphs. We remove all nodes that are not connected to other nodes from these graphs. If a graph is disconnected we use the largest connected part. On the network graphs we use the route distribution system described in Section 3.5.1 for as many message exchanges as is needed for the route table in each node to become stable.

4.2. Destination distributions

For our evaluation we define two categories of destination sets: Geographically scoped and randomly distributed destinations. We believe these two sets cover most realistic use cases.

4.2.1. Geographically scoped destinations

In most networks that we have evaluated we observe that the geographical distance between two routers and the network distance (number of hop between them) is closely linked. This observation has led us to believe that within a network most geocast traffic will be geographically scoped in its destination router set.

For our geographically scoped destination set we use each node in the network as a source for every possible group of geographically scoped destinations. The destinations are selected based on their location, for n destinations we select a node in the network and add the $n-1$ geographically closest nodes to the set. Each node is selected once, duplicate sets are filtered out as they would represent the same destination area. The source is never included in the destination set. The result is that each possible geographically scoped (source, destination) combination in the network is evaluated exactly once.

4.2.2. Randomly distributed destination

Because in practice it seems unlikely that all geocast destination will be geographically clustered in a network, we also evaluate randomly selected destinations. We believe such a situation can occur when a network (A) serves multiple other networks (for example B, C, E). Let us assume networks B, C and E all cover our destination area. It is unlikely that the connections of these networks to A are geographically clustered, thus the randomly distributed destination scenario is also important.

As with the geographically scoped destinations we select each node once as the source. For the destinations we select every possible combination of destinations that do not include the source exactly once.

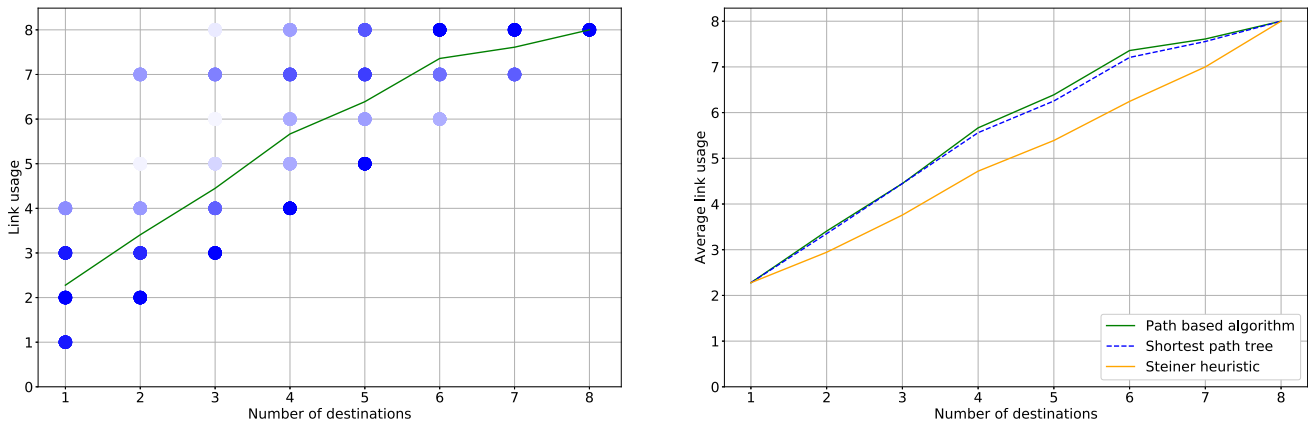
4.3. Evaluation metrics

To evaluate the routing performance we look at the resulting link usage in our evaluation graphs. This metric will give us an indication of how efficient the routing algorithm performs its goal of establishing a shortest path forwarding tree. We compare the average number of links used per number of destinations.

We define link usage as the number of links that are used to forward a message from the source to all destinations. If a link was used twice (i.e. in both directions) this counts as two link uses. For example the forwarding tree shown in Fig. 4a uses 7 links while the more efficient algorithm in Fig. 4d only uses 5 links.

To better illustrate the method we use to present our results we will demonstrate this method for a single network. In Fig. 11a, we show the routing cost in links used over all geographically scoped destination locations in the example network from Fig. 10. The green line represents the average link usage, with the colour intensity for the blue dots showing the relative occurrence of the link usage for a certain number of destinations. The effect of the loop inside a loop (explained before in Section 3.5.4) can be clearly seen here with the cost of 3 and 4 destinations mainly around 4 and 7, but not 5 and 6.

Fig. 11b plots the performance in link cost of the path-based routing algorithm (green line) against the performance of a shortest path tree (dashed blue line) and a Steiner tree heuristics algorithm (orange line) for the same network used in Fig. 11a. The green line corresponds to



(a) Full results (b) Averaged costs

Fig. 11. Normalized link cost for the network from Fig. 10. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

the solid green line in Fig. 11a. We can see that the performance of the routing algorithm in terms of links used is close to that of the shortest path tree.

As different networks have different numbers of routers and links, the results for them are not directly comparable. We normalize the link usage to allow us to make this comparison. The normalization of link usage is done by dividing the link usage with the number of links in the network, resulting in a number between 0 and 1. Values above 1 are possible if there are multiple transmission on the same link.

4.4. Evaluation method

We evaluate our routing algorithm by running it on a collection of real world networks taken from the Topology Zoo project [21]. We initialize every network by performing the route distribution step until the routing table of each router is stable. This is done by letting the routers exchange information in steps, in each step all routers transmit their path information to all their neighbours.

We evaluate each possible (source, destination(s)) combination, generated in the way described earlier in this section, in the network by inserting a packet with the given destination(s) at the source router and forwarding it until no router has any operations left to perform. Forwarding is performed by the path-based algorithm described in Section 3.5.2 and the distance-vector algorithm described in Section 3.3. Each packet is forwarded on all the link(s) this algorithm returns based on the source, destinations and information from the previous and candidate next hop routers. Similar to the route distribution, the forwarding is also performed in steps. Each forwarding step allows all routers to forward a packet if they have any.

This simulation is run for a subset of destinations that are geographically scoped and for randomly distributed destinations as described before. We then compare the average number of links used for a given number of destinations to the number of links used by a shortest path tree and a Steiner tree for the same (source, destination) combinations. The shortest path tree is calculated by finding the shortest paths from the source to all destinations and taking the union of these paths. The Steiner tree is calculated using a well known heuristic [17].

4.5. Evaluation results

Now that we have described how we evaluate our algorithms, we will show the average normalized link usage of our algorithms compared to a shortest path and Steiner tree in this section.

In Fig. 12, we show the normalized link usage on the y-axis. The x-axis represents the normalized number of destinations. This normalization is done by binning the number of destinations for every 10%. We show DV-algorithm 4, the path-based algorithm, shortest path tree and Steiner heuristic normalized over a subset of real world networks. This figure contains results for a subset of graphs containing the 86 graphs over which we also have a complete set of shortest path and Steiner heuristic results for randomly distributed locations. This set is limited due to the time needed to evaluate all random combinations in larger networks and contains Topologyzoo networks with 20 nodes or less.

Fig. 12a shows the geographically scoped results over the 86 graphs while Fig. 12b shows the same values but for randomly distributed destinations. We believe such a scenario could occur in transit networks were the points networks connect to each other do not necessarily correlate with their geographic coverage, especially if these networks cover the same area.

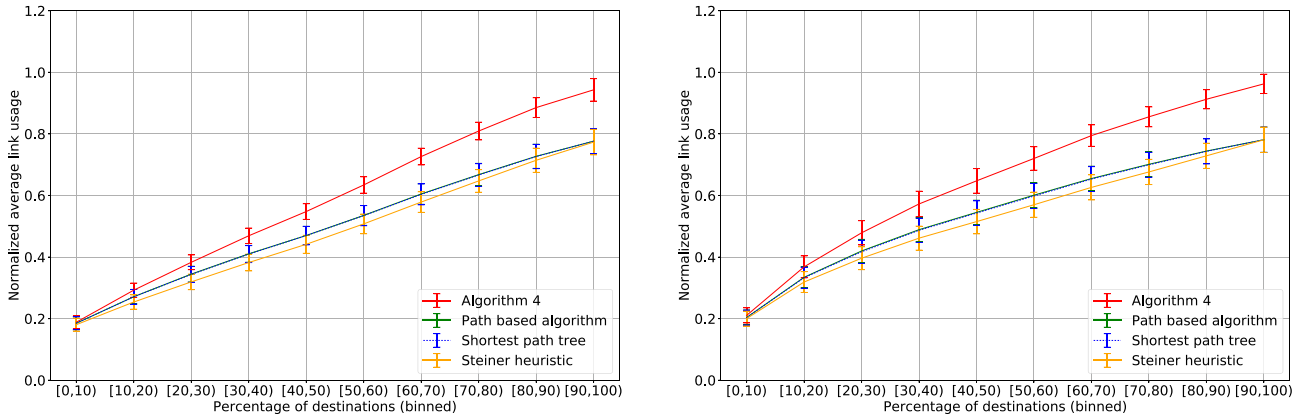
In both cases the line for our path-based algorithm and the shortest path tree almost completely overlap. As expected based on our previous work, the average extra link usage compared to the Steiner tree is relatively small. We also note that our best purely DV based algorithm performs reasonably well when the number of destination in a network is small.

Fig. 13 contains geographically scoped results over the complete set of 227 real-world network graphs we used for the evaluation. Fig. 13a shows results over this set using the same method used for Fig. 12a. We can see that over this larger set of graphs, which also contains larger networks, the average cost for our path-based algorithms is similar, but the link usage of DV-algorithm 4 is slightly higher implying that its performance might degrade depending on the network size.

In Fig. 13b we show the normalized average link cost of an entire graph (the average number of links used over all geographic destination combinations divided by the total number of links in a graph) plotted against the average node degree of the network. The average node degree is the average number of links a router has, this is an indication of how well connected a network is. We can see that the average link cost of the path-based routing algorithm is close or equal to that of the shortest path tree. In general, the cost for geographically scoped destinations is close to that of the ideal Steiner tree. We also observe that the more well connected a network is, the lower the average cost to reach a certain destination area.

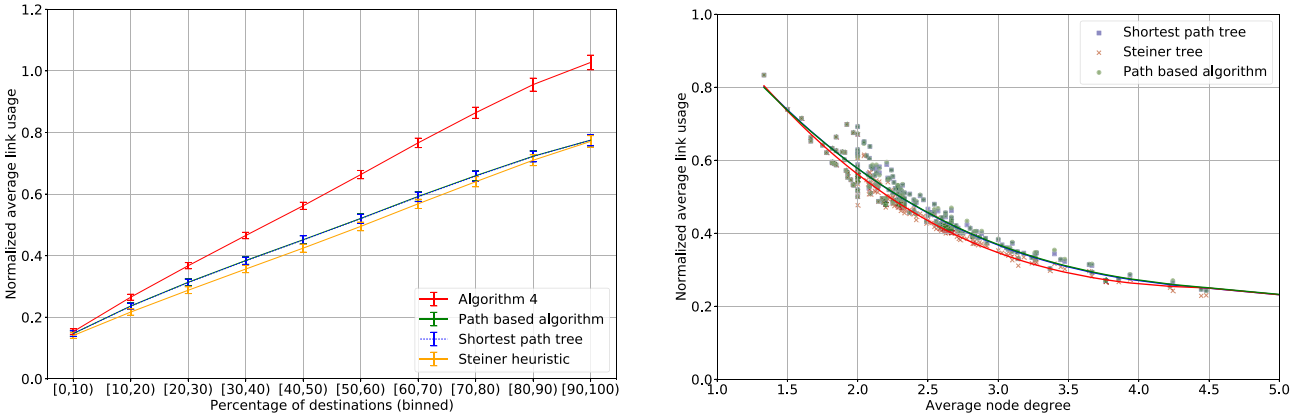
4.6. Path knowledge vs. hop knowledge

Our distance-vector-based algorithm described in Section 3.3.4 has a higher link usage compared to the path-based algorithm described later. It does however have some benefits over the better performing algorithm:



(a) Geographically scoped (b) Randomly distributed

Fig. 12. Normalized link cost for real world graphs comparing geocast and multicast.



(a) Link usage set against percentage of destinations addressed (b) Link usage set against the average node degree of the network

Fig. 13. Geographically scoped normalized link usage for all real world graphs.

- Lower communications overhead due to DV like cost exchange
- Lower forwarding complexity

The communication overhead depends on the size of the network. The larger the network is, the longer paths are that our path-based algorithm has to communicate through the network. The forwarding complexity also depends on the path length, but even with short paths the extra steps required to combine them in a next hop and previous hop view would result in a higher complexity than the distance-vector-based approach

In the 227 real networks, on average the distance-vector-based algorithm has 28% worse link usage compared to the more complex algorithm with a standard deviation of 0.26. The best case was identical link usage with the worst case 112% extra links used. We can conclude that in some networks the extra transmission overhead could be an acceptable trade-off for the lower computational burden put on the routers themselves. There is no single perfect choice here, the algorithm will have to be selected based on the network. We do observe that the overhead of the distance-vector algorithm is lower in smaller networks, and is almost always high in larger networks of more than 15 nodes.

5. Implementation

In order to better validate the proposed algorithms we have implemented both the distance-vector and path-based geographic routing algorithms. Both are backed by a protocol for advertising paths, as presented in Section 3. Routers will periodically send path information to their neighbours. These packets contain the best paths a router has to all other routers it is aware of. We also use this path distribution method for our DV implementation to eliminate timing differences caused by the speed of the underlying information distribution system. We simply only use the associated cost information for the distance-vector implementation. We will start by presenting the general structure of the software, followed by our information distribution, information tracking and forwarding approach.

5.1. Software structure

Our software consists of 8 main components, which are shared by both implementations: The Packet receiver, Advertisement parser, Packet forwarding, Link path table, General path table, Coverage table, Route advertisement generator, and Packet sender. These components and their mutual relationships can be seen in Fig. 14. The general structure of the software is mostly identical for both algorithms, only the “Packet Forwarding” section has algorithm specific behaviour.

The packet receiver handles all incoming packets. Packets that have a geocast destination address are passed to the packet forwarding system while advertisement packets are passed to the advertisement parser.

The route advertisement parser parses the advertisement packet into path data and coverage data. Path data consists of a path for each router included in the advertisement. This data is given to the per link path table and the general path table. Coverage information consists of a geographic address of the coverage area and the id of the covering router. This information is passed to the coverage table.

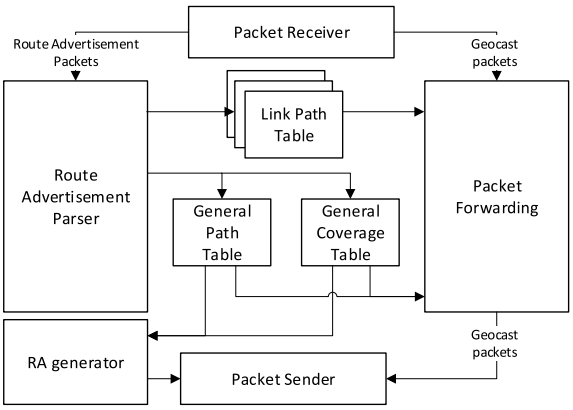


Fig. 14. Global structure of the routing software.

The per link path table is a table of all path advertisements received on a link. It is essentially the best path table of the router on the other side of the link, with the possible exception of paths that include the current router (unless no other path was available).

The general path table contains the best and second best known path from all other routers in the network. It also includes the best next hop for each destination. Due to our lowest id next hop choice this is not necessarily the same as the best known incoming path.

The coverage table is simply a mapping of coverage areas to the router id of the routers covering those areas. The Packet forwarding system receives geocast packets and actually runs the forwarding algorithm based on the information it receives from the coverage, path and per link tables. It gives the packet including a list of links to forward it on to the Packet sender.

The RA generator uses the information from the path and coverage table to generate a route advertisement for each link the router has. When for a link, the neighbouring router is included in the path the second best known path is sent when available.

Finally the Packet sender, as its name implies, sends out all packets on the correct interfaces.

5.2. Route distribution

Routers periodically send route advertisements consisting of all shortest paths and coverage areas known to the router. They take the following form: Advertising router ID (1 byte), Number of advertisements in packet (1 byte) followed by the actual advertisements. A single route advertisement consists of: the coverage area (16 bytes: an IPv6 address), the path length (1 byte), the covering router id (1 byte) and the advertised path (of path length bytes, with a minimum of 1).

The best route table is used to generate these route advertisement packets. As described in Section 3.5.1, a neighbour specific modification is made if a path contains the neighbour it is transmitted to. If this happens the second best route is used if it exists, letting the neighbour know that there is another path. If there is no alternate path, the available path is used. This lets the receiving router know there is no other known path, except the return path to a certain other router.

Assuming perfectly synchronized distribution messages between routers (every router immediately tells its neighbours about new information), it follows that the theoretical time needed for every node to have at least one path to each other node is given by $t_c = t_{ra} \cdot d$. Where t_c is the time needed to converge (in seconds), t_{ra} the time between route advertisements and d the diameter of the network (the maximum distance, or hops, between any two nodes). This is the theoretical lower bound for convergence time, in practice the time needed for convergence will always be higher.

5.3. Distance-vector-based forwarding

To provide a comparison to the path-based algorithm we have also implemented our distance-vector-based forwarding method. To accurately compare the distance-vector to the path-based algorithm we have kept most of the systems identical. We use the same distribution and information tracking systems and only modify the forwarding decision making system. We simply use the length of the best path in the router's global table as the cost metric. This approach allows us to use the distance-vector algorithms without changing the underlying distribution system. The benefit is that the results from running these algorithms become comparable, there are no timing differences as a result of different distribution systems.

This implementation follows the rules we describe in Section 3.3 for the 4th DV algorithm, which includes the lowest next hop id rule. Overall this approach results in a relatively simple forwarding algorithm at the cost of a non-perfect forwarding tree, which in some situations can lead to links being used that are not needed and can even cause packets to be delivered twice at the same router.

5.4. Path-based forwarding

The path-based forwarding procedure is based on the known paths to the source and destination(s) as described in Section 3.5. To briefly recap: For each destination d in the set of destination routers D we evaluate the forwarding path through the current router. We combine our best paths to the source and d into a candidate forwarding path. We compare this path to a similar combination of paths reported by the candidate next hop and the previous hop (this is the main reason we keep a path table for each interface). If our candidate path is better than the other two options the packet is forwarded on this path. This approach ensures that our path is indeed the shortest path from source to destination based on our limited knowledge. Using this method we can construct forwarding trees that are close to the shortest path tree in terms of link usage, as we have shown in Section 4.

6. Implementation evaluation

To validate our implementation we have tested it on a large selection of networks during different convergence scenarios. We will first explain the general evaluation approach which will explain the metrics and networks used, followed by the evaluation results.

6.1. Evaluation approach

In this section we will describe the metrics we are interested in, the different scenarios that were used for the evaluation, and present an overview of the tools used and our method of variable selection.

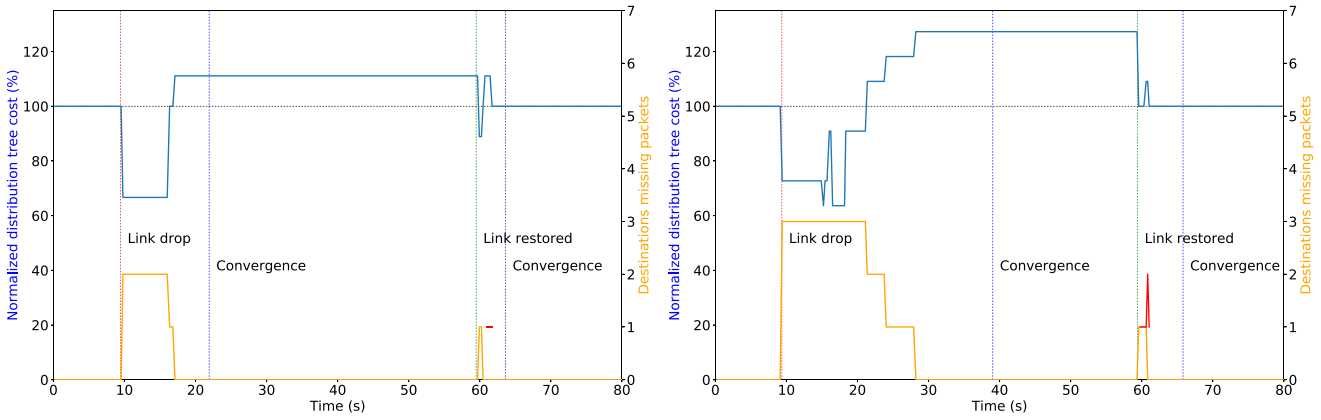
6.1.1. Evaluation metrics

Our evaluation is focused on the convergence times and the behaviour of both protocols during convergence following a link drop and subsequent link restoration. We evaluate our system on the following timing values related to convergence:

- initial convergence time,
- convergence time following a link drop,
- convergence time following a restored link, and
- packet delivery restoration time following a link drop.

This last metric will differ between the distance-vector and path-based algorithms, the first three should be the same as we use the same underlying route distribution system.

We further evaluate the number of packets that are lost during protocol convergence. For this last metric we take into account the number of destinations that did not receive a packet. For example, if 2 out of 3 destinations did not receive a certain packet then these two packets are



(a) Example run 1 (b) Example run 2

Fig. 15. Examples of convergence behaviour on different networks. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

considered lost. Using this method we hope to accurately represent the number of deliveries that were missed during the convergence time.

We will also look at the possible multi-path issues during a convergence phase, in which packets are delivered to a router over multiple paths. Ideally this should never occur, but it is likely unavoidable during protocol convergence due to incomplete or outdated network knowledge in different routers. In the case of the distance-vector based protocol, we expect larger numbers of duplicate packets as the information to build a perfect shortest path tree is just not available.

As a base-line for how efficient the protocols construct their forwarding path we will look at the link usage. We use a normalized link usage metric to more easily compare efficiency between different networks. This normalization is done by describing the link usage as a fraction of the total number of links in a network.

6.1.2. Evaluation scenario

We start each evaluation run by giving the routing algorithm time to converge. We have set this waiting time to 25 s as we have not observed the convergence taking longer than this time. After this initial time, we start transmitting packets from a randomly selected source to a randomly selected destination area. 10 s after the transmissions start we drop a randomly chosen link on the forwarding tree for those packets. We measure how long it takes the algorithm to converge again and the effect this has on the packets that are in transit during this time. 50 s after the link drop, we restore the link and we measure the convergence duration again. We also measure the number of packets lost during the entire run.

6.1.3. Networks and variable selection

Like the evaluation in Section 4, we use real world network graphs taken from the Topologyzoo project [21] to evaluate our protocol on a set of realistic networks. We used a total of 162 networks, those ranging in size from 6 to 51 routers, with an average of 26 routers. In the case a network is not connected, we use the largest connected part. We import these networks into a Mininet virtual network [24] by generating a new node running our routing implementations for each vertex in the graph and establishing a link (1 Gb/s Ethernet) between nodes if the corresponding graph has an edge.

The latitude and longitude location (as given in the Topologyzoo) of the node is used to generate a coverage area of a size corresponding to depth 10 of our addressing scheme (Section 2.2) to ensure routers cover reasonable portions of a network. For each run within a network a destination area is randomly generated. This area can cover between 10% and 95% of the bounding box containing all nodes in the network. We also randomly select a source router for each run, this router can be inside or outside the destination area.

To emulate a link drop we set both ends of a link to have a 100% packet loss rate using the Linux network emulator [25] functionality. The link to drop is chosen randomly from the set of links on the shortest path tree from the source to the destinations. With this link selection we try to guarantee that the dropped link will at least have some effect on the forwarding situation. The dropped link is chosen in such a way that is does not lead to a disconnected network, so the algorithm can actually converge to the new situation and still reach all destinations.

Routers are configured to exchange route advertisements every 0.5 s. Route entries expire after two seconds, leading to a relatively fast update time.

The distance-vector and path-based algorithm are run on the same set of networks with the same sources and destinations. The link that is dropped from the network is also kept identical between the two different algorithms. Due to the way a dropped link is chosen and the way our algorithms construct their forwarding trees, it is not guaranteed this link is on the forwarding tree all the time. For both algorithms the dropped link was on the forwarding tree 92% of the runs.

6.2. Results

In this section we will present our results. We start with an overview of our algorithm's performance in a converged state and continue with the convergence time of the protocol in different network states. We end the section with an analysis of packet loss and duplicate packet delivery during different convergence situations.

6.2.1. Example runs

To better explain our experimental results, we show the convergence behaviour of runs in two different networks in Fig. 15. On the left y-axis (blue line) we show the normalized distribution tree cost where 100% corresponds to the link usage of the initial distribution tree. On the right y-axis (yellow line) we show packet loss as the number of destination routers that did not receive a packet at a certain time. The x-axis shows the elapsed time from the start of the packet transmissions in seconds. The time of the link drop is marked as a dotted red vertical line and the link restore time with a dotted green vertical line. The dotted blue vertical lines mark the time at which the protocol has converged to the new situation.

We can see that immediately following the loss of a link there is a period of packet loss, but packet delivery is restored before the network has fully converged. This can be explained by the fact that the link loss (almost always) occurs on a path that is used, resulting in local convergence before the protocol in the entire network has had time to converge. A similar pattern can be seen following the link restore, but the packet loss is minimal here. This is likely caused by the fact that

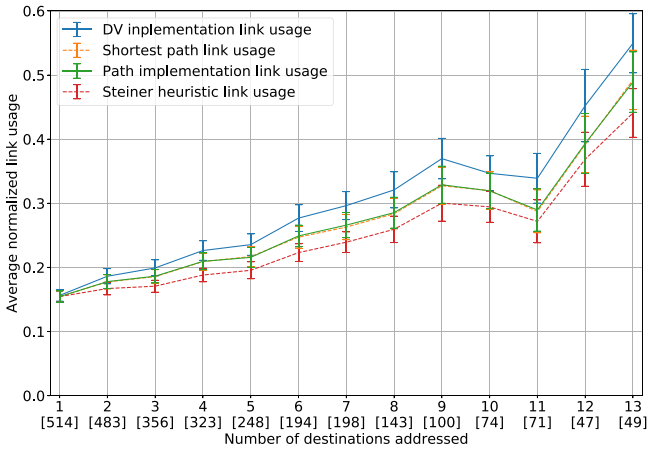


Fig. 16. Normalized link usage in a converged network.

information about new link propagates faster compared to information of lost links due to the way that time-outs function. The small red lines that can be seen following the link restoration represent packets that arrived multiple times at a router in the network. Packets are sometimes routed over multiple paths during convergence, temporarily increasing link usage as can be seen in the graphs. The normalized distribution tree costs decreases after the link loss as the forwarding tree does not reach the destination nodes any more. Eventually the protocol converges and the normalized cost exceeds 100%, due to the larger tree that is needed to route around the missing link. Following the restoration of the dropped link we can see the normalized cost return to 100% as the protocol returns to using the initial distribution tree.

6.2.2. General link usage

In Fig. 16 we plot the link usage of both our implementations against the shortest path tree and a Steiner tree for the same (source, destination) tuples. The shortest path and Steiner tree link usage are a subset of the data used for Fig. 12 limited to the (source, destination) tuples used here. On the y-axis we show the average of the normalized value of the link usage, meaning the number of links used to reach all destinations divided by the number of links in the network. The x-axis shows the number of destinations addressed and below that, the number of runs with this number of destinations in brackets. The difference in number of runs is a consequence of our destination area generation system, which chooses areas with a small number of routers in geographically large networks. The error bars show the 95% confidence interval for that value. Runs with more than 10 destination were not numerous enough to provide statistically significant results.

In general our path-based protocol establishes forwarding trees that use a comparable number of links compared to a shortest path tree. In our 2916 runs there were only 392 runs (13%) where the path-based algorithm established a path with a different number of links than the shortest path tree. In most of these cases (238) this path was one link longer, and in some rare cases (154) one shorter. The former can in some cases be explained by minor inefficiencies, and the latter difference can be explained by situations where there are multiple shortest paths from the source to a single destination. If the chosen path overlaps with that of another destination the resulting link usage can be lower compared to the situation where the other path was taken.

The distance-vector algorithm has a higher link usage compared to the path-based one. The main cause of this is that multiple links are sometimes used to reach the same destination. The higher transmission cost is offset by a simpler algorithm. For this algorithm, 1130 runs (39%) had a different link usage compared to the shortest path tree. A total of 1037 runs had a higher link cost while 93 runs had a lower cost. As with the path-based algorithm, the lower link costs were caused by situations were multiple shortest paths were available, and some overlapped. This result corresponds with the initial results from Section 4.

We can conclude that our path-based protocol performs mostly as expected with respect to the number of links used to construct a forwarding tree. Our distance-vector based protocol has significantly higher link usage.

6.2.3. Convergence time

One of the more important aspects of any routing protocol is the time it takes to converge following a change in the network. The correlation between the network size (the number of routers in the network) and the time convergence takes during the evaluation is shown in Fig. 17 for path-based and DV-based routing. In these figures we show per network averages as points, the lines represent the average over all runs. We can see that the initial convergence time (blue line) of the network shows a strong correlation to the network size. As a larger network mostly corresponds to more possible paths it correlates with the time it takes for the algorithm to converge.

Both the path-based and distance-vector graphs show comparable results for the convergence times. This is the result of both using the same underlying route distribution method. Small differences are caused by differences in the exact time nodes started in relation to each other combined with the time-out of 2 s.

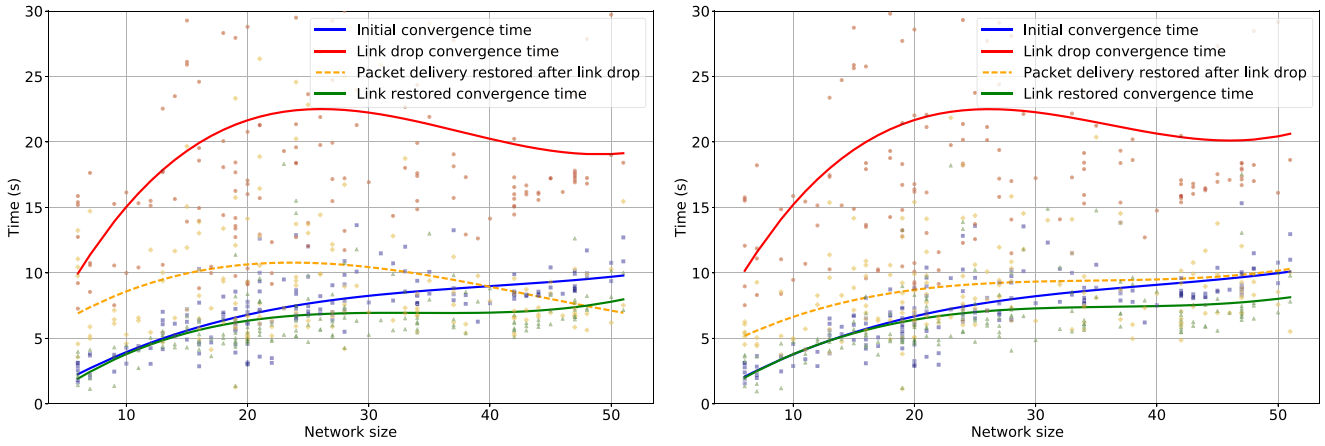
On average the convergence time after a link drop (red line) is significantly larger than that of adding or restoring a link (green line) to the network. As noted before this can be explained by the time-out of 2 s that needs to occur before link loss is propagated while new links are advertised with at most a 0.5 s delay.

One of the interesting things to note in this graph, is that the time to convergence after a link restoration (green line) is lower than the initial convergence time (blue line) for larger networks. We expect this is caused by the locality of the change. In larger networks a single link change is not likely to affect path entries further away in the network, although this does strongly depend on how well connected the network is.

For the path-based protocol, the time it takes for full packet delivery to be restored after a link drop (orange striped line) follows a similar pattern as the link drop convergence time, but takes less time overall (Fig. 17a). The distance-vector protocol shows behaviour more like the link restore convergence time (Fig. 17b). The main difference between the packet delivery restoration time in both protocols is that the path-based method seems to perform worse in networks with 15 to 35 nodes, but better in larger networks. This effect is likely caused by the location of the dropped link in relation to the destination nodes. In relatively small networks it is more likely that a destination is close to a dropped link leading to a lower chance that the path-based method has an obvious route around the problem. The distance-vector approach has no knowledge of these paths and simply attempts (an inefficient) alternative path once it knows about the missing link.

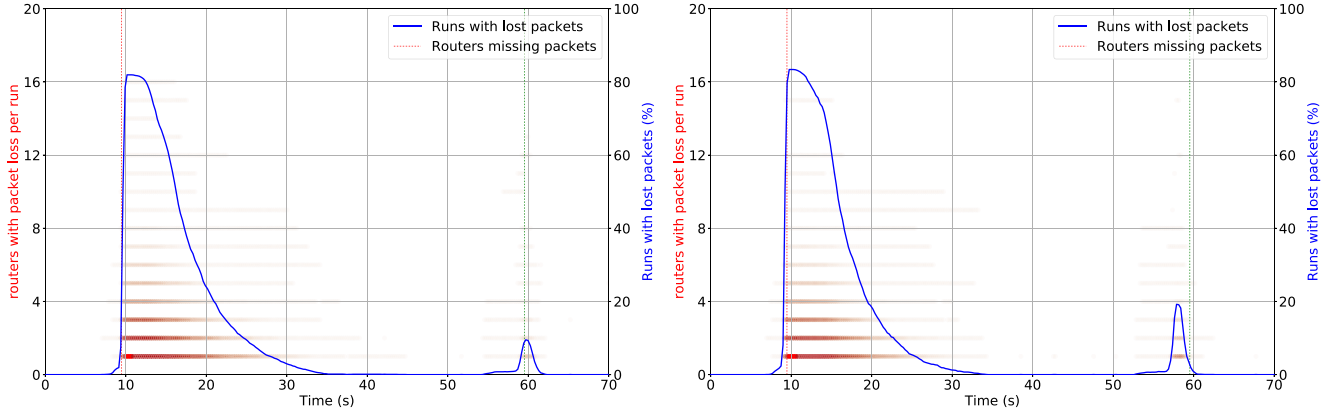
6.2.4. Behaviour during convergence

As can be expected, during the convergence directly following a link drop, a number of packets is not delivered to the destination routers. We plot these losses for the path-based algorithm in Fig. 18a and for the distance-vector based results in Fig. 18b. In these figures, the red dots represent the number of routers that did not receive a packet in a certain simulation run at that time. The intensity of the red colour corresponds to the number of runs in which this number of packets was lost at that particular time. Barely visible dots represent a single occurrence. The blue line represents the percentage of runs in which packet loss occurred at that time. Note that while we plot the red and green dotted line to represent the link drop and restoration point, this does not correspond to the exact drop time in all simulation runs. This can be seen by dropped packets that occur earlier than expected. There are two causes for this effect: the varying start-up time in the emulation,



(a) Path based results (b) Distance vector based results

Fig. 17. Convergence times of all runs in the evaluation. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



(a) Path based (b) Distance vectors

Fig. 18. Packet loss data.

mostly depending on network size, and the fact that packets losses are plotted based on the time their packet entered the network. We believe this is the best approach to keep networks of different sizes comparable as this would otherwise have an effect on the time they are lost.

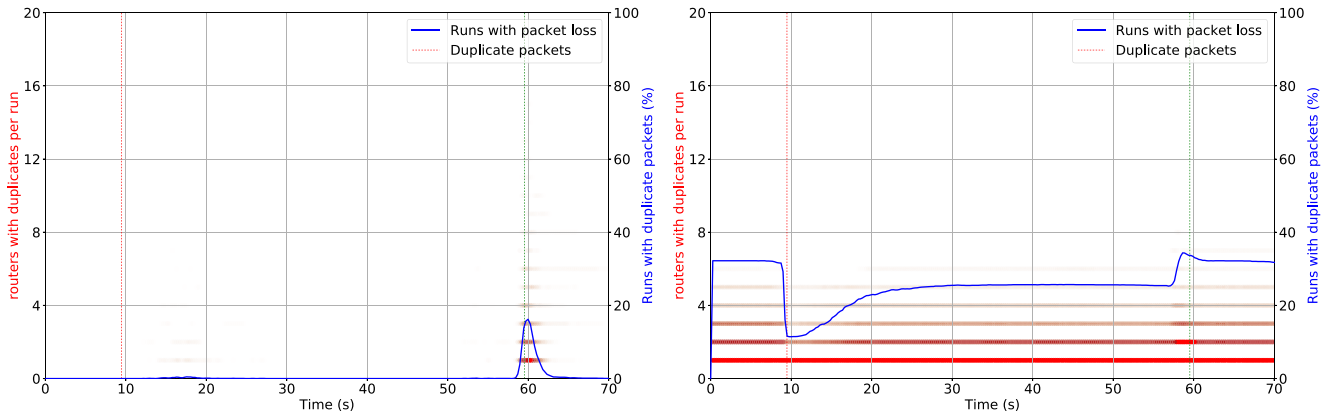
Packet loss mainly occurs after a link drop, and link restoration barely leads to any lost packets. The distance-vector-based routing algorithm actually seems to restore packet delivery slightly faster compared to the path-based algorithm. The packet loss during link restoration for the distance-vector protocol is twice as high compared to the path-based protocol.

While our routing system should not route packets in a loop it does in some specific conditions route packets to a single router through multiple paths. This effect can occur during convergence when the network is temporarily in a configuration that allows a multi path situation to occur. This effect can be observed in Figs. 15a and 15b as a solid red line. We can see this line corresponds to the small peak in path cost directly following the restoration of a link. We show the overall duplicate packet delivery rate for both algorithms in Fig. 19. In these figures the red dots represent the number of routers in a certain run that received a certain packet twice, or more times. As with the loss graphs, the x-axis represents the time at which the packet was sent from the source router measured from the start of a run. The blue line represents the percentage of runs in which a packet was received multiple times by any router. As we can see this effect mainly occurs when a link is restored (or added) to the network. While the path-based approach only has some duplicate deliveries during link restoration, the distance-vector approach appears to have duplicate deliveries all the time. These consistent duplicate packets are caused by the algorithm delivering packets over multiple routes to a single destination in some cases, as explained in Sections 3.3 and 3.4.

Compared to unicast routing we avoid certain issues in our path-based approach, like the count to infinity problem, by relying only on path information. Packets are simply not forwarded any more if a router finds itself on anything but the shortest path from its point of view. On the topic of network change we can conclude that our protocol correctly re-establishes a forwarding tree with only temporary problems, such as a router receiving packets over multiple links, in some cases following a link restoration.

7. Conclusions and future work

In this paper, we have presented a purely distance-vector and a path-based algorithm for geographic routing. We have also presented implementations of both protocols. For the path-based approach, we have shown that the protocol forwards packets along a tree close to the shortest path tree in terms of links used when the network is stable. The Distance-Vector protocol performs worse compared to the path-based one, but is less complex in terms of forwarding rules and required information. We have also shown that during periods where the network changes our protocols can recover in a reasonable time. The time it takes our protocols to recover depends on factors like network size and how well connected the network is.



(a) Path based (b) Distance vector

Fig. 19. Duplicate packet delivery data.

We have shown that our best distance-vector-based algorithm performs relatively well, and in the worst case has only 32% more link cost compared to the shortest path tree. In a situation where the entire network is not addressed this overhead is even lower. Our path-based algorithm can construct forwarding trees to multiple destinations that are close in link cost to the shortest path tree from the source to all destinations. This algorithm improves on the distance-vector-based algorithm, especially in situations where a large number of routers in the network (> 25%) are part of the addressed area.

During network convergence due to link failure we lose packets in both algorithms as can be expected. The algorithms do however recover from this state in a reasonable time. Following a link restoration there is only minimal packet loss in both cases. The path-based protocol does deliver packets over multiple routes to a destination in this situation. While this effect is not desirable it seems limited to certain network topologies and does to some degree prevent packet loss from occurring. The distance-vector approach suffers from duplicate deliveries in more situations.

We believe that the distance-vector algorithm might actually be preferable in certain situations where the extra computational overhead in the router does not outweigh the extra transmission overhead in the network. We expect that this routing approach combined with a hierarchical approach in which autonomous systems advertise one or more areas will eventually allow Internet-wide geocast to become a reality.

To achieve functioning geocast, further work will need to be done. Areas of special interest are hierarchical routing, security and last hop distribution methods.

Hierarchical routing is needed so different autonomous systems can distribute coverage and reachability information. Further work is needed to research methods to extend our current routing proposal to this level.

The are several security issues mostly relating to the possibility of denial of service attacks in the current proposal that need to be addressed. Further work is needed on limiting geocast capabilities to certain users, or limiting the area addressed to prevent wide scale denial of service attacks using geocast.

Work needs to be done on the final hop towards mobile devices, including vehicles. The hop towards end-user devices presents a challenge for geographically scoped communication. There are several different situations in which the method of distribution would need to be tailored to the specific situation. While our addressing method can address small areas, addressing a specific road for example, would still require some form of translation to limit the message to just that road, and not the surrounding area.

For future work, we are planning to improve our information distribution method. In the current implementation coverage area and the path to the covering router are tightly linked. We would like to completely decouple these things to increase scalability. Coverage area information does not need to be advertised as often as path information, and this change could thus decrease overhead. In general our routing protocols, especially with these improvements, can provide another step in enabling Internet-wide geocast in the future.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] G. Karagiannis, G. Heijenk, A. Festag, A. Petrescu, A. Chaiken, Internet-wide geo-networking problem statement, URL https://tools.ietf.org/html/draft-karagiannis-problem-statement-geonetworking-01.

[2] J.C. Navas, T. Imielinski, GeoCast - geographic addressing and routing, in: L. Pap, K. Sohraby, D.B. Johnson, C. Rose (Eds.), MOBICOM, ACM, 1997, pp. 66–76.

[3] F. Cunha, L. Villas, A. Boukerche, G. Maia, A. Viana, R.A. Mini, A.A. Loureiro, Data communication in vanets: Protocols, applications and challenges, Ad Hoc Netw. 44 (2016) 90–103.

[4] T. Fioreze, G.J. Heijenk, Extending the domain name system (DNS) to provide geographical addressing towards vehicular ad-hoc networks (VANETs), in: O. Altintas, W. Chen, G.J. Heijenk (Eds.), VNC, IEEE, 2011, pp. 70–77.

[5] J.C. Navas, T. Imielinski, On reducing the computational cost of Geographic Routing, Rutgers University, Department of Computer Science, Tech. Rep. DCS-TR-408.

[6] T. Imielinski, S. Goel, Dataspace - querying and monitoring deeply networked collections in physical space, in: MobiDE, ACM, 1999, pp. 44–51.

[7] M. Di Felice, L. Bedogni, L. Bononi, Group communication on highways: An evaluation study of geocast protocols and applications, Ad Hoc Netw. 11 (3) (2013) 818–832.

[8] B.T. Sharef, R.A. Alsaqour, M. Ismail, Vehicular communication ad hoc routing protocols: A survey, J. Netw. Comput. Appl. 40 (2014) 363–396.

[9] J. Liu, J. Wan, Q. Wang, P. Deng, K. Zhou, Y. Qiao, A survey on position-based routing for vehicular ad hoc networks, Telecommun. Syst. 62 (1) (2016) 15–30.

[10] Y.-B. Ko, N.H. Vaidya, Geotora: A protocol for geocasting in mobile ad hoc networks, in: Network Protocols, 2000. Proceedings. 2000 International Conference on, IEEE, 2000, pp. 240–250.

[11] B. Karp, H.-T. Kung, Gpsr: Greedy perimeter stateless routing for wireless networks, in: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, ACM, 2000, pp. 243–254.

[12] J. Li, J. Jannotti, D.S. De Couto, D.R. Karger, R. Morris, A scalable location service for geographic ad hoc routing, in: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, ACM, 2000, pp. 120–130.

[13] D. Farinacci, T. Li, S. Hanks, D. Meyer, P. Traina, Protocol independent multicast-dense mode (pim-dm): Protocol specification (revised), RFC 3973.

[14] B. Fenner, M. Handley, I. Kouvelas, H. Holbrook, Protocol independent multicast-sparse mode (PIM-SM): protocol specification (revised).

[15] D. Moscoviter, M. Gholibeigi, B. Meijerink, R. Kooijman, P. Krijger, G. Heijenk, Improving spatial indexing and searching for location-based dns queries, in: International Conference on Wired/Wireless Internet Communication, Springer, 2016, pp. 187–198.
[16] B. Meijerink, M. Baratchi, G. Heijenk, An efficient geographical addressing scheme for the internet, in: International Conference on Wired/Wireless Internet Communication, Springer, 2016, pp. 78–90.
[17] L. Kou, G. Markowsky, L. Berman, A fast algorithm for steiner trees, Acta Inform. 15 (2) (1981) 141–145.
[18] M. Doar, I. Leslie, How bad is naive multicast routing? in: INFOCOM'93. Proceedings. Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future, IEEE, IEEE, 1993, pp. 82–89.
[19] U.T. Nguyen, J. Xu, Multicast routing in wireless mesh networks: Minimum cost trees or shortest path trees? IEEE Commun. Mag. 45 (11) (2007) 72–77.
[20] B. Meijerink, M. Baratchi, G. Heijenk, Evaluation of geocast routing trees on random and actual networks, in: International Conference on Wired/Wireless Internet Communication, Springer, 2017, pp. 127–142.
[21] S. Knight, H. Nguyen, N. Falkner, R. Bowden, M. Roughan, The internet topology zoo, IEEE J. Sel. Areas Commun. 29 (9) (2011) 1765–1775, http://dx.doi.org/10.1109/JSAC.2011.111002.
[22] Y. Rekhter, T. Li, S. Hares, A border gateway protocol 4 (BGP-4), Tech. rep., 2005.
[23] A.A. Hagberg, D.A. Schult, P.J. Swart, Exploring network structure, dynamics, and function using NetworkX, in: Proceedings of the 7th Python in Science Conference (SciPy2008), Pasadena, CA USA, 2008, pp. 11–15.
[24] Mininet Project, Mininet. URL http://mininet.org.
[25] Linux Foundation, netem. URL http://www.linuxfoundation.org/collaborate/workgroups/networking/netem, 2009.