Local Fast Failover Routing on Directed Networks

Jonas Grobe
TU Dortmund
Dortmund, Germany
jonas.grobe@tu-dortmund.de

Stephanie Althoff
TU Dortmund
Dortmund, Germany
stephanie.althoff@tu-dortmund.de

Klaus-Tycho Foerster
TU Dortmund
Dortmund, Germany
klaus-tycho.foerster@tu-dortmund.de

***Abstract*—The fast and resilient routing of packets in networks presents a complex problem with high practical relevance due to its application in local/wide area networks and the Internet in general: nearly all modern communication networks implement some form of re-routing to retain connectivity after failures. Therefore a lot of research regarding this problem has been conducted, but it is almost exclusively focused on the prevalent case of undirected networks.**

In this paper we hence provide an overview of the current research concerning fast rerouting algorithms and the unique challenges of re-routing on directed graphs, in particular the transferability of existing algorithms to this special case.

Subsequently, one of these algorithms, called Keep Forwarding, originally designed for undirected graphs, is adapted for directed graphs and then evaluated against other algorithms in a simulation on synthetic (random and disk graphs) and real-world (from the Heathland sensor network experiment) topologies. We show that our adaptation is superior to the other algorithms on most topologies, with the average packet loss and stretch being up to 61 % and 65 % lower than the second-best respectively on the real-world topology.

***Index Terms*—Network resilience, fast reroute, local failover**

I. Introduction

Today, we heavily rely on networks. The most famous representative might be the Internet which is used for various applications, such as online shopping and video calls. As can be seen in [1] the use of the Internet is following an upward trend. But with the resulting increase in network scale, link failures occur more frequently which can lead to unwanted and costly outages [2]. Due to stringent dependability requirements, networks need to be able to react quickly to those failures and find alternative paths to route packages to their intended destination.

In networks it is common to distinguish between the control plane and the data plane [3]. The data plane is responsible for forwarding packets based on locally available information, while the rules guiding this forwarding process are established by the control plane. However, in the event of network failures, packet forwarding in the data plane is delayed until new paths have been constructed for all impacted routers in the control plane. This process can introduce significant delays, sometimes reaching several hundred milliseconds [3], [4], which can noticeably disrupt crucial internet applications, including web page loading, online gaming, and voice chats [4]–[6].

Therefore, modern networks employ fast failover routing, operating on the data plane, to deal with failures up to several orders of magnitude faster [3], [7]. Fast failover routing uses pre-computed paths so that traffic can be redirected around the failure. Fast failover routing on graphs is a thoroughly researched problem (see, e.g., the recent survey by Chiesa et al. [4]) with high practical relevance, especially through its application in the Internet. But routing problems are also found in sensor networks [8] and can be used to model a range of practical problems like logistics networks [9] or power distribution [10].

However, research in this area mostly focuses on undirected or bidirected graphs, when, on the other hand, especially in networks such as sensor networks, links can also be directed. In this paper we hence focus on the less researched topic of fast failover routing on *directed* graphs and analyze *oblivious fast failover* routing algorithms. These algorithms construct alternative paths proactively and utilize solely the incoming port, source node, and target node information to determine the next hop during packet forwarding, thus allowing for straightforward implementation through routing tables.

The advantage of oblivious routing is its simple implementation, because packets do not need to be extended with additional information and outgoing ports can be determined via routing tables. This proactive approach allows for swift responses to failures, enabling route switching within the data plane within microseconds [4]. Furthermore, as the packet itself remains unaltered by such an oblivious routing, other network functions and protocols remain undisturbed.

Our Contributions. In this paper, we present our contributions towards evaluating routing algorithms on directed graphs. First, we have developed an easily expandable Python-based simulation framework tailored for assessing routing algorithms, which we provide at https://github.com/jonas-grobe/directed-routing. Leveraging this simulation environment, we conducted evaluations on the performance of Bonsai [11] and Grafting [12] algorithms when applied to directed graphs, including wireless graphs and a real-world sensor network topology. Moreover, we successfully adapted the Keep Forwarding [13] algorithm to accommodate directed graphs, and our modified version demonstrated promising results with average packet loss up to 72 % lower and average stretch[1] up to 76 % lower than Bonsai, although heavily depending on the graph type. The runtime was also lower compared to Bonsai and Grafting (§IV).

[1]Stretch – difference between utilized path length and shortest path.

Organization. We next briefly survey the background and related work in §II. In §III, we then introduce the algorithms Bonsai, Grafting, and Keep Forwarding in more detail, assessing their suitability for directed graphs. We then adapt Keep Forwarding to also support directed graphs. The subsequent evaluation of the algorithms is described in §IV, where we explain our simulation method, introduce the metrics we use to compare the algorithms and present our results. A conclusion of our results is presented in §V.

II. Background and Related Work

Failures often appear in all types of computer networks [14], e.g., ISPs [15], [16], data centers [2], [17], and cloud provider WANs [18], with many further outages being reported on, e.g., [19]–[22]. As such, there has been much research on fast recovery of such failures, we refer to the recent survey by Chiesa et al. for an overview [4]. Going forward, we will focus on link failures, which can also be used to model node failures, by simply failing all links incident to a node.

The Control Plane: The Second Line of Defense. A lot of effort has been spent on control plane methods [23], [24], such as link reversal [25], [26], fast (re)convergence [27], or general SDN schemes [28], [29], but these methods are orders of magnitude slower than reactions in the data plane [30]. Hence, we aim to investigate faster recovery methods, with a focus on the data plane.

The Data Plane: The First Line of Defense. There has also been no shortage[2] on fast recovery methods against link failures in the data plane, implementations are available for many common protocols such as MPLS [31], [32], Segment Routing [33]–[36], IP in general [37], [38], BGP [39], and for more advanced hardware in the form of programmable data planes and SDN [40], [41]. Many of these algorithms (e.g., MPLS and Segment Routing) require packet (header) changes or even programmable state on the switches and routers, to, e.g., exploit failure-carrying of encountered failures [42] or to, e.g., perform graph exploration with rotor router algorithms [43]. As powerful as these approaches may be, they require specialized hardware and/or protocol changes and hence are not commonly available.

Oblivious Fast Rerouting Mechanisms on (Un-)Directed Graphs. Motivated by the above downsides, research has also investigated how to perform fast rerouting in the presence of link failures without packet or state modifications.[3] Akin to oblivous routing [46], the rerouting algorithm may rely solely on local information, such as incident failures, the incoming packet's port, and its source and destination, to make forwarding decisions via match-actions. The key question is whether the packet can still reach its destination node from the source node, as long as both nodes are still in the same connected component post-failures. As the focus of our paper is on directed graphs under the above oblivious setting, we now also discuss the applicability of the research in this area to directed graphs; to the best of our knowledge, no prior study was performed for oblivious fast rerouting on directed graphs.

Feigenbaum et al. [7] showed that one failure can be survived in this setting, even without source-information, but two failures are then impossible to survive in general [47], though often still achievable in practice [48]. When including the source, the goal post shifts by one, i.e., resilience to two failures is possible and to three failures is impossible [49]. In general, resilience to an arbitrary number of failures is essentially only possible on outerplanar graphs [50]. These works do not translate well to directed graphs, as they heavily rely on bouncing back the last hops, traversing geometric faces [51], or hitting the failure again from the other direction, which is not possible in directed graphs in general. The same downsides also apply to methods that successively try to explore link-disjoint paths [52] or trees [53]. Here, for each source-target node pairing, paths/trees are constructed, whose leaves are neighbors of the target node. During routing, packets are transmitted through a depth-first search (DFS) to reach the designated target, where however such a DFS cannot "go back" in directed graphs.

When leveraging higher graph connectivity, one can build multiple spanning rooted arborescences (trees) [54], changing to the next arborescence after each failure. With respect to directed graphs, this general method is also applicable, as a strongly k-connected directed graph allows for k of such arborescences. In undirected graphs, earlier work focused purely on the number of arborescences [47], with further work investigating restricted graph classes [55], Shared Risk Links Groups (SRLGs) [56], and short rerouting paths in Bonsai [11]. As Bonsai is applicable to general (directed) graphs and focuses on low stretch in combination with high resilience, we will further investigate its performance on directed graphs later in this paper, along with the so-called grafting approach by Foerster et al. [12], which also uses non-spanning arborescences. Without graph connectivity guarantees however, recent parallel work [57] showed that already one single failure cannot be survived in directed graphs, even if all nodes retain a path to the destination post-failure.

By forgoing resilience guarantees, prior work also investigated greedy fast rerouting approaches, most prominently Keep Forwarding by Yang et al. [13]: herein, one attempts to route the packet closer to the destination, and if that fails, trying to keep the distance at least the same in traversals, or, if all fails, going a bit back, before trying again to get closer. In principle, the approach is also applicable to directed graphs, but the (central to the algorithm) used traversal approach does not translate well to directed links, which we further investigate as well later in this paper.

In summary, although there has been no direct prior investigation of oblivous fast rerouting algorithms on directed graphs, some prior work can be applied in this context. In the next section, we focus on translating the most promising methods, before later implementing and evaluating them.

[2] see again, e.g., the survey by Chiesa et al. [4]

[3] We note that there is also randomized fast rerouting, e.g., [44], [45], which however suffers from, e.g., TCP reordering problems and is generally not available on routers. We thus focus on deterministic rerouting.

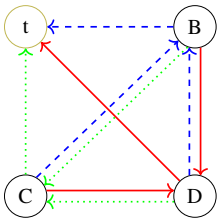

Fig. 1. Graph with three arc-disjoint t-rooted spanning arborescences.

III. Algorithms

In this section, we introduce three oblivious fast rerouting algorithms: Bonsai [11] in Section III-A, Grafting [12] in Section III-B, and Keep Forwarding [13] in Section III-C. We conclude that the first two algorithms can be used for directed routing without modification. However, this is not the case for Keep Forwarding, as we explain in the corresponding section, before presenting a possible adaptation in Section III-D. Each algorithm introduced in this section uses *oblivous fast failover* routing. Therefore, we briefly recall the definition of oblivious routing:

Definition 1: Oblivious Routing [46]: A routing algorithm is *oblivious* if the router only uses following information for packet forwarding: Incoming port through which the packet arrived, source node and target node.

In order to extend this definition to the setting of failover routing, we must also include the node-incident failures, as else the rerouting algorithm could not distinguish between links with and without failures.[4]

Definition 2: An *Oblivous Fast Failover Routing* is an oblivious routing algorithm which may also match on the incident link failures for packet forwarding.

Armed with these definitions, we now discuss the algorithms in more detail.

A. Bonsai

In the following section we describe Bonsai [11] – an oblivious fast failover routing algorithm based on arc-disjoint arborescences. Essentially, we will consider arborescences as connected spanning subgraphs s.t. each node has an outgoing arc, except for a sink, which has none. We next define arborescences and give an example in Fig. 1.

Definition 3: Arborescence [11]: Let (u, v) denote a directed arc from node u to v. A directed subgraph T is an r-rooted spanning arborescence of G if (i) $r \in V(G)$, (ii) $V(T) = V(G)$, (iii) r is the only node without outgoing arcs and (iv), for each $v \in V \setminus \{r\}$, there exists a single directed path from v to r. When it is clear from the context, we use the term "arborescence" to refer to a t-rooted spanning arborescence, where t is the destination node.

To minimise stretch and packet loss, flat (i.e., low depth) arborescences are preferred. The construction of the maximum number of arborescences as flat as possible is NP-hard [58] [11], but the paper introduces three heuristics in polynomial time: Greedy, Round-Robin and Random. Knowledge of the incoming port and the target node of the packet suffice to determine the current arborescence the packet is routed on. If the next edge on the arborescence failed, the router switches to another arborescence, thus providing some resilience.

Greedy works especially well with low failure rates, where only the flattest arborescences are used. Higher failure rates force the use of deeper arborescences, which leads to round-robin performing better in these cases. The authors of Bonsai recommend to use these two variants over random.

Although only used on undirected graphs in the original paper, Bonsai can also be used on directed graphs, because arborescences are *directed* subgraphs.

B. Grafting

In the case of Bonsai, only spanning arborescences are constructed, allowing any arborescence to be used from any node during routing. However, this can result in many unused edges and a single low-degree node can significantly reduce the number of spanning arborescences and resilience.

Grafting [12] provides a solution by enabling the construction and connection of multiple non-spanning arborescences. Three approaches have been developed in the Grafting paper [12]: DAG-FRR, Cluster-FRR, Augment-FRR. For our experiments we chose DAG-FRR, as it yielded the best results in preliminary experiments. Cluster-FRR struggled to find many clusters in the sampled graphs, and Augment-FRR exhibited high variance in its results, as shown in [12].

Like Bonsai, Grafting algorithms were originally demonstrated on undirected graphs, but can also be used on directed graphs without modifications, as the underlying concepts are inherently directed.

C. Keep Forwarding

Keep Forwarding (KF) is a different routing approach which is not based on arborescences. Instead, the network is separated into layers, depending on each node's distance to the target node. The packet is then routed to layers near the target. KF relies on two key concepts: Partial Structural Networks and c-KF Traversals, which we introduce in the following two definitions, before we describe its rerouting in more detail.

Definition 4: Partial Structural Network [13]: A Partial Structural Network (PSN) is constructed for each destination node. To achieve this, all other nodes are categorized into layers, referred to as A-layers. Nodes within the same layer have the same distance to the destination node. Connections within a layer are called A-links, while connections between layers are referred to as M-links (Downlinks if they lead to a layer closer to the destination node, Uplinks otherwise).
An example of a PSN can be seen in Fig. 2.

Definition 5: c-KF Traversal [13]: A traversal on an undirected graph $G = (V, E)$, which visits every node $v \in V$ at least once and every edge $e \in E$ at least once, without traversing any edge more than once in the same direction. It can be constructed on any undirected connected graph (for

[4]This concept is sometimes also denoted as static fast failover routing.

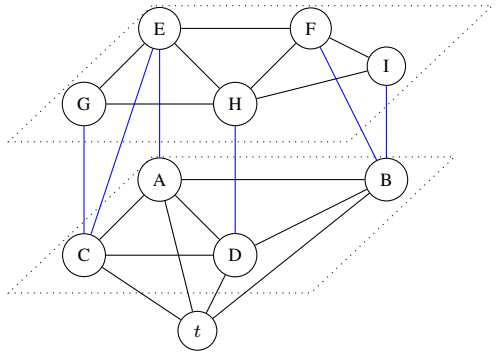

Fig. 2. PSN of target t with two layers and M-Links between (analog to [13]).

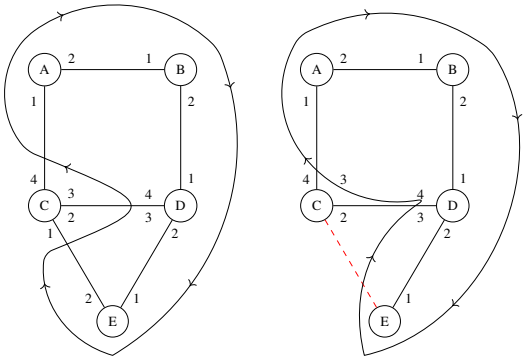

Fig. 3. A c-KF Traversal on a graph. The failure of edge E-C leads to E sending the packet back to D, because there is no other edge available. Result is another traversal encompassing all nodes and all working edges – c-KF Traversals have some failure resilience (analog to [13]).

construction details, see [13]). For an example of a c-KF Traversal, see Fig. 3.

Routing in KF is precomputed in three steps [13]:

1) The PSN for each potential destination node is computed, see Definition 4.
2) Outgoing connections of each node are assigned priorities. Downlinks, which bring the packet closer to the destination, receive the highest priority. If no downlinks are available, packets get routed on the current A-Layer using a c-KF Traversal to visit every node in search of a node with a downlink. Uplinks are only used, if no other links are available. The incoming edge is only used to send the packet back if no other edge with the same or a better link type is available.
3) Finally, a routing table is constructed using connections with different priorities. This table provides an ordered list of further connections based on the destination of the packet and the incoming port.

D. Keep Forwarding Extension

The basic approach of routing preferably along downlinks can be transferred to directed environments. However, when no downlinks are available or all downlinks failed, a method to route the packet on the A-layer to other nodes is needed. In the original implementation a c-KF Traversal is used, but the existence of such a traversal is only guaranteed on undirected graphs. To mitigate this limitation, we partition each A-layer into strongly connected components and for each try to construct a c-KF Traversal. If that is impossible, we use a depth-first search (DFS) to route the packet to as many nodes as possible.

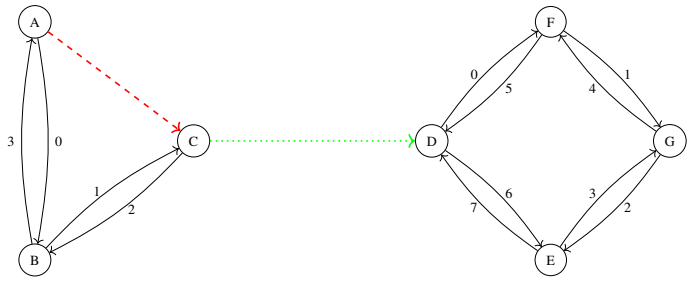

Fig. 4. This A-layer graph consists of two strongly connected components: {A,B,C} and {D,E,F,G}. On the left component a DFS is constructed (order given by edge annotations). The dashed red edge is not used in the DFS and added as low-priority A-link to be used only when edge A-B failed. The dotted green edge connects both components and because it leads to the bigger component, it is added as high-priority A-link. It will be used if C has no downlink available. The right component is traversed via c-KF Traversal.

This makes the algorithm usable, but leads to unused edges and lower resilience (demonstrated in Fig. 4).

Nonetheless, that approach already demonstrated good performance in our first evaluations, but still left 70 % - 90 % of A-links unusable, as they were not used in DFS or were between components – an undesirable situation. We were able to further improve the performance by adding following links to the routing:

- Links from a component to another, bigger component were added as high-priority A-links. They are used in all cases, where no downlink is available.
- Other links (i.e. links to smaller components or links not used by DFS) were added as low-priority A-links, used in routing only if no downlink and no other A-links are available.

This enables full use of all graph edges leading to a node that is connected to the destination. In that case the edge must be a downlink, uplink or A-link and will be used, at least if all other edges on the starting node fail.

IV. Evaluation

In the first Section IV-A we define the metrics used to evaluate and compare the algorithms. The random generation of graphs and failures for the simulation is explained in the following Section IV-B. Simulation results for the different metrics are presented in Sections IV-C and IV-D. Finally, we discuss all results in Section IV-E.

A. Metrics

To evaluate the algorithms, a set of metrics is needed. We chose the following two to highlight different aspects which are also relevant to practical use:

- Packet loss – % of deliverable packets lost (edge failures taken into account).

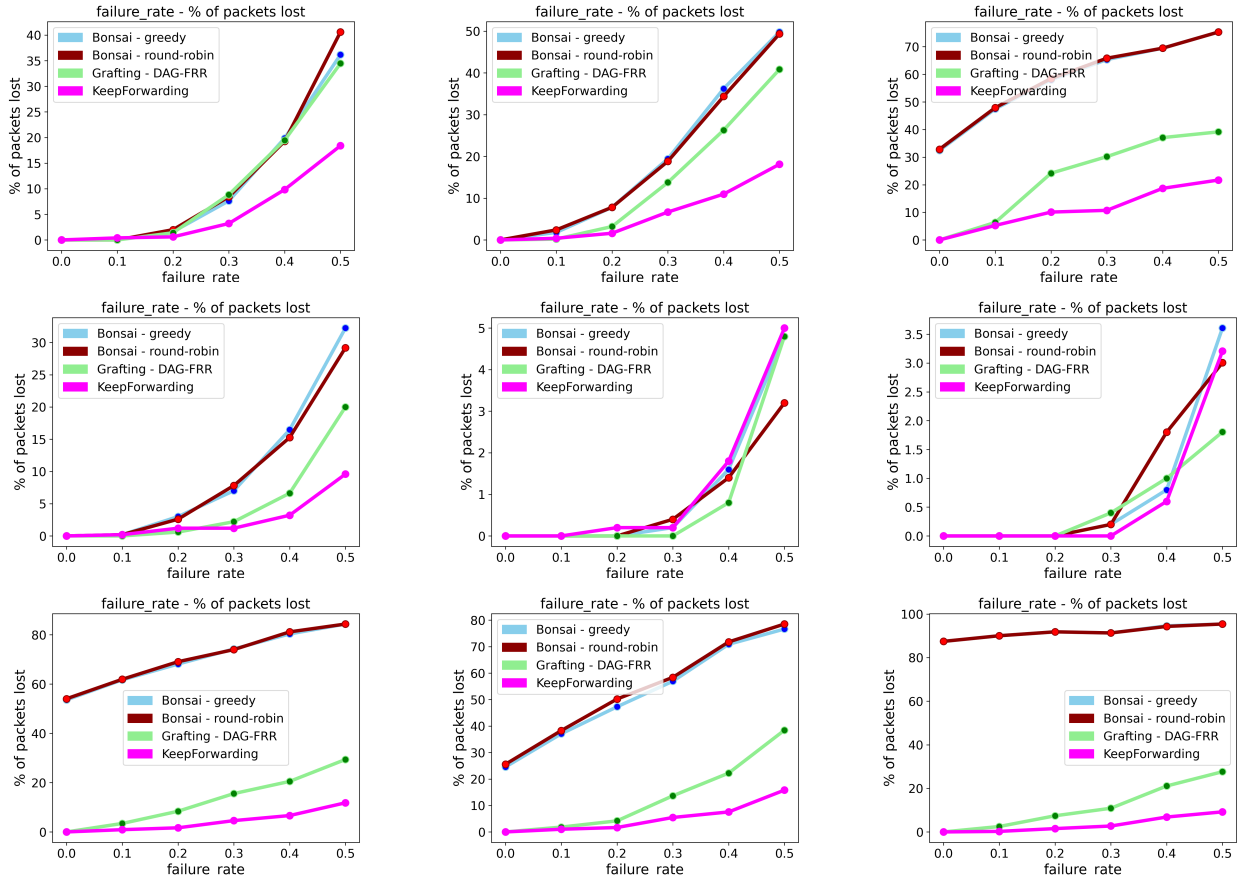

Fig. 5. Packet loss of different routing algorithms. Directed random regular graph (25 nodes, degree of 6, 10 %/30 %/75 % of edges directed left to right). Second row Directed Erdős-Rényi graph. Parameters: ($n = 25$, $p = 0.35$), ($n = 50$, $p = 0.35$), ($n = 25$, $p = 0.6$) left to right. Third row: Wireless graph (25/50/25 nodes, area of size 50x50/80x80/50x50, sending range between (10,20)/(15,25)/(5,25)). 500 iterations per data point.

- Stretch – difference between the length of the path that is utilized and the length of the shortest possible path (edge failures taken into account).

A high stretch typically correlates with a high packet loss, as each edge increases the risk of loss. However, since the path length of lost packets is not considered in the calculation of the stretch, it is possible to have a low stretch combined with a high loss rate, warranting a separate consideration.

B. Graph and failure generation

As our experiments show, algorithm performance depends on the graphs used. To cover a range of different graph topologies, we use three methods for random generation in combination with different parameters:

- Erdős-Rényi - number n of nodes and probability p of each possible edge being created are given [59].
- Random regular – Creates a random graph of n nodes, each having a degree of d. A fraction p of random edges is then directed in a random direction.
- Wireless – Generates a directed graph of n nodes, simulating a wireless communication network. Nodes are placed uniformly at random in a specified area, each assigned a sending range picked uniformly at random from some interval $[a, b]$ with $0 < a \leq b$. Directed edges connect nodes as follows: a node u with sending range r can send to a node v if and only if the distance between u and v is at most r.[5]

Additionally, we simulated all algorithms on a real-world graph from the Heathland Experiment - a wireless sensor network in a harsh environment with directed edges (Fig. 7).

In randomly generated graphs, both the source and target nodes are chosen randomly. In the Heathland graph, the target node is predetermined (as in the original experiment setup), while the source node is selected randomly.

To evaluate algorithm performance under failures, we simulate uniformly distributed edge failures (the same method as in [12]), varying failure rates between 0 % and 50 %.

C. Packet loss

Fig. 5 and Fig. 8 show the packet loss of different routing algorithms on different graphs. On directed random regular graphs and the Heathland graph, KF has the lowest packet loss (average packet loss between 50 % to 80 % lower compared to second best algorithm for each case), while Grafting performs better than Bonsai. On wireless and random regular graphs with 75 % of the edges being directed, no

[5]If $a = b = 1$, this would be a unit disk graph [60].

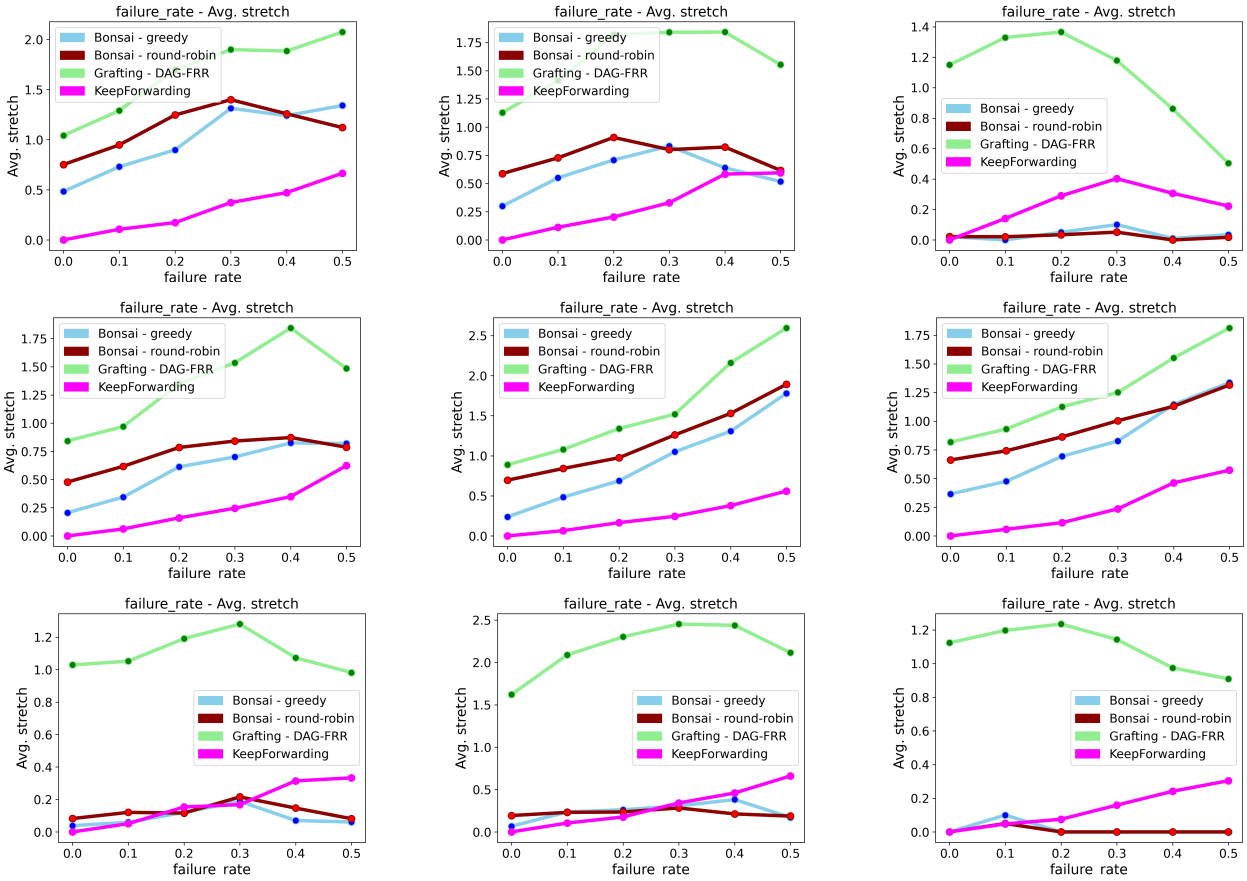

Fig. 6. Stretch of different routing algorithms. Directed random regular graph (25 nodes, degree of 6, 10 %/30 %/75 % of edges directed left to right). Second row Directed Erdős-Rényi graph. Parameters: ($n = 25$, $p = 0.35$), ($n = 50$, $p = 0.35$), ($n = 25$, $p = 0.6$) left to right. Third row: Wireless graph (25/50/25 nodes, area of size 50x50/80x80/50x50, sending range between (10,20)/(15,25)/(5,25)). 500 iterations per data point.

spanning arborescence can be constructed, leading to high packet loss even at 0 % edge failure rate for Bonsai. Grafting, which enables the construction of non-spanning arborescences, does not have this issue.

On directed Erdős-Rényi graphs, the outperformance of KF is not as significant. On Erdős-Rényi graphs with $n = 50$ and $p = 0.35$, KF has the highest average packet loss – 44 % higher than Bonsai round-robin.

D. Stretch

Fig. 6 and Fig. 8 show the stretch of different routing algorithms on different graphs. Grafting has the highest stretch on all graphs. Bonsai has a lower stretch, with a remarkably low stretch on wireless and regular random graphs with 75 % directed edges. This can be explained in conjunction with the high packet loss: Packets can either be directly routed to target or be lost.

Apart from these special cases, KF has the lowest (average stretch between 48 % to 85 % lower compared to second best algorithm for each case) stretch and is the only algorithm with the optimal stretch of 0 when no failure occurs on every graph type. This is an advantage for everyday routing in practice, as this enables the implementation of resilient routing via KF with no extra stretch if no failures occur.

E. Discussion

Packet loss and stretch of the different algorithms depend on the graph type examined. On directed random regular graphs, KF exhibited the lowest packet loss and also the lowest stretch (with exception of 75% directed edges, where Bonsai had a lower stretch but also an extremely high packet loss). On the directed Erdős-Rényi graphs however, many arborescences could be constructed and the difference in performance of the algorithms was lower – in the case of $(n = 50, p = 0.35)$ average packet loss for KF was 44 % and 29 % higher than for Bonsai and Grafting respectively.

Regarding computational runtime, our adapted Keep Forwarding was the fastest, staying well below 0.1 seconds for all instances, whereas Grafting and the greedy Bonsai variant went above 3.5 and 1.0 seconds for 50 nodes, respectively, showing a strong upwards trend as instance size increased with the round robin version staying between greedy and KF.

We conclude that our modified KF has advantages over Bonsai and Grafting, especially on smaller and sparsely connected graphs (where especially Bonsai has problems due to the limited number of spanning arborescences) and low failure rate environments (KF is the only algorithm with a stretch of 0 at failure rate 0 % on all graphs). Moreover, the

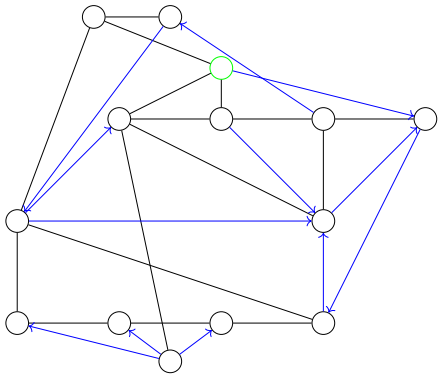

Fig. 7. Heathland experiment graph (analog to [8], [61]). Directed edges are highlighted in blue. The target node used in the original experiment setup is marked in green.

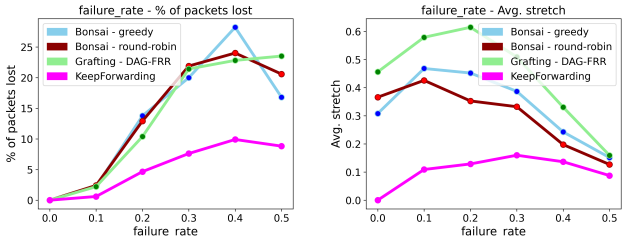

Fig. 8. Packet loss and stretch of different routing algorithms. Heathland graph. 500 iterations per data point.

computational runtime for KF is low in comparison to the other three algorithms.

V. Conclusion

In this paper, we initiated the investigation of local fast failover routing on directed networks. We surveyed the literature for suitable methods and investigated three routing algorithms originally devised for undirected graphs. Remarkably, two of these algorithms, namely Bonsai and Grafting, seamlessly extend to directed graphs without necessitating any modifications, whereas KF needed to be adapted and extended. The modified KF then had the lowest runtime and lowest stretch. Its packet loss depends on the random graph generator used – while it exhibited the lowest packet loss on random regular graphs, Bonsai and Grafting outperformed it on Erdős-Rényi graphs. However, on the real-word Heathland graph, KF again had the lowest packet loss of all algorithms. In order to facilitate further research, we provide our implementation at https://github.com/jonas-grobe/directed-routing.

A. Outlook

In this paper, we examined the routing problem within a simplified network model with precomputation using full graph knowledge and considered only single routing tasks with the goal of minimizing packet loss and stretch. A more realistic model could simulate multiple concurrent routing tasks with a high load on edges or nodes leading to delays and losses [62], as well as short-lived failures due to e.g. link flapping [63]. Under these circumstances, further adjustments to the routing algorithms presented here may be necessary.

Furthermore, we measured path length and delay in hops and assumed an evenly distributed failure rate, while in practice each edge might have different parameters describing average delay and failure probability [10].

It is important to note that our investigation solely focused on data plane algorithms. In practice these would rely on control plane algorithms for construction. Established control plane algorithms like Distance Vector Protocols (neighbours exchange routing tables and update accordingly) or Link State Protocols (neighbours exchange connection information) [64] are designed for undirected environments, similar for bidirectional communication with the (logically) centralized controller in Sofware Defined Networks [65]. For practical use, their performance in directed networks should be evaluated, and the algorithms should be modified or replaced to enhance performance. One might even need to consider manual "one-off" installation methods, as, e.g., for data source nodes with no incoming connections as in the Heathland experiment.

References

[1] Statista, "Number of internet users worldwide from 2005 to 2022 (in millions)," https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/, 2022.

[2] P. Gill *et al.*, "Understanding network failures in data centers: measurement, analysis, and implications," in *SIGCOMM*. ACM, 2011.

[3] J. Liu *et al.*, "Ensuring connectivity via data plane mechanisms," in *USENIX NSDI*, Lombard, IL, Apr. 2013, pp. 113–126.

[4] M. Chiesa *et al.*, "A survey of fast-recovery mechanisms in packet-switched networks," *IEEE Commun. Surv. Tutorials*, vol. 23, no. 2, pp. 1253–1301, 2021.

[5] J. Young and T. Barth, "Web performance analytics show even 100-millisecond delays can impact customer engagement and online revenue," Akamai Online Retail Performance Report, 2017.

[6] J. Saldan, "Delay limits for real-time services," IETF Draft, 2016.

[7] J. Feigenbaum *et al.*, "Brief announcement: On the resilience of routing tables," in *Proc. ACM PODC*, 2012.

[8] V. Turau *et al.*, "The heathland experiment : results and experiences," in *REALWSN*, no. 5, 2005. [Online]. Available: http://hdl.handle.net/11420/13281

[9] Z. Fan *et al.*, "Research on cold chain logistics path optimization considering cascading failure," *IOP Conference Series: Materials Science and Engineering*, vol. 787, no. 1, p. 012031, mar 2020.

[10] R. Huang *et al.*, "A routing algorithm based on weighted graph for power distribution network," in *Simulation Tools and Techniques*. Springer, 2019, pp. 104–114.

[11] K.-T. Foerster *et al.*, "Bonsai: Efficient fast failover routing using small arborescences," in *IEEE/IFIP DSN*, 2019, pp. 276–288.

[12] ——, "Grafting arborescences for extra resilience of fast rerouting schemes," in *IEEE INFOCOM*, 2021.

[13] B. Yang *et al.*, "Keep forwarding: Towards k-link failure resilient routing," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, 2014, pp. 1617–1625.

[14] J. Rak and D. Hutchison, Eds., *Guide to Disaster-Resilient Communication Networks*, ser. Computer Communications and Networks. Springer, 2020.

[15] A. Markopoulou *et al.*, "Characterization of failures in an operational IP backbone network," *IEEE/ACM Trans. Netw.*, vol. 16, no. 4, pp. 749–762, 2008.

[16] D. Turner *et al.*, "California fault lines: understanding the causes and impact of network failures," in *SIGCOMM*. ACM, 2010.

[17] R. Potharaju and N. Jain, "When the network crumbles: an empirical study of cloud network failures and their impact on services," in *SoCC*. ACM, 2013, pp. 15:1–15:17.

[18] R. Govindan *et al.*, "Evolve or die: High-availability design principles drawn from googles network infrastructure," in *SIGCOMM*, 2016.

[19] D. Madory, "Renesys blog: Large outage in pakistan," https://dyn.com/blog/large-outage-in-pakistan/.

[20] R. Singel, "Fiber optic cable cuts isolate millions from internet, future cuts likely," https://www.wired.com/2008/01/fiber-optic-cab/, 2008.

[21] Wikitech, "Site issue Aug 6 2012," http://wikitech.wikimedia.org/view/Site_issue_Aug_6_2012, 2012, (lLast accessed June 2024).

[22] C. Wilson, "'Dual' fiber cut causes Sprint outage," https://web.archive.org/web/20080906210432/http://telephonyonline.com/access/news/Sprint_service_outage_011006/, 2006.

[23] C. Jiang *et al.*, "PCF: provably resilient flexible routing," in *SIGCOMM*. ACM, 2020.

[24] H. H. Liu *et al.*, "Traffic engineering with forward fault correction," in *SIGCOMM*. ACM, 2014.

[25] E. Gafni and D. P. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," *IEEE Trans. Comm.*, vol. 29, no. 1, pp. 11–18, 1981.

[26] M. S. Corson and A. Ephremides, "A distributed routing algorithm for mobile wireless networks," *Wirel. Netw.*, vol. 1, no. 1, pp. 61–81, 1995.

[27] C. Busch, S. Surapaneni, and S. Tirthapura, "Analysis of link reversal routing algorithms for mobile ad hoc networks," in *SPAA*. ACM, 2003.

[28] M. Markovitch and S. Schmid, "SHEAR: A highly available and flexible network architecture marrying distributed and logically centralized control planes," in *ICNP*, 2015.

[29] A. Vahdat *et al.*, "A purpose-built global network: Google's move to SDN," *Commun. ACM*, vol. 59, no. 3, pp. 46–54, 2016.

[30] J. Liu *et al.*, "Ensuring connectivity via data plane mechanisms," in *NSDI*, 2013.

[31] P. Pan, G. Swallow, and A. Atlas, "Fast reroute extensions to RSVP-TE for LSP tunnels," *RFC*, vol. 4090, pp. 1–38, 2005.

[32] J. S. Jensen *et al.*, "P-Rex: fast verification of MPLS networks with multiple link failures," in *CoNEXT*. ACM, 2018.

[33] A. Bashandy, C. Filsfils, B. Decraene, S. Litkowski, P. Francois, D. Voyer, F. Clad, and P. Camarillo, "Topology independent fast reroute using segment routing," *Working Draft, Internet-Draft draft-bashandy-rtgwg-segment-routing-tilfa-05*, 2018.

[34] K.-T. Foerster *et al.*, "TI-MFA: keep calm and reroute segments fast," in *Global Internet Symposium*. IEEE, 2018.

[35] ——, "Local fast segment rerouting on hypercubes," in *OPODIS*, 2018.

[36] M. Parham *et al.*, "Maximally resilient replacement paths for a family of product graphs," in *OPODIS*, 2020.

[37] A. Atlas and A. Zinin, "Basic specification for IP fast reroute: Loop-free alternates," *RFC*, vol. 5286, pp. 1–31, 2008.

[38] G. Rétvári *et al.*, "IP fast reroute: Loop free alternates revisited," in *INFOCOM*. IEEE, 2011.

[39] C. Filsfils *et al.*, "Bgp prefix independent convergence (pic)," *Cisco, San Jose, CA, Tech. Rep*, 2011.

[40] Switch Specification 1.3.1, "OpenFlow," in *https://bit.ly/2VjOO77*, 2013.

[41] M. Chiesa *et al.*, "PURR: a primitive for reconfigurable fast reroute: hope for the best and program for the worst," in *CoNEXT*. ACM, 2019.

[42] K. Lakshminarayanan *et al.*, "Achieving convergence-free routing using failure-carrying packets," in *SIGCOMM*. ACM, 2007, pp. 241–252.

[43] D. Dereniowski, A. Kosowski, D. Pajak, and P. Uznanski, "Bounds on the cover time of parallel rotor walks," in *STACS*, 2014.

[44] M. Chiesa *et al.*, "On the resiliency of randomized routing against multiple edge failures," in *ICALP*, 2016.

[45] G. Bankhamer *et al.*, "Local fast rerouting with low congestion: A randomized approach," in *ICNP*. IEEE, 2019.

[46] Y. Azar *et al.*, "Optimal oblivious routing in polynomial time," in *STOC*, 2003.

[47] M. Chiesa *et al.*, "On the resiliency of static forwarding tables," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1133–1146, 2017.

[48] E. van den Akker and K. Foerster, "Short paper: Towards 2-resilient local failover in destination-based routing," in *ALGOCLOUD*. Springer, 2024.

[49] W. Dai *et al.*, "A tight characterization of fast failover routing: Resiliency to two link failures is possible," in *SPAA*. ACM, 2023, pp. 153–163.

[50] K. Foerster *et al.*, "On the price of locality in static fast rerouting," in *DSN*. IEEE, 2022, pp. 215–226.

[51] G. Karamoussanlis, S. Althoff, E. van den Akker, and K. Foerster, "Analyzing network routing resilience: A hybrid approach of face and tree routing," in *RNDM*. IEEE, 2024.

[52] K.-T. Foerster *et al.*, "Casa: Congestion and stretch aware static fast rerouting," in *IEEE INFOCOM*, 2019, pp. 469–477.

[53] O. Schweiger *et al.*, "Improving the resilience of fast failover routing: Tree (tree routing to extend edge disjoint paths)," in *ACM ANCS*, 2021, p. 1–7.

[54] M. Chiesa *et al.*, "Exploring the limits of static failover routing," *CoRR*, vol. abs/1409.0034, 2014.

[55] K. Foerster *et al.*, "Local fast failover routing with low stretch," *Comput. Commun. Rev.*, vol. 48, no. 1, pp. 35–41, 2018.

[56] ——, "Improved fast rerouting using postprocessing," in *SRDS*. IEEE, 2019, pp. 173–182.

[57] E. van den Akker and K. Foerster, "Brief announcement: On the feasibility of local failover routing on directed graphs," in *SSS*, ser. Lecture Notes in Computer Science. Springer, 2024.

[58] J.-C. Bermond and P. Fraigniaud, "Broadcasting and NP-completeness," in *Graph Theory Notes of New York*, vol. 12, 1992, pp. 8–14.

[59] P. Erdős and A. Rényi, "On random graphs 1." *Publicationes Mathematicae*, vol. 6, no. 3-4, pp. 290–297, 1959.

[60] B. N. Clark *et al.*, "Unit disk graphs," *Discret. Math.*, vol. 86, no. 1-3, pp. 165–177, 1990.

[61] R. Karnapke, "Unidirectional links in wireless sensor networks," doctoralthesis, BTU Cottbus - Senftenberg, 2013.

[62] B. Peis *et al.*, "Packet routing: Complexity and algorithms," in *Approximation and Online Algorithms*, E. Bampis and K. Jansen, Eds. Springer, 2010, pp. 217–228.

[63] W. Dai, K. Foerster, and S. Schmid, "On the resilience of fast failover routing against dynamic link failures," *CoRR*, vol. abs/2410.02021, 2024. [Online]. Available: https://arxiv.org/abs/2410.02021

[64] S. G. Sudip Misra, *Network Routing. Fundamentals, Applications, and Emerging Technologies*. Wiley, 2017.

[65] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2015.