Bachelor thesis

Comparison of Loop-Free Route Update Algorithms in Multi-Stream, Non-Time-Based Software Defined Networks

Luca Strick
232239

September 11, 2025

Begutachtung:
Prof. Dr. Dr. Klaus-Tycho Förster
Prof. Dr. Christian Janiesch

TU Dortmund University
Computer Science Faculty
Lehrstuhl IV - Praktische Informatik
https://ms.cs.tu-dortmund.de/

Abstract

With networks getting more complex and implementing software defined networks, SDN controllers need to change the route that flows take efficiently and without causing packet loss. This thesis looks at multiple route-update algorithms. It compares loop-free algorithms with flow swapping algorithms in non-time-based Software Defined Networks. We create a test environment to evaluate these algorithms on real hardware. In order to reduce the propagation time of route changes, we apply the routes directly using the manufacturer-provided API of our routers. We run four tests on each route change: An ICMP-Echo test, an iperf-TCP test, an iperf-UDP test, and a traceroute test. With this data, we can evaluate the differences between the algorithms. Using flow swapping algorithms, we experience packet loss of up to 3% during a route change in our environment. We additionally developed the Three-Round algorithm to reduce the amount of packet loss, with a fast calculation time and a fixed runtime of three rounds.

All tests are run with four flows at the same time to evaluate the effects of running these algorithms in multi-stream environments.

We argue that there is a conflict between propagation-time and packet loss. No algorithm we analysed is able to achieve good results in both metrics on non-time-based Software Defined Networks. Network administrators trade off their need for a quick route change against their need to not have any packet loss. If it is necessary to have absolutely no packet loss, we recommend using Greedy, as it guarantees loop-freedom and can, under good circumstances, generate fast results. If the requirements for packet loss are not as strict and speed is of most importance, we can recommend using One-Round, as it is always quick.

Contents

1. Introduction

We will first explain the motivation for this thesis in chapter 1.1. After that, we will introduce our main results in chapter 1.2 and the structure of this thesis in chapter 1.3.

1.1. Motivation

Complex networks contain many flows going from multiple sources to multiple destinations. More and more networks use Software Defined Networks to move the routing decision point from the individual routers into a single control plane [Kre+15]. With the rise of Software Defined Networks, a new problem arises: route changes. To change the path data takes through the network without packet loss, it is required that no intermittent loops emerge during route changes [MW13]. There are several solutions proposed for this problem. Round-based loop-free updates are well researched in literature [För+18] [MW13] [Alk24b] [Alk24a], with the downside of taking long to apply changes. Time-based updates are relatively new and promising [MM16b], as they can be a lot faster than round-based ones.

According to Förster, Schmid, and Vissicchio [FSV19] there are many reasons why there might be route changes: Security policy changes, traffic engineering, maintenance work, link failures, or service relocations. Most scenarios where route changes are needed occur during usage, which stresses the importance of implementing route changes without service impacts. Several route-update algorithms that solve this problem exist in literature and need to be evaluated in a realistic environment.

In a previous thesis, Alkhatib [Alk24b] compared the algorithms Greedy, Brute-Force, Backward and their custom developed One-Round algorithm in networks with traffic between one source and one destination. This thesis builds upon this to investigate whether similar algorithms can be used for networks with multiple sources and multiple destinations (multi-stream networks).

Additionally, Zheng et al. [Zhe+19] propose the *Chronicle* algorithm [Zhe+18], which aims to provide a faster and congestion-free solution for multi-stream SDNs. Chronicle requires time-based SDNs and profits from their ability to do changes across multiple routers simultaneously. We replicate this behaviour by sending a command to apply pending changes to all routers at the same time. We investigate the difference in packet loss this causes compared to loop-free route-update algorithms.

There have been, to our knowledge, no studies comparing the packet loss of time-based algorithms in non-time-based networks with multiple flows to traditional round-based algorithms. We strive to close this gap by comparing these algorithms on real world hardware and analyse how they perform under realistic circumstances. This is critical for network administrators to evaluate which route update algorithms best fit their needs. While some networks might have very strict requirements for packet loss, others might value update speed more.

As route updates need to be fast, even in large networks, the speed of the tools proposed needs to be tested, evaluated and compared. We measure the time it takes to generate valid route change schedules, as well as the time it takes to apply the changes onto routers.

1.2. Main Results

In this Thesis, we apply the algorithms Backward, Brute-force, Greedy, and One-Round that were previously tested for single-flow networks to multi-stream networks. We show in our tests that these algorithms can be applied in multi-stream networks with good results.

We also test the algorithms Chronicle and One-Round, that are better suited for time-based Software Defined Networks, in our network. This thesis demonstrates that this is possible and works, but leads to up to 15x more packet loss than loop-free algorithms. The advantage of Chronicle and One-Round is that they need significantly fewer rounds. This translates into time savings of up to 50% during the change.

In order to reduce this packet loss, we develop our own algorithm: Three-Round. We demonstrate that Three-Round produces up to 35% less packet loss than Chronicle and One-Round and can be executed up to 30% faster.

We argue that the choice of tool and computing platform can drastically change the time it takes to facilitate the changes. While Alkhatib [Alk24b] experiences propagation times for one flow of 19 to 123 seconds for their Ansible-based solution, we get propagation times for four flows down to 0.4 to 3 seconds. The choice of a specific algorithm is also important, as is evident by our findings that while Backward could generate loop-free schedules in under a millisecond, Greedy needed 0.17 s and Brute-force up to 1.96 s.

To allow further research, we construct our test environment in an easily reproducible way.

1.3. Structure

This thesis is structured into seven chapters.

In chapter 2, we will introduce and define the terms used in this thesis. Chapter 3 will introduce the five algorithms already known from literature. Our research questions and their background will be explained in chapter 4. We follow this up by describing our own algorithm Three-Round in chapter 5. The setup and components of our experiment will be introduced in chapter 6. Chapter 7 will present results of our experiment. Chapter 8 will discuss the results. The thesis will be concluded and topics for further work will be presented by chapter 9.

2. Background

This chapter introduces various terms, to be used later in this thesis. We will start by defining Software Defined Networks in chapter 2.1, following up with the specific definition of time-based Software Defined Networks in 2.2. Afterwards, we will explain Route Updates in chapter 2.3 and flow swapping in chapter 2.4. We will end by introducing Ansible in chapter 2.5 and RouterOS in Chapter 2.6

2.1. Software Defined Networks

Software defined networks (SDN) differ from typical networks in separating the network's control software from the routers and switches in the network [Kre+15]. A router, often called switch in the context of Software Defined Networks, only receives the forwarding table from the controller. This removes the decision-making process from routers. These tables come with forwarding information that only matches on packet header fields [FSV19].

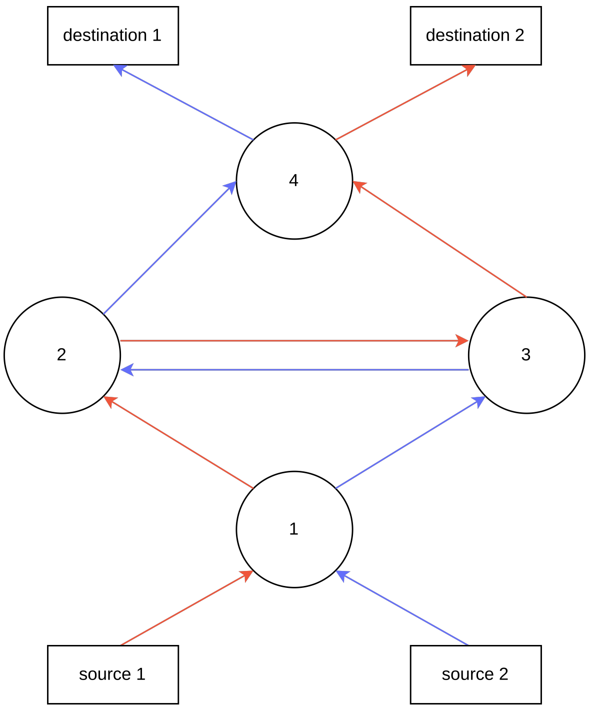


Figure 2.1.: Drawing of a network with four nodes. Shown are two flows (red and blue) both starting at a different source and ending at a different destination

A flow is considered a session between a source to a destination that transfers data at a fixed rate [MM16a]. Each flow has a defined sequence of routers it takes to get from the source to the destination. This sequence can be changed by route updates.

For the purpose of this thesis, we define multi-stream SDNs to have multiple flows, with each flow having a unique tuple of start and destination. An example of a multi-stream SDN is shown in Figure 2.1. We define single-flow SDNs as networks where only one flow exists concurrently.

2.2. Time-Based Software Defined Networks

Time-based/Timed-SDNs were introduced by Mizrahi and Moses [MM16a] to allow for more efficient route updates. Contrary to other updates, the control software does not directly apply route changes. Instead, it adds changes to buffers that are sent to all routers. At a specified time sent with the buffer, all routers apply these changes to their routing table. As the time is synchronized between all routers, this allows network administrators to carry out a change on all routers exactly at the same time, without regards to instabilities of the connection to the routers [Zhe+19].

2.3. Route Updates

Route updates change the forwarding rules in all routers to modify the route a flow takes [För+18]. These operations can also change the routes of multiple flows simultaneously.

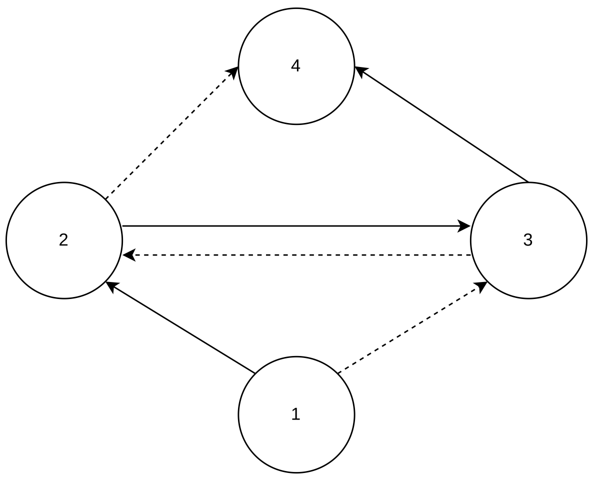


Figure 2.2.: Visualization of a route change in an example network. The straight lines visualize the old route, the dotted lines the new route.

Figure 2.2 illustrates such a route change. While currently traffic flows from router 1 to 2 to 3 to 4 the route needs to be changed, so that traffic flows from 1 to 3 to

2 to 4. While a naive method would be to send all route changes in one shot, this might cause routing loops as router 3 might update before router 2. The routing loop would disappear after router 2 is updated. It is hard to predict the duration of packet loss due to the lack of knowledge about the precise timing and delays between the controller and the routers [MW13]. Multiple solutions proposed in literature try to prevent packet loss during route updates. We present them in the following subchapters.

2.3.1. 2-Phase-Commit protocols

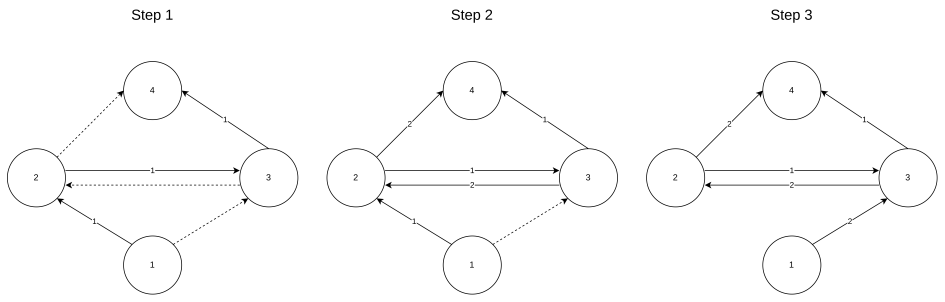


Figure 2.3.: Visualization of a route change in an example network using the 2-phase-commit protocol. The straight line visualizes the old route, the dotted line the new route. The number on the lines is the tag allowed over the link.

2-Phase-Commit protocols work by tagging all packets. When a route update is necessary, it is configured in all routers as a new *tag*. Now this tag can be configured at the beginning of the route. New traffic is then routed using the new rules. This solution requires additional memory, larger header space, and can be problematic when used together with middle boxes [För+18]. Figure 2.3 shows the steps necessary to facilitate a route change using 2-Phase-Commit protocols.

2.3.2. Loop-Free Round-Based algorithms

Loop-free round-based algorithms split the route updates into rounds. Each round starts after all routers have applied the previous round [För+18]. This makes it possible to create loop- and packet loss free update schedules by ensuring that all changes in a round cannot cause loops with other changes in that round. While this does not have the problems 2-Phase-Commits have, it is a lot slower, as updating networks can require up to $\Omega(n)$ rounds, with n being the amount of routers [LMS15]. Förster et al. [För+18] show that finding a loop-free round-based schedule with a minimal amount of rounds is NP-complete if it takes more than 2 rounds.

To create a round-based schedule for our example in figure 2.2, we need to ensure that router 2 and 3 are not updated in the same round [FSV19].

2.3.3. Time-Based Updates

Time-based updates use the features afforded by time-based Software Defined Networks. They allow, similar to 2-Phase-Commits, that all routers are updated at the same time and therefore support flow swapping, as explained in chapter 2.4.

2.4. Flow swapping

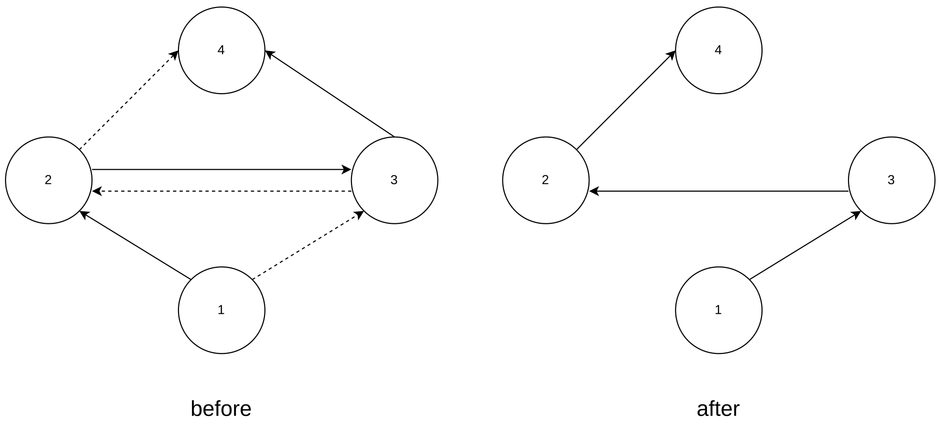


Figure 2.4.: Visualization of a route change in an example network. Shown are the before and after state of the network using flow swapping.

Flow swapping is described by Mizrahi and Moses [MM16a] as updating two or more routers at the same time, as shown in figure 2.4. They argue that flow-swapping is sometimes necessary to avoid congestion and introduce time-based updates to allow for it.

2.5. Ansible

Ansible is an automation tool that can configure servers, routers, and much more [Inc]. Configuration is done by applying *Playbooks* on hosts by executing them using the `ansible-playbook` tool. A Playbook is a `yaml` file defining the desired actions and state of the configuration of computers or routers.

2.6. MikroTik RouterOS

Routers by MikroTik come with MikroTik RouterOS[1] preinstalled. RouterOS handles packet forwarding and firewall tasks. It can either be configured with the configuration tool *Winbox* or through an API [Doc].

[1]https://help.mikrotik.com/docs/spaces/ROS/pages/328119/Getting+started

RouterOS scripts are sequences of RouterOS commands that can be stored and executed on the router.

3. Algorithms

This chapter will introduce the algorithms evaluated by this thesis. We will introduce each Algorithm with their fundamental idea and problems. We start with the Greedy algorithm in Chapter 3.1, followed by the Backward algorithm in Chapter 3.2, the Brute-force algorithm in Chapter 3.3, Chronicle in Chapter 3.4 and One-Round in chapter 3.5.

The chapters will use Pseudocode to explain the algorithms. We will use the following notation:

- *old_route*: List of the routers the flow takes before the route change
- *new_route*: List of the routers the flow takes after the route change
- route **from** A: A route from A to the next router
- route **to** A: A route from any router to router A
- route **from** A **to** B: A route between router A and router B
- r^+: When r is a route change, r^+ only contains the added routes
- **graph from**: Creates a directed graph of a route. Each router is a node, and each route is represented by a directed edge between two nodes.

3.1. Greedy

Greedy algorithms are well-known in literature [DMR21] [Vin02] [Jun99]. They work by *greedily* applying the best option for the current situation [DMR21]. In the case of route updates, that is trying to apply all changes every round. Förster, Schmid, and Vissicchio [FSV19] describe an algorithm to prevent loops in route updates. In each round, all changes that pass two checks are scheduled: First we check that the change does not cause loops, even under the assumption that all previously scheduled changes of the round do not contain removals. We check if the destination would still be reachable, even if no other routes have been added, yet. Additionally, we ensure that the destination is still reachable, even if no other previously scheduled routes in this round have been added. The second check is not described by Förster, Schmid, and Vissicchio [FSV19] in algorithm 1, but important to ensure low packet loss. When all changes are made, the algorithm exits.

```
1: procedure GREEDY($old\_route$, $new\_route$)
2:     required $\leftarrow$ changes between old route and new route
3:     $g \leftarrow$ graph from $old\_route$
4:     $n \leftarrow 0$
5:     while $|\text{required}| > 0$ do
6:         $n \leftarrow n + 1$
```

```
        $b \leftarrow$ copy of $g$                          ▷ Contains state before the round
        $a \leftarrow$ copy of $g$                          ▷ Contains only positive changes
        for each $r \in$ required do
            if $r^+$ does not cause a loop in $a$ and $r$ does not cause connectivity loss in $b$ then
                Apply $r$ in round $n$
                Apply $r$ in $g$
                Apply $r^+$ in $a$
                Remove $r$ from required
            end if
        end for
    end while
end procedure
```

Using a greedy algorithm for calculating route updates can be very inefficient in some cases. Förster et al. [För+18] show that a single wrong choice by the algorithm can increase the amount of rounds from $O(1)$ to $\Omega(n)$, with n being the amount of routers.

3.2. Backward

The Backward algorithm applies all new routes starting with the last router of the new route path and ending with the first one. It changes the forwarding table of one router per round [Mat+16].

```
procedure BACKWARD($old\_route$, $new\_route$)
    $n \leftarrow 0$
    for $i \leftarrow |new\_route| - 2$ to $0$ do
        if $new\_route[i]$ is in $old\_route$ then
            if route from $new\_route[i]$ to $new\_route[i+1]$ is in $old\_route$ then
                continue
            end if
            Remove route from $new\_route[i]$ in round $n$
        end if
        $n \leftarrow n + 1$                    ▷ Only increase if there are changes
        Add route from $new\_route[i]$ to $new\_route[i+1]$ in round $n$
    end for
end procedure
```

By going from the back to the front of the new route one router per round, there can never be loops [Ami+16] [MW13]. We modified this algorithm slightly. We skip rounds that would cause no changes. This way the algorithm can use less than n rounds, if some routes are already correctly set.

The algorithm will always produce a solution with (n − *routes that are already set*) rounds, even if a simpler schedule with fewer rounds exists.

3.3. Brute-Force

The Brute-Force algorithm works by checking every possible combination of changes. It schedules the combination with the least number of rounds, that pass the requirements outlined in chapter 3.1 [Alk24b].

```
procedure BruteForce($old\_route$, $new\_route$)
    required $\leftarrow$ changes between old route and new route
    $g \leftarrow$ graph from $old\_route$
    Use rounds from BruteForceRound(required, $g$)
end procedure

procedure BruteForceRound($required$, $network$)
    if $|\text{required}| = 0$ then
        return []
    end if
    $b \leftarrow$ None
    for $i \leftarrow |required|$ to $0$ do
        for each $c \in$ Combinations of $required$ with $i$ elements per combination do
            $v \leftarrow 1$                                  ▷ Stores if route change is still valid
            $r \leftarrow$ copy of $required$
            $current\_network \leftarrow$ copy of $network$
            $positive\_network \leftarrow$ copy of $network$
            $changes \leftarrow \{\}$
            for each $o \in c$ do
                if $o^+$ causes loop in $positive\_network$ or $o$ causes connectivity loss in $network$ then
                    $v \leftarrow 0$
                end if
                Apply $o$ in $current\_network$ and add it to $changes$
                Apply $o^+$ to $positive\_network$
                Remove $o$ from $r$
            end for
            if $v = 0$ then
                continue
            end if
            solution $\leftarrow$ BruteforceRound($r$, current_network)
            if solution has less rounds than best or $b$ is None then
                $b \leftarrow changes \cup$ solution
            end if
        end for
    end for
end procedure
```

As this is NP-complete [För+18], we use a backtracking-based approach to find the solution, as such algorithms are suited well for NP-completed problems [DMR21].

3.4. Chronicle

Zheng et al. [Zhe+18] introduced *Chronicle* as an algorithm for scheduling multi-stream route updates in time-based SDNs. When trying to find a suitable schedule, it takes the congestion of paths into account.

Chronicle expects a network graph without loops [Zhe+18]. This is in contrast to the other algorithms, which specialize in calculating updates without causing loops, even if the network topology would allow for them.

We implemented an algorithm inspired by Chronicle that does not expect timing information and outputs a round-based schedule. Our implementation therefore roughly follows the pseudocode by Zheng et al. [Zhe+19].

3.5. One-Round

Similar to the One-Round algorithm by Alkhatib [Alk24b], we implemented an algorithm that schedules all changes in the first round. This is a flow swap on all routers, as described in chapter 2.4.

The algorithm generates the same solution as our Chronicle variant for route changes that are not influenced by congestion.

```
procedure OneRound($old\_route$, $new\_route$)
    for 0 to $i \leftarrow |old\_route| - 2$ do
        Remove route from $old\_route[i]$ to $old\_route[i+1]$ in round 1
    end for
    for 0 to $i \leftarrow |new\_route| - 2$ do
        Add route from $new\_route[i]$ to $new\_route[i+1]$ in round 1
    end for
end procedure
```

In time-based SDNs without congestion, this algorithm would therefore produce an optimal solution, as it is ensured that all routers apply the new routes simultaneously.

4. Research Questions

We formulated two main research questions for this thesis:

How can algorithms for loop-free route updates in Software Defined Networks be applied in multi-stream networks?
There is a lot of literature that analyzes route updates and their effects for single-flow SDNs [Alk24b] [Alk24a] [För+18] [Zhe+17] [LMS15], with only some papers considering multiple flows [MW13] [Zhe+19] [XLH]. While the update strategies in the literature mainly focus on single-flow networks, we can use the same update strategies in multi-stream networks if the forwarding rules are per-destination [MW13]. We want to explore this, to evaluate if it is a suitable approach in realistic environments.

How much packet loss does adapting time-based algorithms cause for non-time-based Software Defined Networks?
As only some networks support time-based updates, we want to find suitable alternative options to use algorithms designed for time-based Software Defined Networks. To achieve this, we want to send a command to apply pending changes to all routers at the same time to imitate how time-based updates work. As this will come with more packet loss than loop-free route updates, we need to measure the packet loss produced and compare it with loop-free route updates.

In addition to these research questions, we developed an algorithm that sits on a middle ground between flow swapping and round-based algorithms. We developed Three-Round and introduce it in chapter 5.

5. Three-Round

While One-Round should work well in time-based SDNs, it will cause packet loss in our network, as we cannot guarantee that all changes are applied simultaneously. We developed Three-Round to reduce this packet loss.

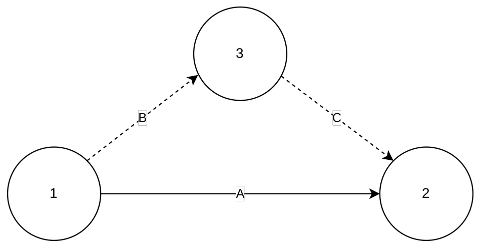


Figure 5.1.: A network with three nodes symbolizing routers, which has ideal conditions to use Three-Round

Consider the route change in figure 5.1, where we want to update the path the flow from router 1 to router 2 takes. While currently edge A is used, we want a new flow to take edge B and C. Updating router 1 and router 3 in the same round could lead to race conditions, where edge A is removed and B is set, but C is not set yet. A race condition like this can happen, as we do not know in which order changes in a round are executed. This would cause packet loss, as router 3 does not know where to route the traffic.
This can be prevented by updating router 3 in round one, as it does not have any effects on the flow. We can then update 1 in the next round. This strategy is always possible, when there are nodes that are not currently part of the flow. We incorporated this fact into our custom algorithm, which therefore always uses three rounds:

1. Determine routers that currently do not route traffic. Apply the changes for these routers.
2. Apply changes for all other routers.
3. Clean-up routes still left from the old route.

The third round removes any unused routes that get no traffic after round two.

```
1: procedure ThreeRound($old\_route$, $new\_route$)
2:     for 0 to $i \leftarrow |new\_route| - 2$ do
3:         if $new\_route[i]$ is in $old\_route$ then
4:             if route from $new\_route[i]$ is in $old\_route$ then
5:                 continue
6:             end if
7:             Remove route from $new\_route[i]$ in round 2
8:             Add route from $new\_route[i]$ to $new\_route[i+1]$ in round 2
```

```
 9:         else
10:             Add route **from** $new\_route[i]$ **to** $new\_route[i+1]$ in round 1
11:         end if
12:     end for
13:     for 0 to $i \leftarrow |old\_route| - 2$ do
14:         if route from $old\_route[i]$ is not in $new\_route$ then
15:             Remove route from $old\_route[i]$ to $old\_route[i+1]$ in round 3
16:         end if
17:     end for
18: end procedure
```

While this algorithm is not motivated by any existing literature, the basic concept can be compared to that of relaxed loop-freedom, where routers that do not *currently* get *new* traffic are updated without verifying that it does not cause loops [LMS15]. The key difference is that while algorithms conforming to relaxed loop-freedom generate loop-free schedules for traffic newly entering, that is not guaranteed for Three-Round, as *all* nodes currently getting traffic are updated in round two. Three-Round is also simpler by updating all routers that do not get any traffic in the first round. These differences allow Three-Round to always produce schedules with two rounds plus clean-up.

6. Experiment Setup

To evaluate the algorithms, we construct a network of two five-port computers and eight MikroTik hEX routers running RouterOS. An additional computer is used as a controller. We chose a static network topology without changes between test runs to provide consistency between tests and to avoid cabling mistakes.

In chapter 6.1, we will describe the setup of our test network. Chapter 6.2 will introduce the hardware used to facilitate the tests. The software components we wrote in order to calculate and execute route changes, run the tests, and export graphs will be introduced in chapter 6.3. During the setup and testing of our setup, we ran into some problems, which will be explained in chapter 6.4. In order to allow for further research, we created our software components to more easily reproduce our results. We will explain the steps necessary in chapter 6.5.

6.1. Network-Setup

The routers and computers are connected as shown in figure 6.1. The three computers act as a source, destination, and controller for our tests and will therefore be referred to as such.

We have assigned four source addresses (fd8a::1 to fd8a::4) to the loopback interface of our source and four destination addresses (fd8b::1 to fd8b::4) to the loopback interface of our destination. We designate each address to a link to a router. This way we can control exactly which links traffic takes from and to the computers.

The source and destination computers are connected via a direct cable that is always used as a (back-)route back to the source. This allows us to attribute changes to the forward-route, without having to worry about changes to the back-route.

To ensure precise time measurements in UDP latency tests, we have configured the source and destination computers to sync their times using PTP. This allows us to get accuracies as low as tens of microseconds [Che22].

Additionally, we connected all computers and routers to a switch to create an out-of-band management network. Out-of-band management uses a separate dedicated network to allow the controller to talk to the routers [Jal+17]. This network is never used to carry test-traffic. This is ensured by using the legacy protocol IPv4 in that network, while we use IPv6 in our test network. The network is used to allow the controller to talk to the source, destination, and all routers.

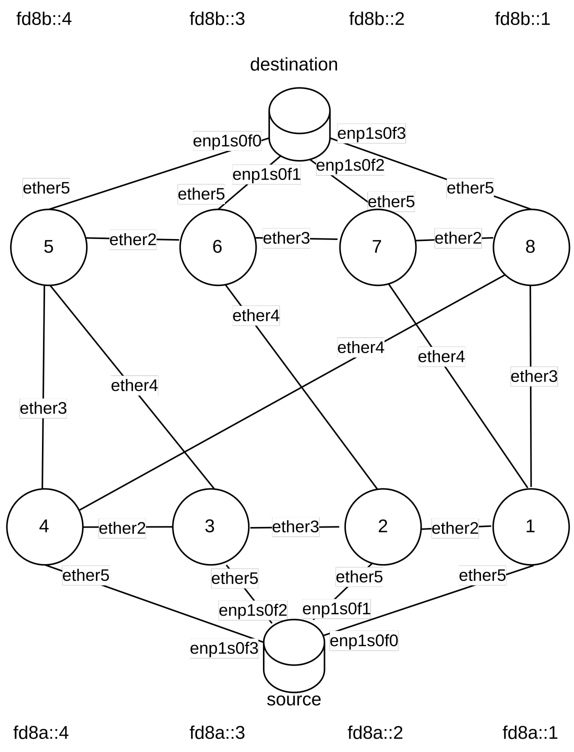


Figure 6.1.: Visualization of the network of the test setup. On the bottom and on the top is the source and destination computer. Circles are routers. Lines are links between the routers. The out-of-band management network that is connected to all routers and computers is not shown.

6.2. Hardware

Controller, source, and destination are DELL D17S computers with Debian 12. These computers are powerful enough to handle all tasks in this thesis with moderate CPU usage. Our setup is shown in figure 6.2.

All routers are MikroTik hEX routers with RouterOS 7.18.2. The routers are not powerful enough to route gigabit streams, even with *fasttrack6* enabled and do not support Jumbo frames. We therefore decided to limit our testing bandwidth to 50 Mbit/s per flow in order to have a static throughput that is not influenced by the routers' performance.

6.3. Software

In order to allow for reproduction and to provide consistent results, we created a system containing the following elements: Ansible-Setup-Playbook, Route-Change-Calculator, Route-Applier, and Test-Controller.

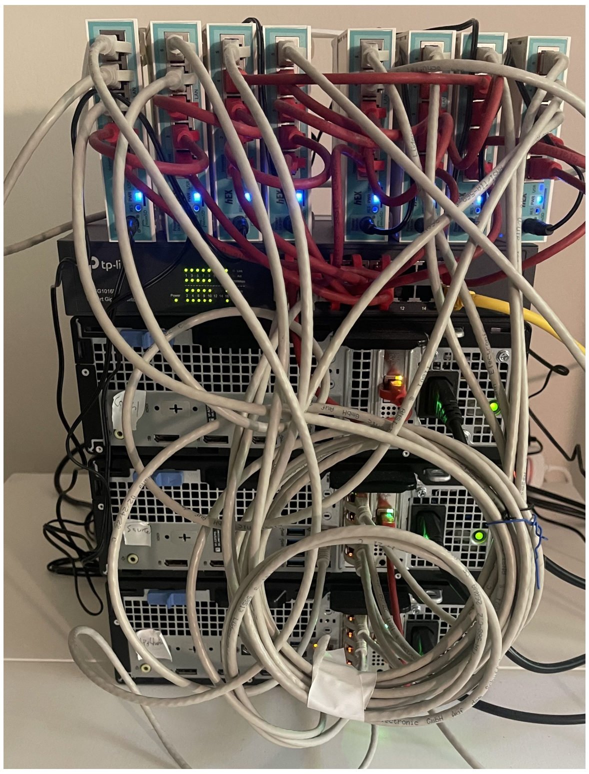


Figure 6.2.: Image of three computers, one switch and 8 routers.

These components roughly follow the Unix philosophy "Do one thing and do it well, write programs that work together" [Sal94] by working as independent software components. The interactions between the components are depicted in figure 6.3. This enables us to look at each component individually and allows further work to re-use some components. In this section, we will explain what each component does and how they work together.

6.3.1. Ansible-Setup-Playbook

We decided to use Ansible for the basic setup of the test computer and routers. This is done to let others reproduce our findings by being able to apply our configuration on their hardware.

In order to have a consistent setup on all routers, we apply the "router-setup" playbook[1] to all routers. This playbook ensures that there are no unnecessary default configurations on the routers. Afterwards, it configures upstream over IPv4 to the internet over the management network, to allow for updates and remote management of the router. It then applies firewall rules. Contrary to the router's default configuration, we do not configure a rule to drop invalid traffic for IPv6, as this rule

[1]https://gitlab.fachschaften.org/ba-loop-free-routing-non-time-based/ansible/-/blob/main/router_setup.yml

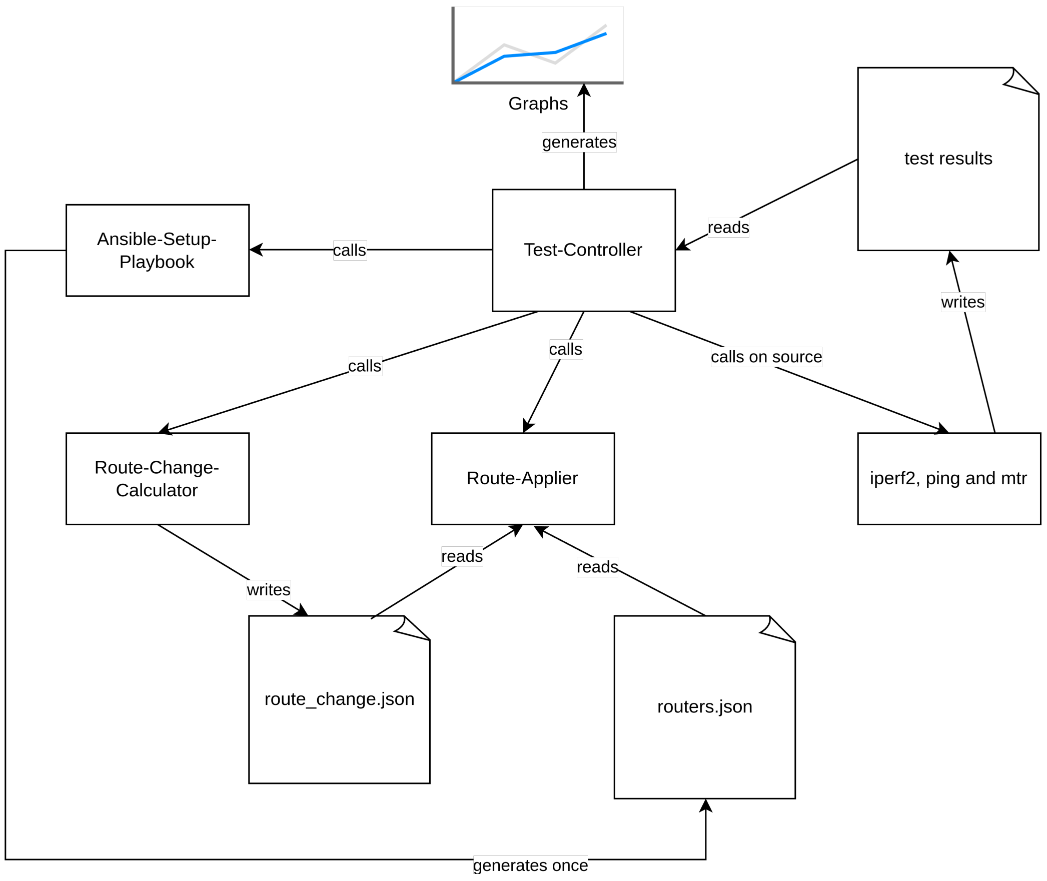


Figure 6.3.: Drawing of the architecture of the test setup

caused unexplainable packet loss during testing. Lastly, we configure our test network: We set static IPv6 addresses on every link between two routers. We also set a few routes:

- Routes back to our sources, to allow for ICMP packets to reach them, e.g. when the TTL expires mid-path. This is especially useful for our “traceroute” test.
- Routes from router 5-8 to the destination address assigned to the loopback interface of the destination computer, if it is designated for that link.

The ansible playbook “debian”[2] configures the testing computers. It installs various software we need for tests, configures an SSH daemon using the ansible-role “fsi-ansible.sshd”[3] and copies the network config from our repository. It configures PTP for time synchronization between the source and the destination computer. The playbook also configures the destination computer to automatically start an iperf2 server for TCP tests and installs all software components on the controller computer.

[2]https://gitlab.fachschaften.org/ba-loop-free-routing-non-time-based/ansible/-/blob/main/debian.yml
[3]https://gitlab.fachschaften.org/fsi-ansible/sshd

A third central playbook is "clear_router.yml"[4], which resets the routes of all routers to their default state outlined above and removes pending scripts, if there are any. This playbook is used by our Test-Controller component to clean up between test runs.

6.3.2. Route-Change-Calculator

The Route-Change-Calculator[5] is an interactive Python3 application, which takes new routes as input and calculates a "route_change.json" containing the necessary steps to change to these routes from our default routes for a given algorithm.

All algorithms outlined in chapter 3 are implemented in Route-Change-Calculator and can therefore be tested. For this to work, Route-Change-Calculator asks for the new routes, the algorithm of choice, and the waiting time between rounds when executed. The waiting time between rounds is only written to the result file for later use by Route-Applier. The algorithm of choice is passed the new and default routes and calculates a route-change-rounds file. This file contains the information about added and removed routes for each round. At the beginning of the file, default routes for all flows are added to always start tests with a fixed baseline. The below listing shows an example of such a route-change file.

Listing 6.1: route-change.json example

```
{
  "rounds": [
    {
      "removeRoutes": [
        {
          "from": "r1",
          "to": "r2",
          "flow": "1"
        }
      ],
      "addRoutes": [
        {
          "from": "r2",
          "to": "r7",
          "flow": "1"
        }
      ],
      "propagationTime": 5000
    }
  ]
```

[4]https://gitlab.fachschaften.org/ba-loop-free-routing-non-time-based/ansible/-/blob/main/clear_router.yml

[5]https://gitlab.fachschaften.org/ba-loop-free-routing-non-time-based/route-change-calculator

```
}
```

6.3.3. Route-Applier

After the Route-Change-Calculator has calculated a route-change file, it can be passed, in combination with a `routers.json` to the Route-Applier component. The `routers.json` is used as a configuration file holding the API access data and metadata about the connection between routers. It is generated by the Ansible-Setup-Playbook and stored on the controller.

The Route-Applier component consists of a small Go program that connects to the MikroTik API using community maintained bindings[6] and executes the RouterOS commands necessary for route changes. When wanting to apply route changes, minimizing the time it takes to apply the changes is often a goal [FSV19]. Go is chosen as it can be compiled into native binaries and has good support for asynchronous operations.

The option `propagationTime` in the route-change file can be used to wait between applying rounds. We use it in chapter 7 in order to see which change caused which effects by looking at the time the changes happened at.

The application of the routes uses four steps per round:

1. RouterOS script files containing the necessary commands for the route change are generated. All routes are added with a distance of 1 and later degraded to a distance of 10, to have a higher priority than routes that are getting deleted. A resulting script file could look like this:

   ```
   /ipv6/route/add dst-address=fd8b::4/128 gateway=fd80:0:0:1::1 distance=1
   /ipv6/route/set number=[find distance=10 dst-address=fd8b::2/128 gateway=fd80:0:0:5::3] gateway=fd80:0:0:6::6
   /ipv6/route/set number=[find distance=10 dst-address=fd8b::4/128 gateway=fd80:0:0:1::1] gateway=fd80:0:0:5::3
   /ipv6/route/set distance=10 numbers=[find dst-address=fd8b::4/128 gateway=fd80:0:0:1::1 distance=1]
   ```

2. The RouterOS scripts are transferred to the routers.
3. The RouterOS script files are executed on the routers.
4. The RouterOS script files are deleted from the routers.

All steps are executed simultaneously on all routers using Go subroutines. This is required for good results for our Chronicle, One-Round and Three-Round implementations.

We first send a script file with the required commands to the router. After all routers have received the script file, we apply them on all routers simultaneously. We do this in order to better mimic time-based SDNs, by first writing all route changes in a file and later applying them. As MikroTik has no built-in option to write changes to a buffer and later apply them, we decided to use script files.

[6]https://github.com/go-routeros/routeros

6.3.4. Test-Controller

The test controller is split into two parts: `create_plots` and test execution script files. The script files are used to start Route-Change-Calculator, Route-Applier and the testing tools iperf2, mtr and ping. We run the testing tools in sequence to avoid them influencing each other. `create_plots` is a small Python application that parses the output files of our tests and creates plots using the Python library *plotly*. On every run, all files are parsed. Afterwards, this data is used to generate multiple different graphs.

6.4. Problems

While performing our evaluation, we found multiple problems that we had to work around.

6.4.1. Routers Not Routing

In around one of 100 tests, the routers suddenly stopped routing. As far as we can tell, sometimes RouterOS does not accept valid routes. There is no indication that the route is not installed correctly. Disabling fast-track does not help. Disabling the non-working routes and enabling them without changes again seems to fix the problem temporarily.

We have reported this issue to the manufacturer. We can reproduce this bug with RouterOS 7.18.2, 7.19.1 and 7.20.0-beta2. We removed all measurements affected by this issue before plotting, as we are convinced that this is a RouterOS bug.

6.4.2. Packet loss during route updating

When updating the routes of a RouterOS router, there is slight packet loss. This issue can be reproduced by creating a script that changes routes back and forth and executing it while measuring for packet loss.

In our tests, this packet loss seems to be smaller when changing the destination of existing static routes, instead of creating new static routes and deleting the old ones. We therefore change routes whenever possible. We have reported this issue to the manufacturer as well. MikroTik told us, that this packet loss is to be expected during the change of a route in RouterOS.

6.4.3. iPerf3 not measuring latency

Similarly to Alkhatib [Alk24b], we could not use iPerf3 for our measurements. While iPerf2 allows us to measure round-trip time for TCP connections and latency for UDP connections, these measurements are not available in iPerf3. We therefore used the older iPerf2 for both our TCP and UDP tests.

6.5. Reproducibility, Configurability and Adaptability

The setup used in this thesis is built in a way that enables further research by being able to modify components on their own.

In order to reproduce the results of this thesis on the same hardware, it is necessary to recreate the network setup and management network. Afterwards, the Ansible-Setup-Playbook needs to be cloned and changed to fit the setup. After connecting all routers, disabling the RouterOS firewall and setting up Debian 12 on the computers, the Ansible playbooks can be executed. The playbooks should set up basic router settings and install all necessary tools on the computers.

On the controller computer, SSH to source and destination and accept the SSH host keys. One can now create an empty directory on the controller and run `test_all`.

7. Experiment Results

We created multiple test cases and tested them with all algorithms to allow comparison between the algorithms. We start all tests with default routes for all flows. Each number represents the identifier of a router. Each line is a flow (flows one to four).

1. 1, 2, 3, 4, 5, 6, 7, 8
2. 2, 3, 4, 5, 6, 7
3. 3, 2, 1, 8, 7, 6
4. 4, 3, 2, 1, 8, 7, 6, 5

Figure 7.1 demonstrates the default routes graphically.

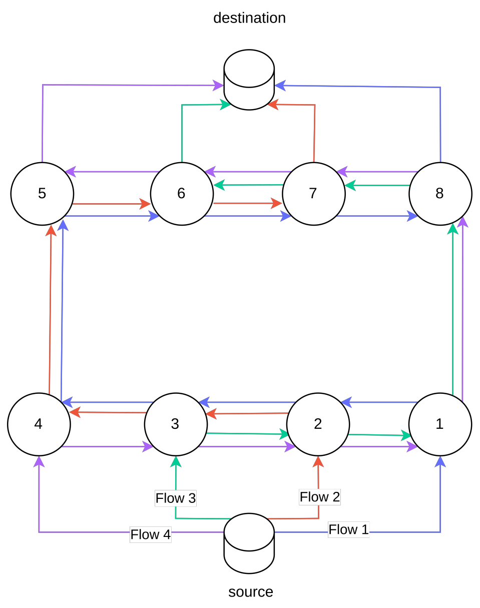


Figure 7.1.: Shown are four flows differentiated by color

We ran all test cases 100 times with a delay of 5 seconds between rounds. This allows us to differentiate between the rounds in the results. The first round begins after 5 seconds. All tests ran for 45 seconds, which gives us time for a maximum of seven rounds.

In order to compare the speed of our solution to the results of Alkhatib [Alk24b] and their Ansible-based solution, we ran route changes again with no waiting times between rounds and measured the time it took to apply the changes. We also ran the route change calculation for every route change 1000 times to compare the time it took to generate the route change.

7.1. Taking the short way

To get a feel for how the algorithms behave when shortening routes, we perform a route change to a route with one hop. This allows us to get a baseline for simple changes with a small amount of rounds and establish near-minimum round-trip-times.

Our selected routes are shown below. Each row represents a flow and each number a route. Figure 7.2 visualizes the changes.

1. 1, 2, 3, 4, 5, 6, 7, 8 to 1, 7, 8
2. 2, 3, 4, 5, 6, 7 to 2, 6, 7
3. 3, 2, 1, 8, 7, 6 to 3, 5, 6
4. 4, 3, 2, 1, 8, 7, 6, 5 to 4, 3, 5

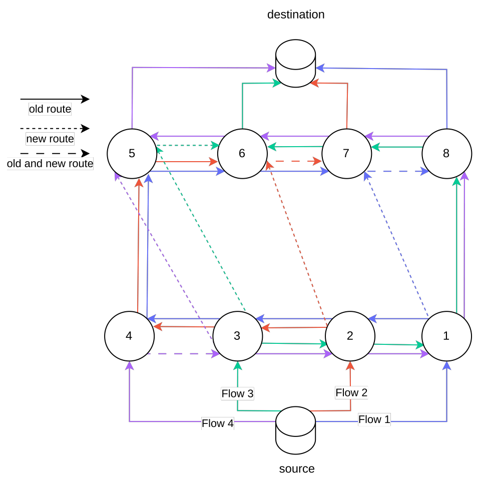


Figure 7.2.: Graphic of the first route change

We will present the results of this route change in chapter 7.1.1, followed by comparisons in chapter 7.1.2, 7.1.3 and 7.1.4. We will end the chapter by discussing the results in chapter 7.1.5.

7.1.1. Results from Algorithms

Backward

The Backward algorithm calculates the route change in two rounds. In round one, all flows are updated. Round two is only required for flow three.

This is further shown by the route update diagram in figure 7.3, when leaving 5 seconds between rounds.

The latency between source and destination is severely affected by the shortening of the routes. This can be seen in figure 7.4 where we can see a steep fall in the fifth

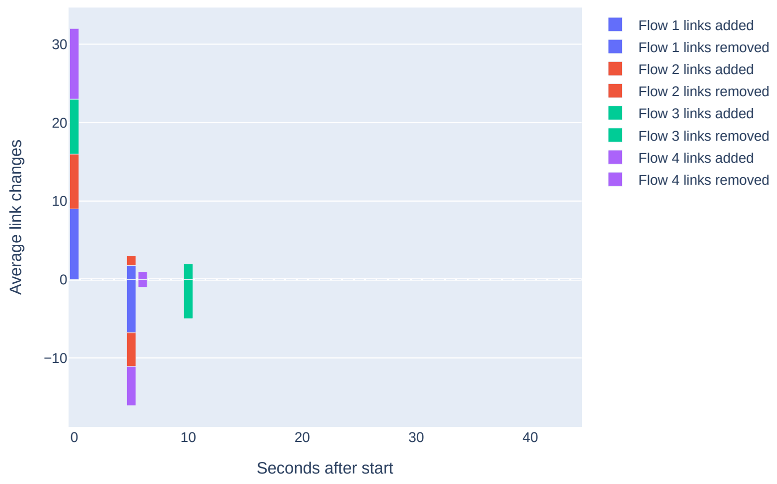
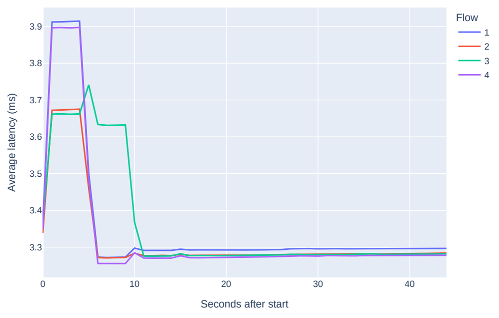


Figure 7.3.: Links disappearing and newly appearing in traceroutes when running the Backward algorithm

second for all flows. The latency of flow three falls again, as expected, at ten seconds (in round two). The round trip time in figure 7.5 behaves similarly, but is significantly smaller than the latency. This is implausible and indicative of either slower routing during the latency tests or an issue with the PTP time-synchronization during the measurements.



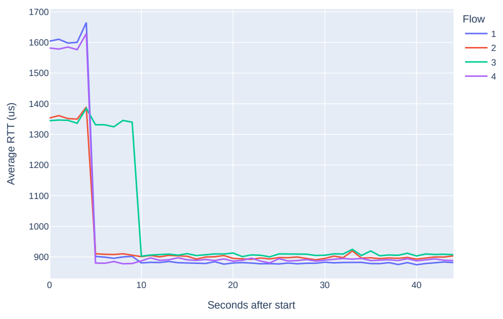


Figure 7.4.: Latency between source and destination of all flows during a route change with the Backward algorithm

Figure 7.5.: Round-Trip-Time between source and destination of all flows during the Backward algorithm

As the Backward algorithm is a round-based loop-free algorithm, we should experience no packet loss. When we look at the results of our ping-test in figure 7.6, we can see little packet loss at the five-second mark. This loss can be explained by route changes sometimes dropping packets in RouterOS, as we have explained in Chapter 6.4.2. This is also illustrated by the retries of the TCP stream in 7.8.

As we only test with 50 mbit/s per flow and there is enough network capacity to catch up, we cannot see any drop in bandwidth.

When executing the next round directly after the previous round finished, the time to apply the changes was between 318 ms and 1048 ms, with a median of 380 ms. As displayed in figure 7.7 most time is consumed by initializing the connection to the routers. Round 1 normally takes longer than round two. There were some cases where round two took longer.

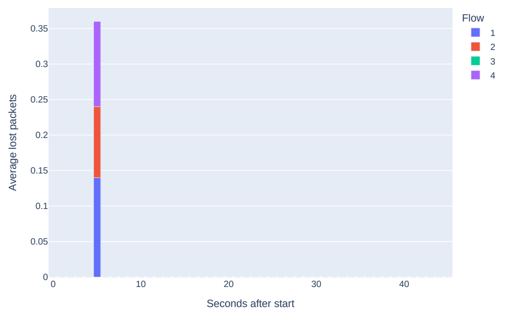


Figure 7.6.: Lost packets in our ping test when executing a route change with the Backward algorithm

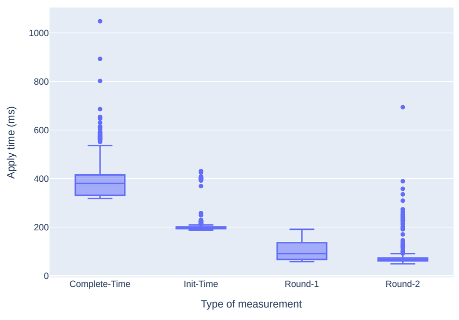


Figure 7.7.: Time that it takes to apply the changes, when not sleeping between rounds

Compared to waiting five seconds between rounds, the effects are more condensed. As can be seen in figure 7.9, there is no influence on the bandwidth.

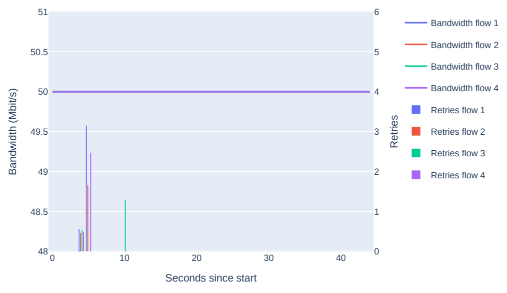


Figure 7.8.: Bandwidth and retries during the execution of the Backward algorithm

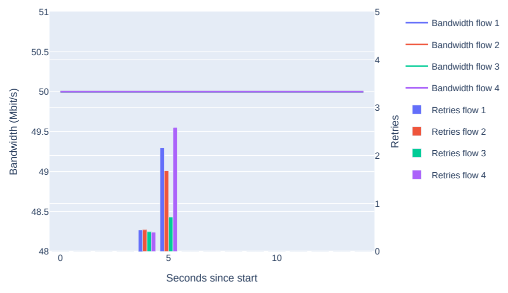


Figure 7.9.: Bandwidth and retries during the execution of the Backward algorithm, when not sleeping between rounds

Greedy

For this route change, the solution of the Greedy algorithm does not differ from the solution of the Backward algorithm.

This causes the results to not differ much from the results of the Backward algorithm. All results can be found in appendix A.

Brute-Force

The solution of Brute-Force is the same as the solution of the Backward algorithm for this route-change, as well.

The results do therefore not differ significantly from the results of the Backward and Greedy algorithms. All results can be found in appendix A.

Chronicle

The Chronicle algorithm applies all route changes in the first round, as there is no congestion.

Applying all changes at once, as is shown in figure 7.10, has a direct influence on the latency in the fifth second. However, it results in a significant higher loss in our ping and bandwidth tests, as is shown in figure 7.11 and figure 7.12. This is expected, as we do not have a Timed-SDN network, but can only try to apply all changes roughly simultaneously.

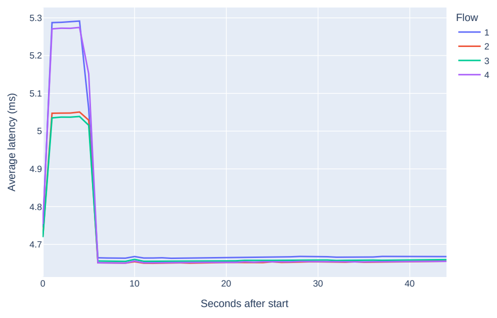


Figure 7.10.: Latency between source and destination of all flows during a route change with the Chronicle algorithm

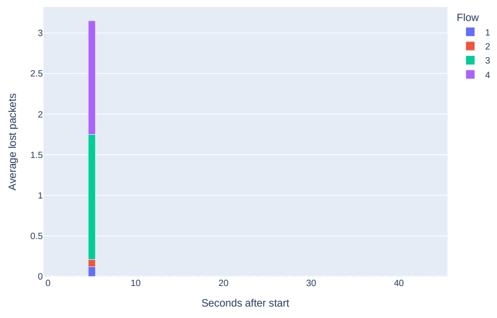


Figure 7.11.: Lost packets in our ping test when executing a route change with the Chronicle algorithm

Figure 7.13 shows the time it took to apply all changes, which is between 340 ms and 859 ms with a median of 408ms.

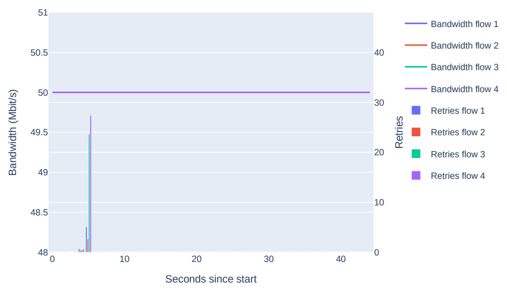


Figure 7.12.: Bandwidth and retries during the execution of the Chronicle algorithm

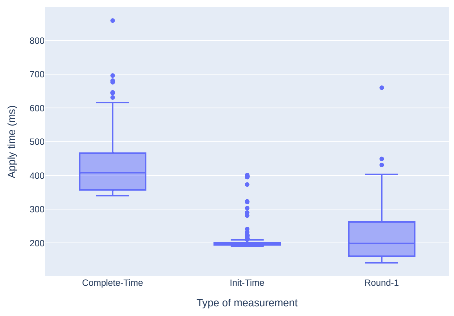


Figure 7.13.: Time that it takes to apply the changes using the Chronicle algorithm

One-Round

The Chronicle schedule and the One-Round schedule are identical.

All results can be found in appendix A.

Three-Round

Our Three-Round algorithm adds a route from router 5 to 6 in the first round. The second round adds all other necessary rounds. Round three removes old, unused routes.

The latency plot in figure 7.14 illustrates that all visible changes are happening in round two.

There is slight packet loss, when round two is applied, as shown in figure 7.15. This packet loss is comparable to the packet loss when using loop-free updates.

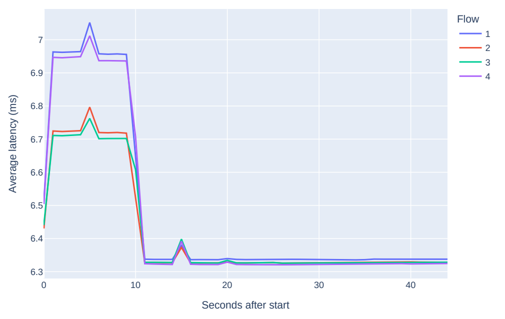


Figure 7.14.: Latency between source and destination of all flows during a route change with the Three-Round algorithm

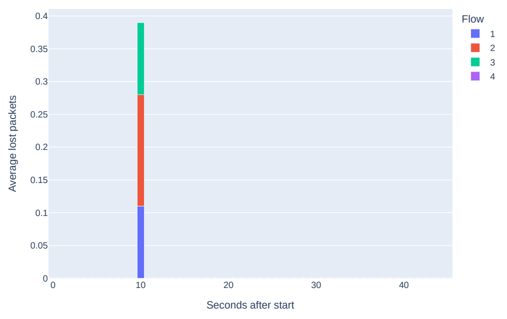


Figure 7.15.: Lost packets in our ping test when executing a route change with the Three-Round algorithm

Without extra time between rounds, the time it takes to apply all rounds is between 470 ms and 1253 ms. Round three, which only cleans up unused routes, is between 107 ms and 706 ms. Detailed information about the apply-time can be found in figure 7.16.

The packet loss does not differ much when running without time between the rounds, as can be seen in figure 7.17.

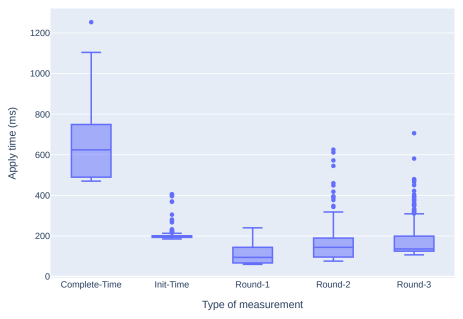


Figure 7.16.: Time that it takes to apply the changes using the Three-Round algorithm, if we leave no time between rounds

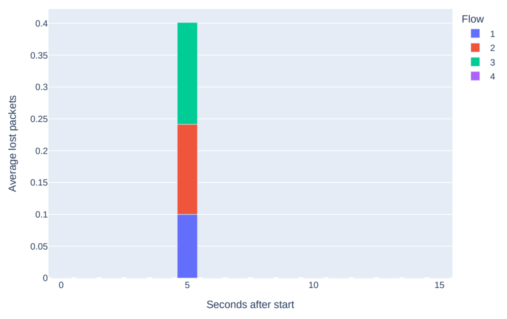


Figure 7.17.: Lost packets in our ping test when executing a route change with the Three-Round algorithm and no extra time between rounds

7.1.2. Comparison of Algorithms

Greedy, Backward, and Brute-force all create good results with an equal number of two rounds.

Chronicle and One-Round only need one round, but experience significantly higher packet loss. While Backward, Brute-force, Greedy and Three-Round have, at most, one lost packet in our ICMP-Echo tests, Chronicle and One-Round have up to seven. Retries and Lost ICMP-Echos are around ten times higher for these algorithms for some flows. This translates to around 1 ms of non-connection for flow three and four. Flow one and two are not affected by this.

Our test flows in figure 7.18 demonstrates that Chronicle and One-Round generate more packet loss than the other algorithms for this test-case.

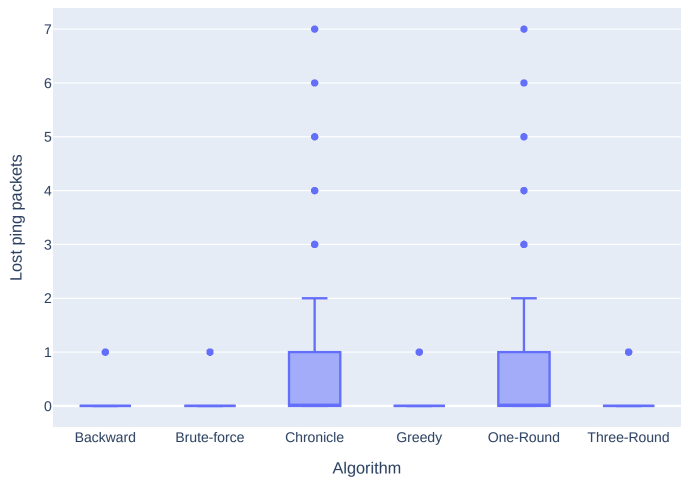


Figure 7.18.: Comparison of lost packets for the different algorithms

Three-Round needs the most rounds for this change. If we do not count the clean-up round, Three-Round still uses as many rounds as Greedy, Backward and Brute-force.

7.1.3. Performance Comparison

Figure 7.19 compares the duration it takes to apply the changes of the algorithms proposed. Interestingly, Brute-force, Greedy, and Backward are typically 22 ms faster than Chronicle and One-Round, even though they have one round more. Three-Round is typically 219 ms slower than Chronicle and One-Round, as it always needs three rounds and takes time cleaning up.

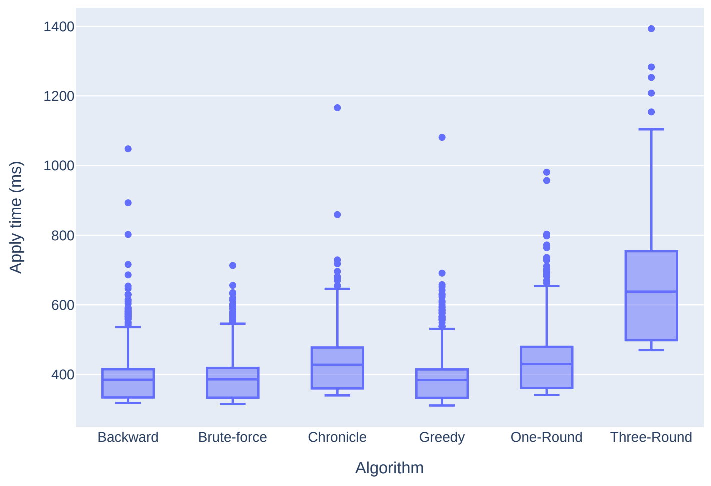


Figure 7.19.: Comparison of the time it took to apply the routes for the different algorithms

7.1.4. Comparison of Calculation Time

In Chapter 7.1.3 we look at the time it takes to apply the changes and omit the time it takes to generate valid changes. As can be seen in figure 7.20, this time varies between the algorithms with Brute-force and Greedy both using between 0.16 seconds and 0.18 seconds. One-Round, Backward, Chronicle and Three-Round use less than one millisecond to generate this route change.

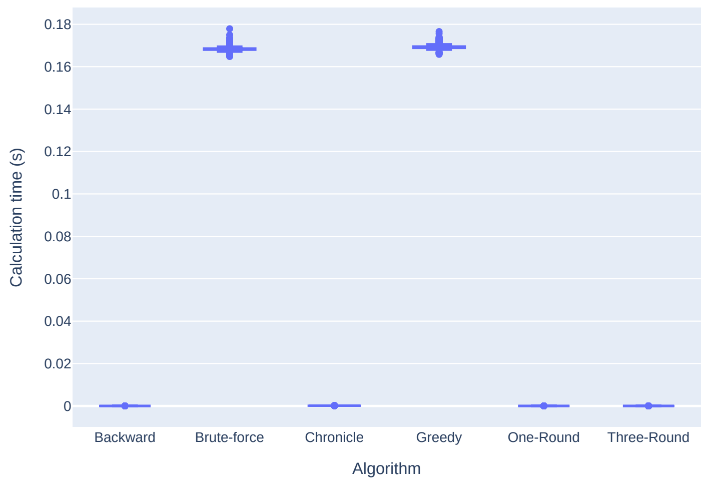


Figure 7.20.: Comparison of the time it took to calculate the routes for the different algorithms

7.1.5. Discussion Of Route Change 1

The best algorithms for this route change are Backward, Brute-force and Greedy. While they have one more round than Chronicle and One-Round, they have up to ten times less packet loss and apply 20 ms faster in our tests.

Three-Round has comparable packet loss to Backward, Brute-force, and Greedy, but takes 219 ms longer to apply. As Backward has the shortest calculation time with less than one millisecond, we recommend Backward for this route change.

7.2. Looping around

As we compare multiple loop-free route-update algorithms, we want to especially test their ability for creating loop-free route-updates. To evaluate the ability of the algorithm to cope with loops, we test the following route changes that are prone to producing a lot of loops. Each row represents a flow and each number a route. Figure 7.21 visualizes the changes.

1. 1, 2, 3, 4, 5, 6, 7, 8 to 1, 7, 6, 2, 3, 5, 4, 8
2. 2, 3, 4, 5, 6, 7 to 2, 6, 5, 3, 4, 8, 1, 7
3. 3, 2, 1, 8, 7, 6 to 3, 5, 4, 8, 1, 7, 6
4. 4, 3, 2, 1, 8, 7, 6, 5 to 4, 8, 7, 1, 2, 3, 5

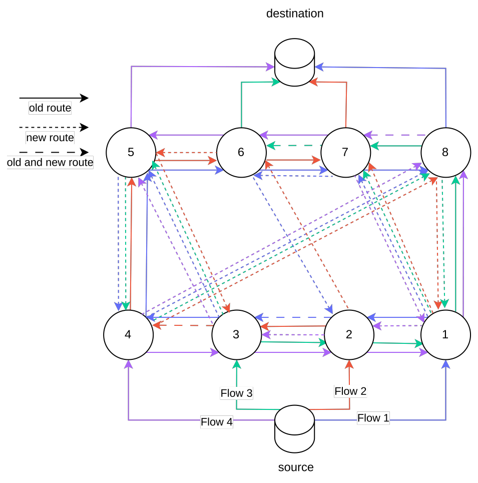


Figure 7.21.: Graphic of the second route change

We will present the results of this route change in chapter 7.2.1, followed by comparisons in chapter 7.2.2, 7.2.3 and 7.2.4. We will end the chapter by discussing the results in chapter 7.2.5.

7.2.1. Results from Algorithms

Backward

The Backward algorithm calculates a route change with six rounds.

As can be seen in the route change diagram in figure 7.22, not every round affects the current flow.

When looking at the round trip time diagram in figure 7.23, we can see an early fall in the round trip time of most flows. This is due to the temporary shortening of the routes, as can be seen in figure 7.22. When the new routes are fully installed, the round trip time rises again, as they are similar in length.

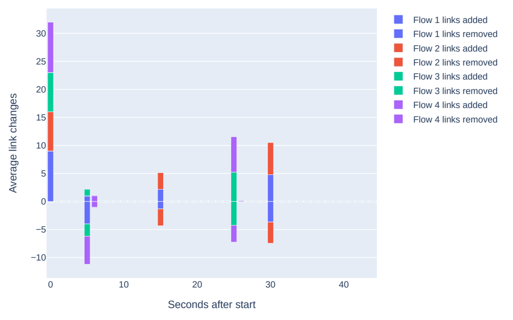


Figure 7.22.: Links disappearing and newly appearing in traceroutes when running the Backward algorithm

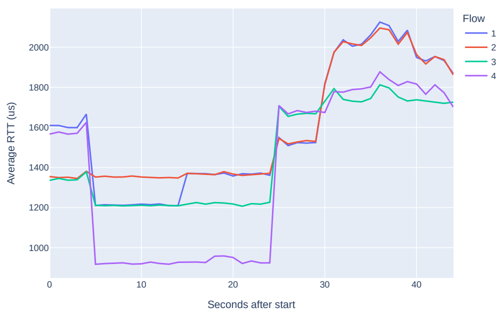
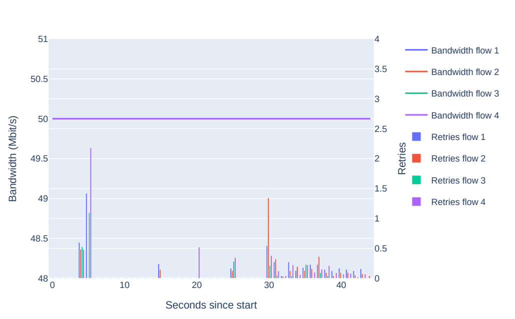


Figure 7.23.: Round-Trip-Time between source and destination of all flows during the route changes of the Backward algorithm

Looking at the retries in figure 7.24 and the packet loss in figure 7.25, we can see only slight packet loss and retries during route changes. We think that these occur due to a bug in RouterOS, which loses some packets during route changes, we have described in chapter 6.4.2. Comparing the diagrams shows some retries starting at around 30 seconds into the test. The need for these retries cannot be caused by our route change, as it is already over at that point. We theorize that this is caused by one of the routers not being able to handle the test traffic without dropping packets. When we look at the last round, this seems to be either router six or router seven. While router seven has to route four flows at this point, there are other routers routing four flows as well.



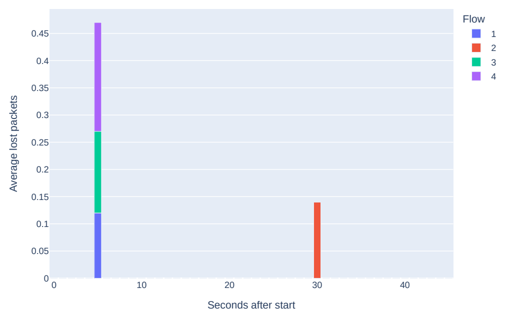


Figure 7.24.: Bandwidth and retries during the execution of the Backward algorithm

Figure 7.25.: Lost packets in our ping test when executing the route change with the backward algorithm

When looking at figure 7.26, we can see that, when we leave no extra time between rounds, the whole process took between 696 ms and 1933 ms, with a median of 1004 ms. All rounds took similarly long.

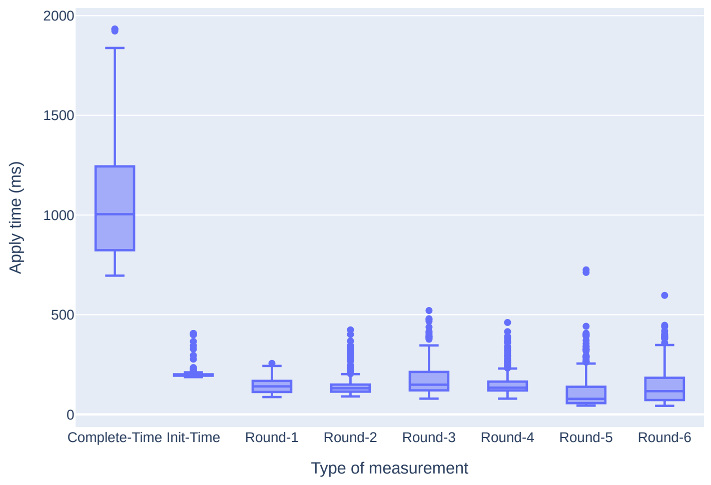


Figure 7.26.: Time that it takes to apply the changes using the Backward algorithm, if we leave no time between rounds

Brute-force

The Brute-force algorithm generates a solution with only four rounds.

Most visible changes happen in round one, as can be seen in the link change diagram in figure 7.27.

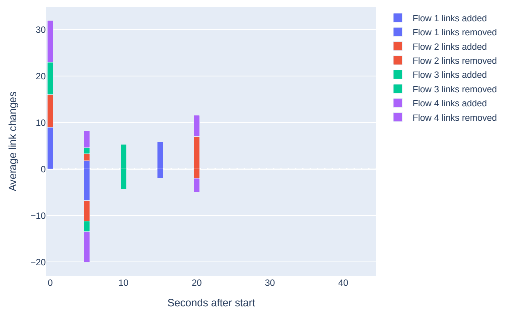


Figure 7.27.: Links disappearing and newly appearing in traceroutes when running the Brute-Force algorithm

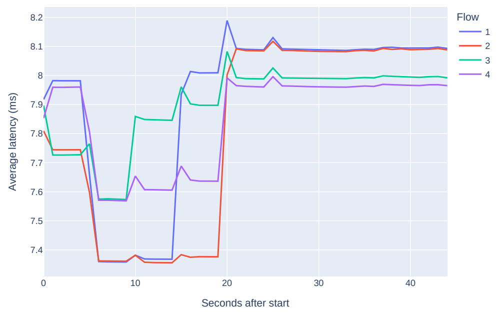


Figure 7.28.: Latency of all flows when running the Brute-force algorithm

Especially interesting is that the first round mostly removes links, while the other rounds add them back again. This can also be seen in the latency plot in figure 7.28.

Due to most changes happening in round one, we can measure the most retries and packet loss in that round. While we can measure up to four retries, the bandwidth stays at 50 Mbit/s. We can also see some retries in figure 7.29 after the end of the route change. This is the same behavior as with the Backward algorithm and suggests that the issue is with the route and not the route change.

Applying the route changes takes between 508ms and 1450ms, with a median of 732ms. As is shown in figure 7.30, all rounds take a similar amount of time, with round one taking the longest.

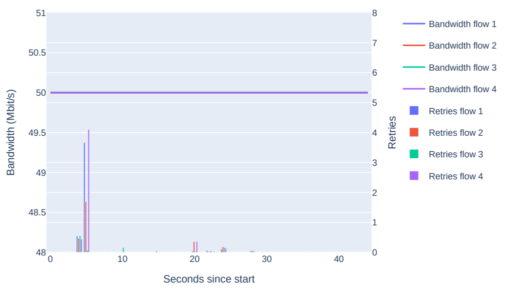


Figure 7.29.: Bandwidth and retries during the execution of the Brute-force algorithm

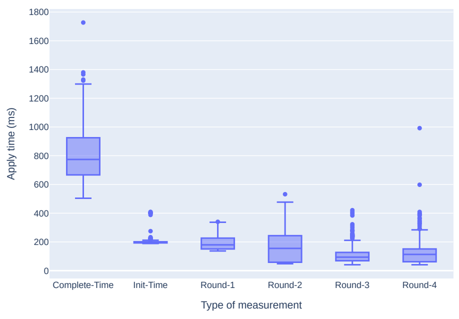


Figure 7.30.: Time that it takes to apply the changes using the Brute-force algorithm, if we leave no time between rounds

Greedy

The Greedy algorithm takes five rounds to complete the route change.

Some packet loss occurs in all rounds, with round one having the most, as can be seen in figure 7.31. This is likely due to most changes happening in that round, as can be seen in the link change diagram in figure 7.32. The bandwidth stays at 50 Mbit/s. One difference to the Greedy and Brute-force algorithm is that we do not see packet loss after the update finished, when sleeping between rounds. We can see the same packet loss when executing one round directly after the previous one, as can be seen in figure 7.33.

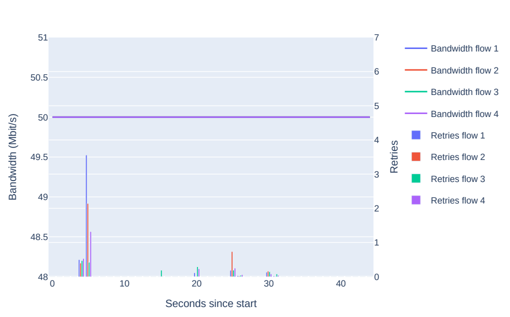


Figure 7.31.: Bandwidth and retries during the execution of the Greedy algorithm

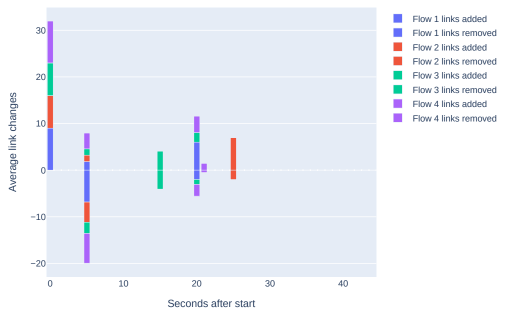


Figure 7.32.: Links disappearing and newly appearing in traceroutes when running the Greedy algorithm

The time it takes to apply the changes was between 593 ms and 1823 ms, with 906.5 ms being the median. As can be seen in figure 7.34, the time each round takes varies slightly with round 5 normally being the fastest one.

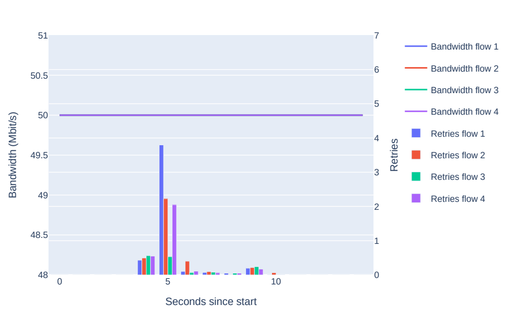


Figure 7.33.: Bandwidth and retries during the execution of the greedy algorithm, when no extra time between rounds

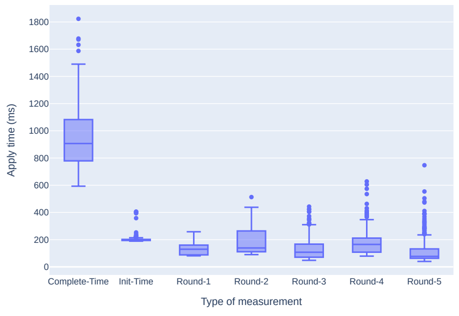


Figure 7.34.: Time that it takes to apply the changes using the greedy algorithm, if we leave no time between rounds

Chronicle

Chronicle schedules all changes in round one, as there is no congestion.

Doing all changes simultaneously in a non-time-based network comes with extra packet loss, as can be seen in figure 7.35. This can also be seen when looking at the retries in figure 7.36. Contrary to other algorithms, we can see a slight loss in bandwidth for flow three. The slight packet loss after the route change known from the Backward and Brute-force algorithms cannot be seen here.

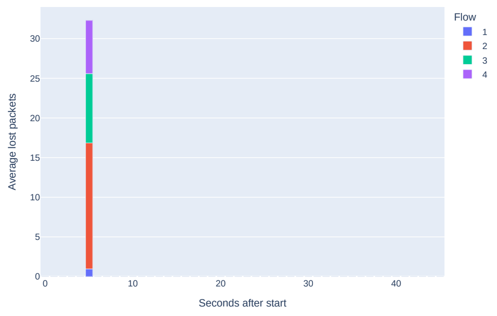


Figure 7.35.: Lost packets in our ping test when executing the route change with the Chronicle algorithm

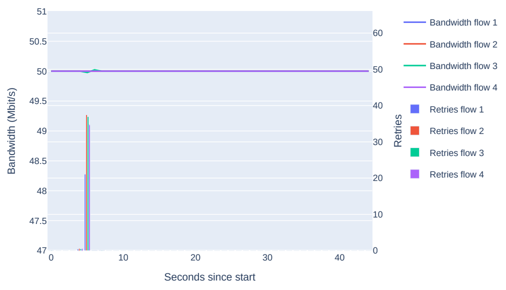


Figure 7.36.: Bandwidth and retries during the execution of the Chronicle algorithm

When looking at the round trip time in figure 7.37, we can see a short spike when the changes are applied.

Figure 7.38 shows that the whole process takes 356 ms to 782 ms with a median of 491ms.

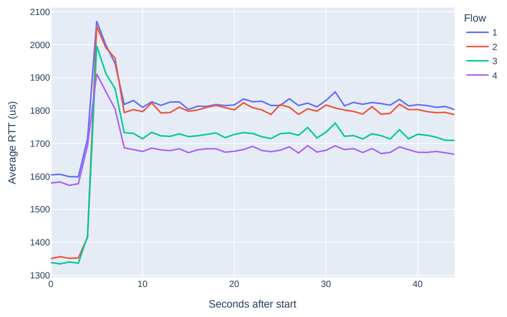


Figure 7.37.: Round-Trip-Time between source and destination of all flows during the route changes of the Chronicle algorithm

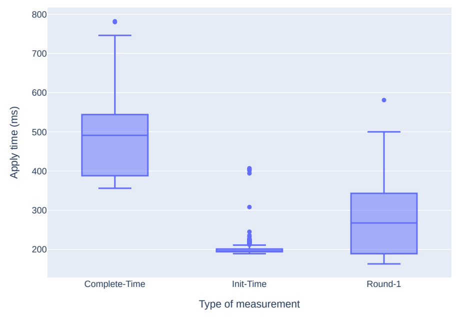


Figure 7.38.: Time that it takes to apply the changes using the Chronicle algorithm, if we leave no time between rounds

One-Round

The One-Round algorithm produces the same route change as the Chronicle algorithm for this case. The results are therefore very similar.

A slight difference, which can be attributed to chance, is the change of flows that lost bandwidth during the tests. Figure 7.39 shows that, contrary to Chronicle, all flows have a small drop in bandwidth.

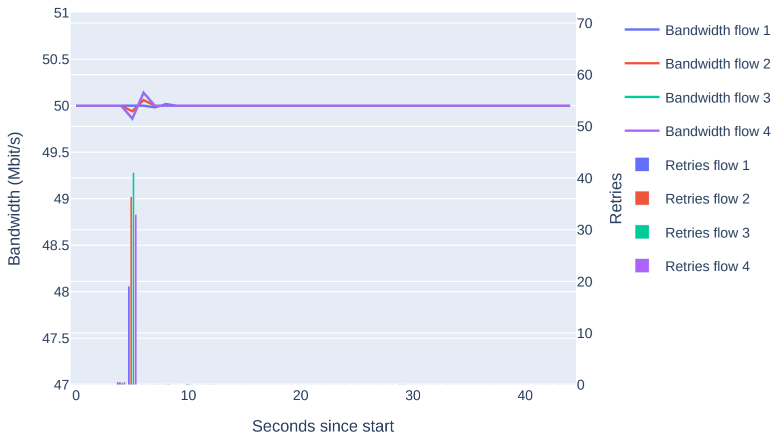


Figure 7.39.: Bandwidth and retries during the execution of the One-Round algorithm

All results can be found in appendix A.

Three-Round

The Three-Round algorithm calculates the route change in three rounds. All visible route changes are happening in round two.

When looking at the bandwidth in figure 7.40, we cannot see any bandwidth drop. The retries for all flows are quite high, with flow one, three, and four being similar to One-Round. Flow two has fewer retries than it has with One-Round and Chronicle.

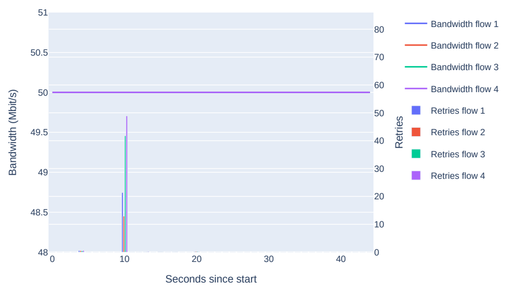


Figure 7.40.: Bandwidth and retries during the execution of the Three-Round algorithm

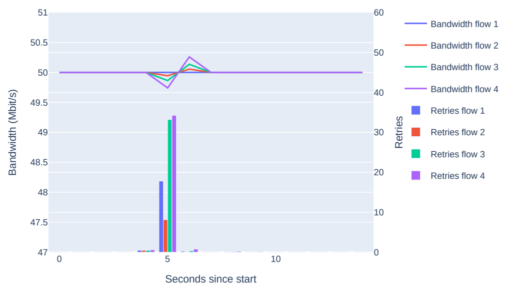


Figure 7.41.: Bandwidth and retries during the execution of the Three-Round algorithm, when no extra time between rounds

The reason that One-Round has not even less packet loss could be that in round one, only four routes (two in flow two and two in flow three) are set. This is because of the default route already covering almost all routers, and therefore almost no changes are possible without affecting the current flow.

When looking at the results from our tests without sleeping between rounds in figure 7.41, we can see that there is a small loss in bandwidth for all flows. The retries do not differ much to sleeping five seconds between rounds.

As shown in figure 7.42, the route change was executed in 449 ms to 1458 ms with a median of 706 ms.

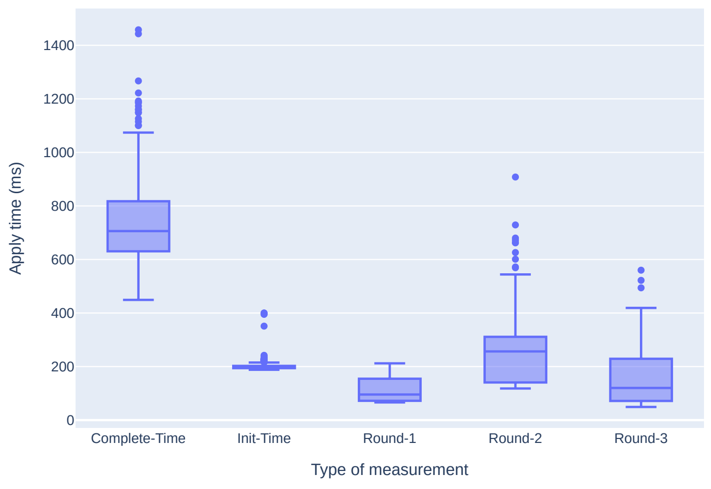


Figure 7.42.: Time that it takes to apply the changes using the Three-Round algorithm, if we leave no time between rounds

7.2.2. Comparison of Algorithms

The number of rounds of loop-free algorithms is between four and six. Brute-force only needs four rounds, the Backward algorithm needs six rounds. One-Round and Chronicle generate route changes with one round, but without the guarantee of loop-freedom.

As can be seen in figure 7.43, Brute-force, Backward and Greedy do have a maximum of one lost ICMP-Echo in our tests. Chronicle and One-Round have the most packet loss with up to 20 lost ICMP-Echos and a median of 8 lost ICMP-Echos. Three-Round comes in between both with up to 13 lost ICMP-Echos and a median of 3 lost ICMP-Echos.

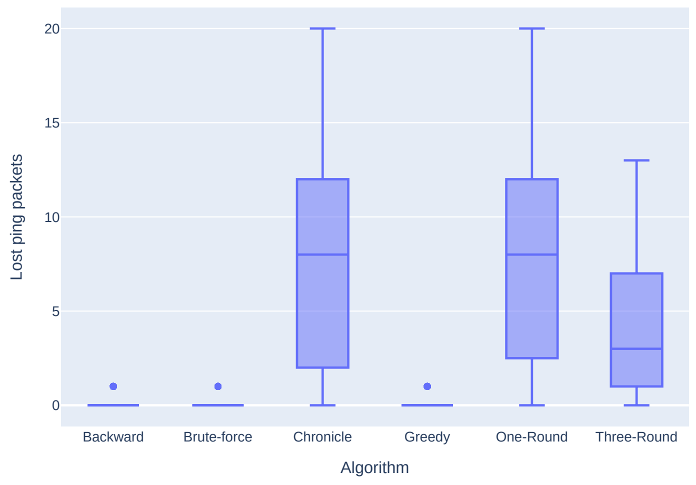


Figure 7.43.: Comparison of lost packets for the different algorithms

7.2.3. Performance Comparison

As is shown in figure 7.44, the fastest algorithms are Chronicle and One-Round with around 480 ms execution time, followed by Three-Round and Brute-force, which are nearly 230 ms slower. Greedy is 200 ms slower than Three-Round and Brute-force, but still a bit faster than Backward, which is still 100 ms slower than Greedy. This roughly fits with the number of rounds the algorithms need.

7.2.4. Comparison of Calculation Time

The calculation time differs for the different algorithms as shown in figure 7.45. While One-Round, Backward, Chronicle and Three-Round need less than one millisecond, Brute-force needs between 1.9 seconds and 2.1 seconds, Greedy between

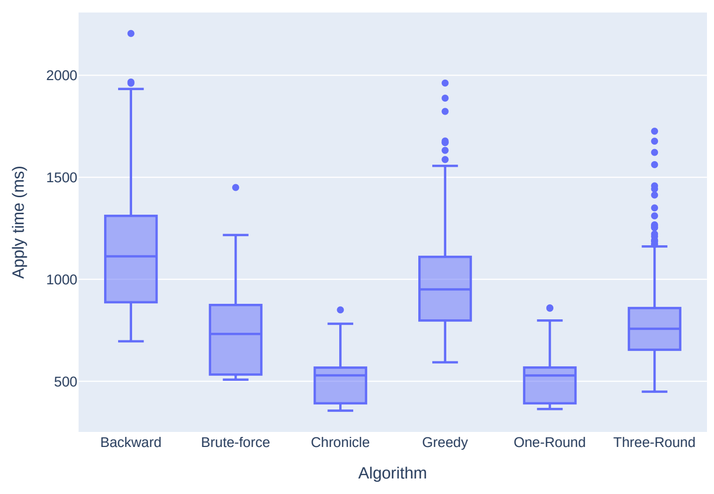


Figure 7.44.: Comparison of the time it took to apply the routes for the different algorithms

0.17 seconds and 0.21 seconds. This demonstrates the high calculation-complexity of Brute-force for complicated route changes.

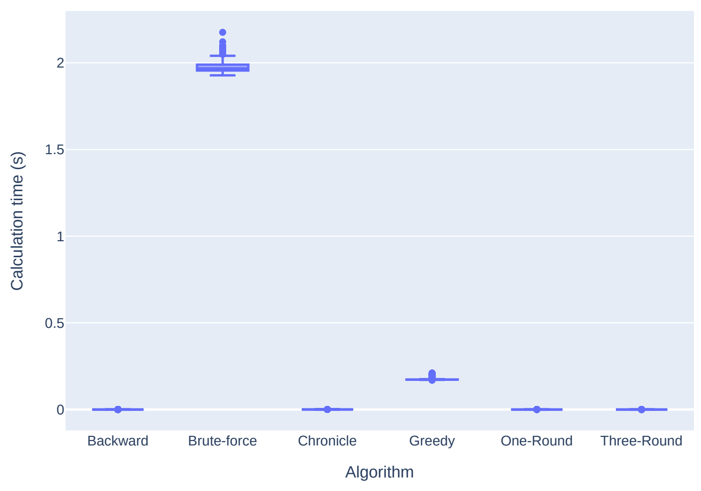


Figure 7.45.: Comparison of the time it took to calculate the routes for the different algorithms

7.2.5. Discussion Of Route Change 2

The results exhibit a greater variation by algorithm in comparison to our initial test route change.

If we stay with loop-free algorithms, Greedy is the best for this route change, as it can be calculated 1.8 seconds faster and executed only 0.2 seconds slower than Brute-force. If some packet loss resulting in slight temporary loss of bandwidth is acceptable, One-Round and Three-Round can both be calculated in less than a millisecond and executed in under 500 milliseconds. Three-Round takes 230 ms longer during the execution, but has less packet loss.

If there are no requirements for the time it takes to calculate the route change, Brute-force provides a very good result, as it is as fast as Three-Round, but without packet loss.

7.3. Going backward

We assess the algorithms' ability to produce schedules for route changes that alter the direction of the traffic flow on numerous links as a third test case. Each link that goes in the opposite direction than before potentially creates a small loop between the two connected routers. A key advantage of round-based route change algorithms is the ability to deal with potential loops, therefore, it is important to test them for small loops as well. We tested the following route changes. Each row represents a flow and each number a route. Figure 7.46 visualizes the changes.

1. 1, 2, 3, 4, 5, 6, 7, 8 to 1, 7, 6, 5, 4, 8
2. 2, 3, 4, 5, 6, 7 to 2, 6, 5, 4, 8, 7
3. 3, 2, 1, 8, 7, 6 to 3, 4, 8, 7, 6
4. 4, 3, 2, 1, 8, 7, 6, 5 to 4, 8, 1, 2, 3, 5

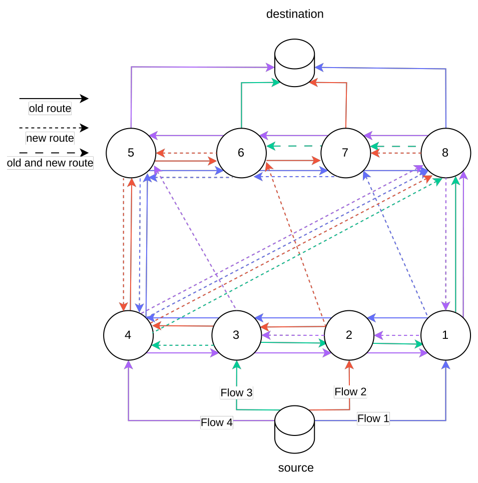


Figure 7.46.: Graphic of the third route change

We will present the results of this route change in chapter 7.3.1, followed by comparisons in chapter 7.3.2, 7.3.3 and 7.3.4. We will end the chapter by discussing the results in chapter 7.3.5.

7.3.1. Result from Algorithms

Backward

The Backward algorithm needs five rounds for the route change.

Figure 7.47 shows that the route change contains two rounds, without visible changes. Contrary to Greedy, Backward does not shorten the path first.

Interestingly, similar to the results in chapter 7.2.1, we can see some retries in figure 7.48 starting five seconds after all changes are already finished. Other than that, the retry values align with the ones we have seen during other route changes. We attribute them to the issue explained in chapter 6.4.2. The bandwidth stays at 50 Mbit/s.

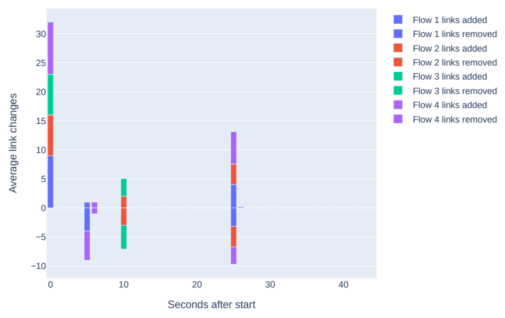


Figure 7.47.: Links disappearing and newly appearing in traceroutes when running the Backward algorithm

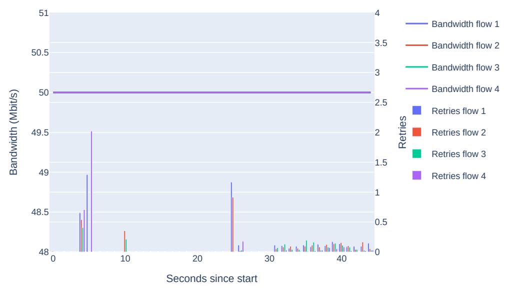


Figure 7.48.: Bandwidth and retries during the execution of the Backward algorithm

As shown in figure 7.49, the time it took to apply the changes using Backward was between 557 ms and 1696 ms.

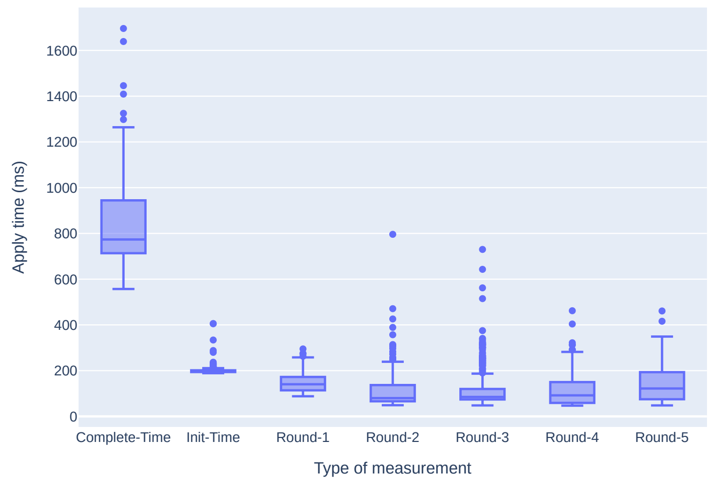


Figure 7.49.: Time that it takes to apply the changes using the Backward algorithm, if we leave no time between rounds

Greedy

Greedy calculates the route change in four rounds.

Looking at figure 7.50, we can see some packet loss throughout and after the changes, with most at the 5 and 20 second mark. We attribute this package-loss to the bug explained in chapter 6.4.2. The bandwidth stays at 50 Mbit/s throughout the test.

The latency in figure 7.51 falls in round one, which indicates that the route is first being shorted and later is extended again. It falls in round two for flows 3 and 4, again. The latency of all flows rises in round four. There are small latency spikes when rounds are applied.

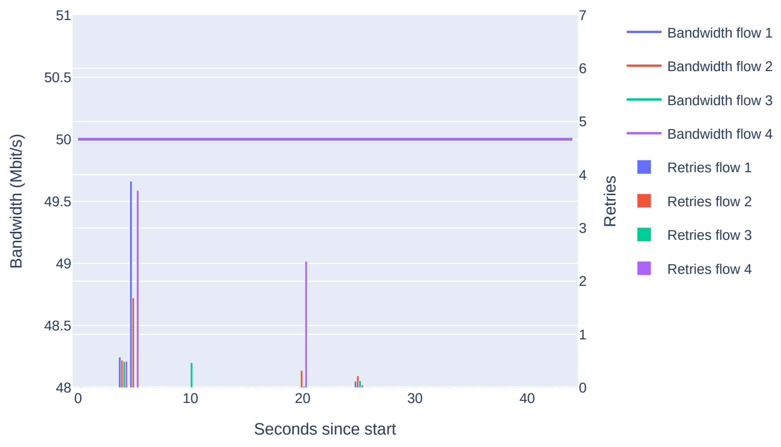


Figure 7.50.: Bandwidth and retries during the execution of the Greedy algorithm

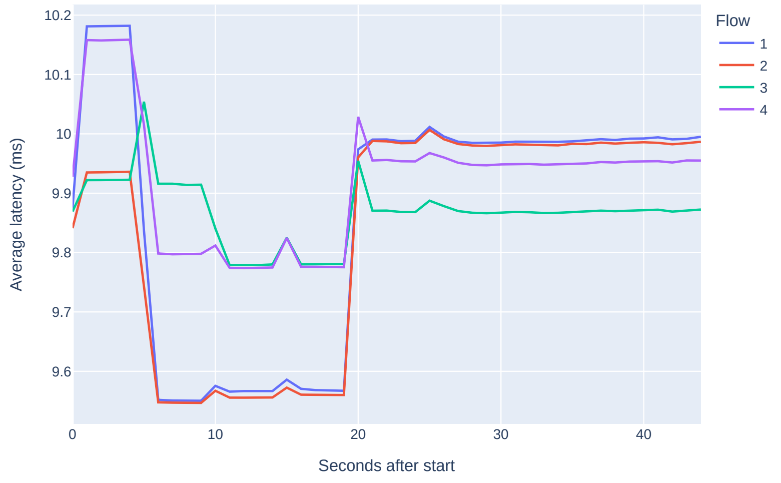


Figure 7.51.: Latency between source and destination of all flows during a route change with the Greedy algorithm

The time it takes to apply those changes was, in our tests, between 496 ms and 1395 ms with a median of 696 ms. As is shown in figure 7.52, round one takes the longest time with 121 ms to 323 ms.

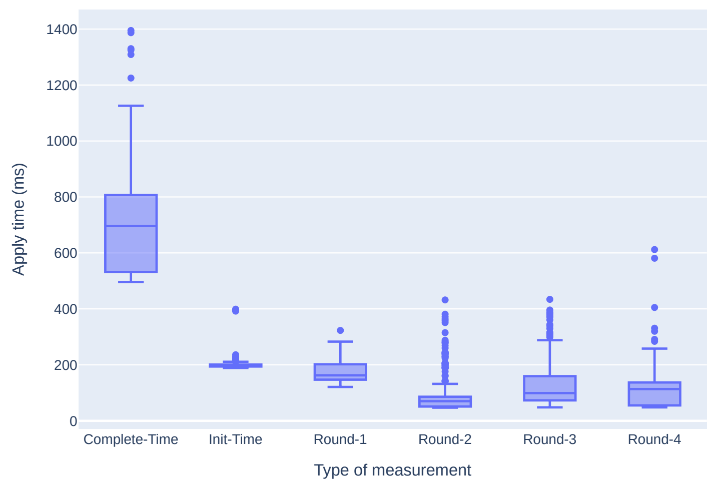


Figure 7.52.: Time that it takes to apply the changes using the greedy algorithm, if we leave no time between rounds

Brute-force

Brute-force and Greedy generate the same route change. The results do not differ much.

All results can be found in appendix A.

Chronicle

Chronicle schedules all changes in the first round, as there is no congestion.

Contrary to the previous algorithms, Chronicle, executed in a non-time-based network, generates a lot more packet loss and therefore needs more retries. As we can see in figure 7.53, all flows are negatively affected by round one. Flow 3 does even suffer a short bandwidth drop.

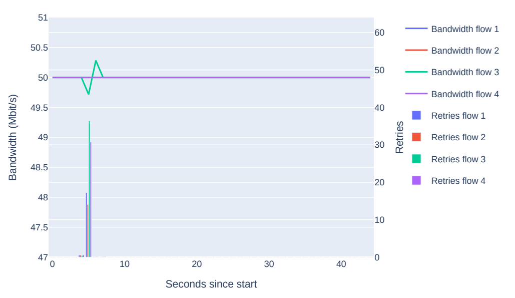


Figure 7.53.: Bandwidth and retries during the execution of the Chronicle algorithm

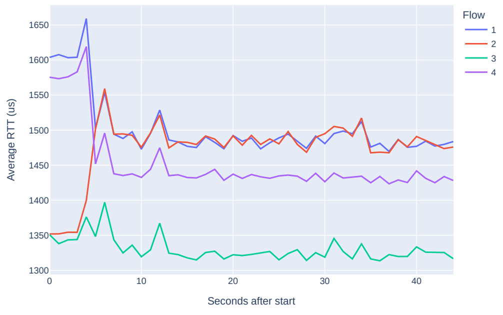


Figure 7.54.: Round-Trip-Time between source and destination of all flows during the chronicle algorithm

The propagation time is between 354 ms and 735 ms, with a median of 493.5 ms, as is visible in figure 7.55.

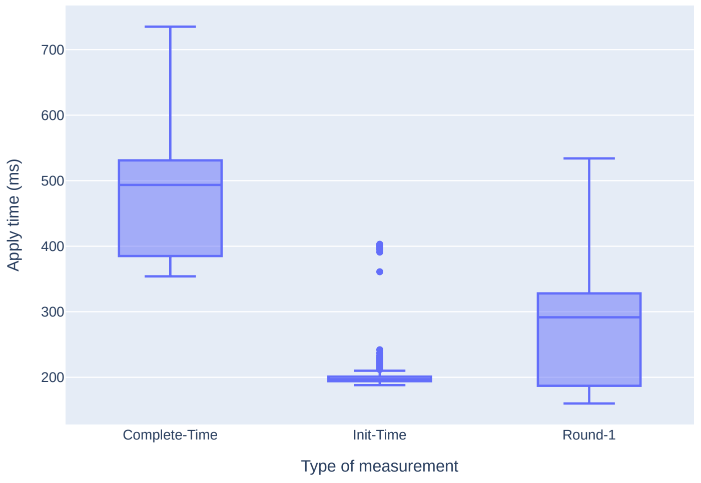


Figure 7.55.: Time that it takes to apply the changes using the chronicle algorithm, if we leave no time between rounds

One-Round

Brute-force and Greedy generate the same route change. The results do therefore not differ much.

One noticeable difference is that we cannot see any bandwidth drop in figure 7.56 during the test.

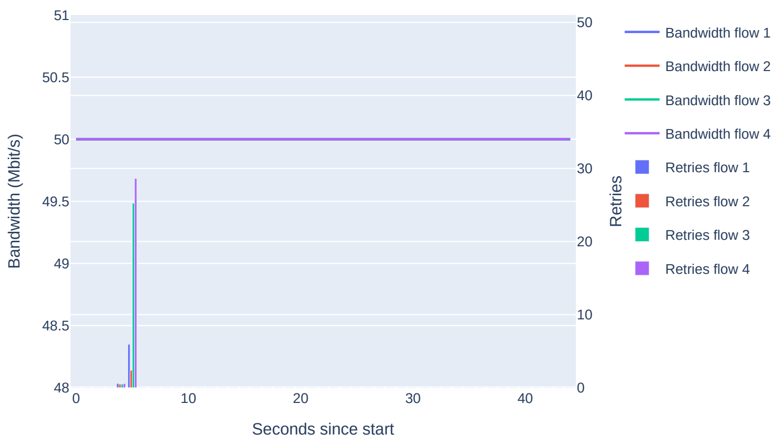


Figure 7.56.: Bandwidth and retries during the execution of the One-Round algorithm

All results can be found in appendix A.

Three-Round

Three-Round generated a route change with three rounds.

While flow one and two have similar numbers of retries to One-Round, the retries of flow four are even higher. The retries of flow three, as can be seen in figure 7.57, are significantly lower and comparable to loop-free algorithms. The bandwidth stays at 50 Mbit/s throughout the test. There are some retries, starting around the time, where the cleanup round ran.

The latency and round trip time are comparable to Chronicle, but shifted by five seconds.

The propagation time of Three-Round was between 469 ms and 1558 ms, with 702.5 ms being the median. Figure 7.58 shows that 74 ms to 841 ms of that time can be attributed to the cleanup round.

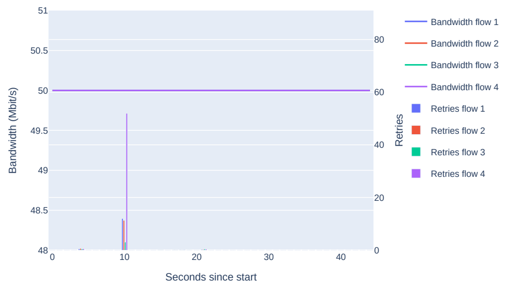


Figure 7.57.: Bandwidth and retries during the execution of the Three-Round algorithm

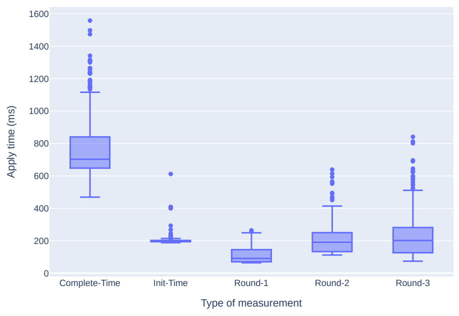


Figure 7.58.: Time that it takes to apply the changes using the Three-Round algorithm, if we leave no time between rounds

7.3.2. Comparison of Algorithms

Loop-free route-update algorithms schedule four (Brute-force, Greedy) or five (Backward) rounds to achieve this route change. Algorithms not adhering to the principals of loop-freedom need one (One-Round, Chronicle) or three rounds (Three-Round).

When comparing the results of our ping tests in figure 7.59, the difference in packet loss between loop-free algorithms and non-loop-free algorithms is clearly visible. While Backward, Brute-force and Greedy have a maximum of two lost ICMP-Echo packets, One-Round and Chronicle have a maximum of 12. Three-Round has a maximum of 10 lost ICMP-Echo packets, but a median of 0, like loop-free algorithms.

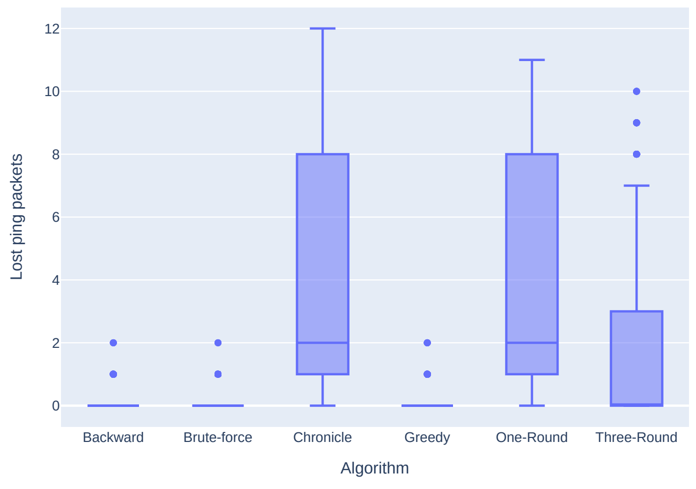


Figure 7.59.: Comparison of lost packets for the different algorithms

7.3.3. Performance Comparison

As can be expected and is shown in figure 7.60, the algorithms with only one round apply faster than the algorithms with more than one round. Brute-force, Greedy, and Three-Round are typically around 200 ms slower than Chronicle and One-Round. Backward is typically 70ms slower than Brute-force, Greedy, and Three-Round.

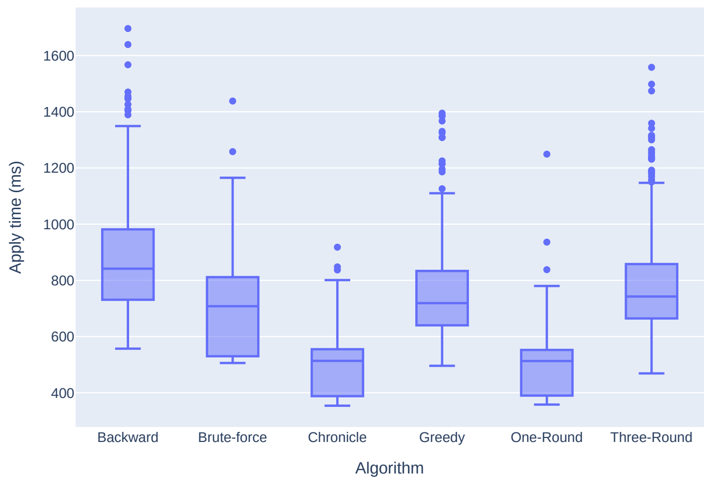


Figure 7.60.: Comparison of the time it took to apply the routes for the different algorithms

7.3.4. Comparison of Calculation Time

Figure 7.61 shows that all algorithms calculated a schedule in a relatively short time. Backward, Chronicle, One-Round and Three-Round need less than a millisecond. Greedy needs between 0.16 s and 0.21 s and Brute-force between 0.21 s and 0.22 s.

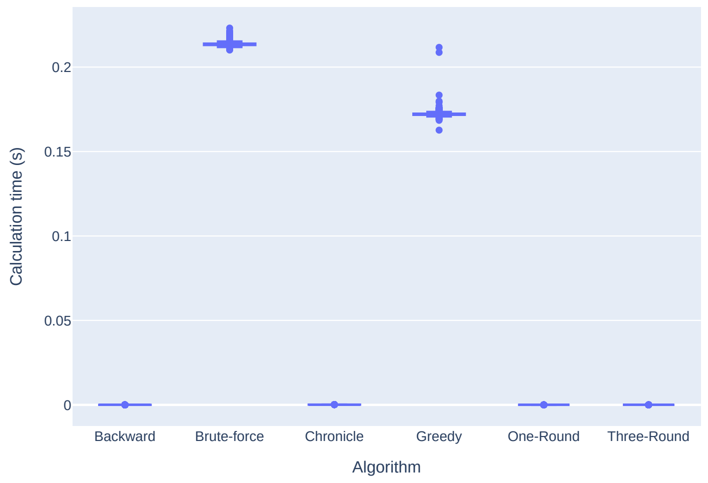


Figure 7.61.: Comparison of the time it took to calculate the routes for the different algorithms

7.3.5. Discussion Of Route Change 3

The route change is less complex than the one in chapter 7.2. The differences between the algorithms are smaller. When minimizing packet loss, we recommend Greedy and Backward for this route change, as they both provide a solution with almost no packet loss and a similar execution time.

One-Round and Chronicle are still 200 ms faster, but have around 6 times more packet loss. Therefore, if time is important and packet loss does not matter as much, One-Round seems to be the best fit.

8. Discussion

In chapter 7.1, 7.2, and 7.3 we looked at the results of running the algorithms described in chapter 3 on real world hardware. We will discuss the results in this chapter, by looking at the results. We use the median values for all time measurements.

The results show that, while there are significant effects to the packet loss resulting from the choice between loop-free algorithms and non-loop-free algorithms, it only slightly affects the bandwidth. The propagation time of the algorithms adhering to the rules of loop-freedom was, in our more complex tests, almost double that of those that do not.

The Backward algorithm calculated all our route changes in less than a millisecond and delivered loop-free results with only minimal packet loss. The propagation time of Backward was the worst of the algorithms tested, with 1000 ms and 773 ms in test two and three. Test one had identical schedules with Backward, Greedy and Brute-force that applied in 386ms. In both complex cases (chapter 7.2 and chapter 7.3), the Backward algorithm needed the most rounds with one or two rounds more than other loop-free algorithms, as it needs one round per changing route per flow. This causes the Backward algorithm to scale worse than other algorithms.

The Greedy algorithm needed significantly more time to calculate than Backward, but generated loop-free schedules with only minimal packet loss and fewer rounds in both complex test scenarios. It took approximately 0.17 s to calculate each route change, but is faster to apply the changes than when using Backward, with 906ms and 696ms in tests two and three. Greedy should be able to scale better than Backward, as it does not need one extra round per changing route.

Brute-force generated loop-free schedules with only minimal packet loss and the fewest rounds for all three tests. While the calculation time for our first, very simple test was equal to the calculation time of Greedy, it had significantly worse results in our other tests. In test three, it was already 40 ms slower. In test two, it took more than ten times as long as Greedy at 1.96 s. Test two shows that Brute-force can be quite slow to calculate if confronted with a complicated route change. It had the shortest apply-time of loop-free algorithms with 386 ms, 732 ms and 708 ms for tests one to three.

Chronicle and One-Round generated the same schedules for all three test cases, as we did not test a network with congestion. The schedules were not loop-free, but used flow swapping. The calculation time of One-Round and Chronicle was quite similar, with One-Round being 120 μs to 180 μs faster. One-Round's calculation time was similar to the calculation time of the Backward algorithm. Chronicle's and

One-Round's schedules could be applied in 408ms, 480ms and 494ms in tests one to three. We could observe up to 3% packet loss during the second route update. This translates to around 5 ms to 10 ms loss of connectivity. While this is more than loop-free algorithms, that experienced up to 1 ms of packet loss in our tests, it is still significantly less than the around 300 ms the handover between wireless networks causes [Zha+07]. Mizrahi and Moses [MM16a] state that if route changes occur every minute, then transient disruptions must be limited to just a few milliseconds.

Three-Round scheduled fewer rounds than loop-free algorithms in test two and three. It was calculated in less than a millisecond and applied in 624 ms, 706 ms and 702 ms. It produced less packet loss than Chronicle and One-Round for some flows, with sometimes producing as little as loop-free algorithms. In our first test, it had a maximum of only one lost ICMP-Echo packet, compared to the maximum of seven packets for One-Round and Chronicle. During the second and third test, Three-Round caused more lost packets than loop-free algorithms with a maximum of 13 and 10 lost ICMP-Echos compared to the one and two by loop-free algorithms. In contrast to this, Chronicle and One-Round caused a maximum of 20 and 12 lost ICMP-Echos. When looking at the flows in direct comparison, it is apparent that Three-Round heavily profits from new routers that appear in the path. If all routers of the new path are already present in the old path, it cannot do anything in the first round.

Choosing the correct algorithm depends on the specific use-case. If you have time or computing power to calculate changes beforehand, Brute-force is probably the best. Greedy had fewer rounds than Backward in all our tests, while Backward was significantly faster to calculate. If you do not have time, but can accept the up to 3% packet loss during the route update, One-Round will achieve good results.

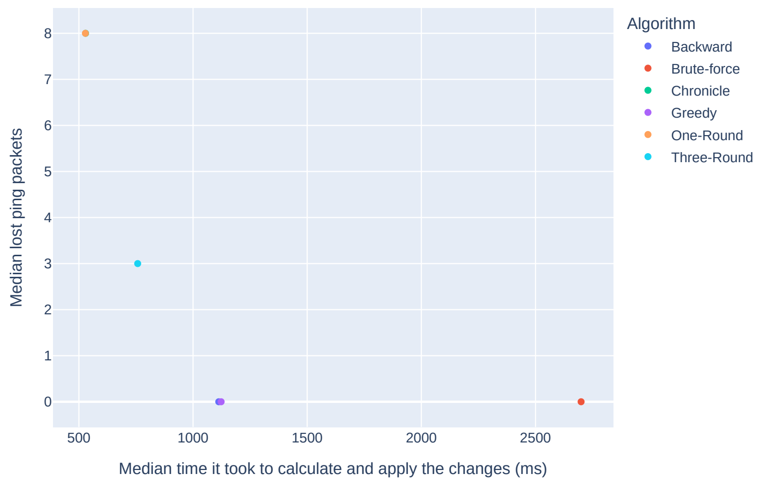


Figure 8.1.: Plot displaying the time it took to facilitate a route change and the package loss for our second test

We identify this as the conflict between packet loss and time. Figure 8.1 makes this conflict visible: None of the algorithms we tested could generate fast and packet loss free results. Three-Round has a spot in between, with often slightly faster calculation times than loop-free algorithms and slightly less packet loss than Chronicle and One-Round.

We executed all algorithms excluding Chronicle for each flow individually and combined the results in one route change schedule, as suggested by Mahajan and Wattenhofer [MW13]. We can therefore answer our first research question defined in chapter 4 “How can algorithms for loop-free route updates in Software Defined Networks be applied in multi-stream networks?”: As long as the flows cannot create any congestion and the flows are routed by destination [MW13], combining the results of the algorithms for single flows achieves good results for route changes in multi-flow environments. There is no need for specialized algorithms for multiple streams in this case.

Our second research question “How much packet loss does adapting time-based algorithms cause for non-time-based Software Defined Networks?” is answered by the numbers outlined above. We could observe up to 3% packet loss in our experiments with time-based algorithms. In some cases, the packet loss of time-based algorithms was 15 times that of loop-free algorithms.

9. Conclusion

In chapter 9.1, we will first summarize the results of this thesis. In chapter 9.2, we will look at open questions that can be answered by further work.

9.1. Results

This thesis looked at the five route-update algorithms Backward, Brute-force, Greedy, Chronicle and One-Round already researched in literature. Additionally, we proposed our own algorithm Three-Round, which works by updating routers that do not have active traffic first.

Some of these algorithms are optimized for time-based networks. As these are not yet common in real-world environments, we tried to emulate their behaviour. In order to simultaneously update all routers, we proposed and tested sending a script file to all routers as a first step. Afterwards, we executed the script file on all routers at the same time.

We constructed a network of eight routers, one source, one destination and one controller. To facilitate multiple tests, we developed four tools working together to execute a route change and measure the effects of it.

In order to further research networks with multiple flows, we executed all our tests on four flows at the same time. During our tests, we did not run into any problems caused by applying the route changes to multiple flows simultaneously. We can therefore conclude that these algorithms work in multi-stream environments.

All our tests with non-time-based Software Defined Networks using time-based algorithms did show packet loss. These did sometimes result in small bandwidth drops. We could observe up to 30 ms connectivity loss during these route updates. Most of the time it was around 5ms to 10ms.

These results underline the importance of choosing a route-update algorithm that fits the use-case perfectly. The choice of algorithm creates a conflict between packet loss and time. Algorithms adhering to the principals of round-based updates and loop-freedom will, in most cases, have significantly less packet loss, but higher execution times than algorithms that do not.

9.2. Future Work

This thesis focuses on the questions raised in chapter 4. There are still a lot of open questions in the realm of route-updates.

As our routers were quite limited in respect to bandwidth, we think that testing Chronicle with routers that have higher routing throughput might be interesting. This way, one could test the congestion avoidance features of Chronicle. This is already done for time-based SDNs by Mahajan and Wattenhofer [MW13] when introducing Chronicle, but could be further tested on non-time-based SDNs like the one in this thesis.

While we upload a script containing the changes needed to all routers first, we started the execution of those from our central controller. An interesting approach to emulate time-based SDNs even better with routers would be to use an inbuilt scheduler to schedule the execution of the script. This is not something we have looked into, as our routers did not support PTP, and we hoped to achieve higher accuracy by starting the execution centrally than with time synchronization using NTP.

While this thesis briefly looked into reducing the execution time of the route changes, we could not compare the execution time with other technologies. Further research into how technologies, like the ones we and Alkhatib [Alk24b] used, affect the propagation time is needed to choose the perfect technology for a specific environment.

As we wanted to create a realistic environment with hardware routers, we were limited in regard to the amount of routers. One could look further into replicating the results of this thesis in virtual networks, where it is possible to emulate hundreds of routers and with delays between them.

These further works would lead to a deeper understanding of time-based SDNs and efficient route updates.

Bibliography

[Alk24a] Bashar Alkhatib. “Automatisierung und Optimierung von relaxierten kreisfreien Routingupdates mit Ansible”. Bachelor’s Thesis. TU Dortmund University, 2024.

[Alk24b] Mohamed Nour Alkhatib. “Untersuchung von automatisierten kreisfreien Routingupdates mit Ansible”. Bachelor’s Thesis. TU Dortmund University, 2024.

[Ami+16] Saeed Akhoondian Amiri, Arne Ludwig, Jan Marcinkowski, and Stefan Schmid. “Transiently Consistent SDN Updates: Being Greedy is Hard”. In: *Structural Information and Communication Complexity*. Ed. by Jukka Suomela. Cham: Springer International Publishing, 2016, pp. 391–406. ISBN: 978-3-319-48314-6.

[Che22] Djalel Chefrour. “Evolution of network time synchronization towards nanoseconds accuracy: A survey”. In: *Computer Communications* 191 (2022), pp. 26–35. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2022.04.023. URL: https://www.sciencedirect.com/science/article/pii/S0140366422001359.

[DMR21] Sushil C. Dimri, Preeti Malik, and Mangey Ram. *Design and Analysis*. Berlin, Boston: De Gruyter, 2021. ISBN: 9783110693607. DOI: doi:10.1515/9783110693607. URL: https://doi.org/10.1515/9783110693607.

[Doc] MikroTik API Documentation. https://help.mikrotik.com/docs/spaces/ROS/pages/47579160/API. Accessed: 2025-01-16.

[För+18] Klaus-Tycho Förster, Arne Ludwig, Jan Marcinkowski, and Stefan Schmid. “Loop-Free Route Updates for Software-Defined Networks”. In: *IEEE/ACM Transactions on Networking* 26.1 (2018), pp. 328–341. DOI: 10.1109/TNET.2017.2778426.

[FSV19] Klaus-Tycho Förster, Stefan Schmid, and Stefano Vissicchio. “Survey of Consistent Software-Defined Network Updates”. In: *IEEE Communications Surveys & Tutorials* 21.2 (2019), pp. 1435–1461. DOI: 10.1109/COMST.2018.2876749.

[Inc] Red Hat Inc. *How Ansible works*. https://www.redhat.com/en/ansible-collaborative/how-ansible-works?intcmp=7015Y000003t7aWQAQ. Accessed: 2025-03-24.

[Jal+17] Ahmad Jalili, Hamed Nazari, Sahar Namvarasl, and Manijeh Keshtgari. "A comprehensive analysis on control plane deployment in SDN: In-band versus out-of-band solutions". In: *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*. 2017, pp. 1025–1031. DOI: 10.1109/KBEI.2017.8324949.

[Jun99] Dieter Jungnickel. "The Greedy Algorithm". In: *Graphs, Networks and Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 129–153. ISBN: 978-3-662-03822-2. DOI: 10.1007/978-3-662-03822-2_5. URL: https://doi.org/10.1007/978-3-662-03822-2_5.

[Kre+15] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. "Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76. DOI: 10.1109/JPROC.2014.2371999.

[LMS15] Arne Ludwig, Jan Marcinkowski, and Stefan Schmid. "Scheduling Loop-free Network Updates: It's Good to Relax!" In: *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*. PODC '15. Donostia-San Sebastián, Spain: Association for Computing Machinery, 2015, pp. 13–22. ISBN: 9781450336178. DOI: 10.1145/2767386.2767412. URL: https://doi.org/10.1145/2767386.2767412.

[Mat+16] Ferrazani Mattos, Diogo Menezes, Muniz Bandeira Duarte, Otto Carlos, and Guy Pujolle. "Reverse Update: A Consistent Policy Update Scheme for Software-Defined Networking". In: *IEEE Communications Letters* 20.5 (2016), pp. 886–889. DOI: 10.1109/LCOMM.2016.2546240.

[MM16a] Tal Mizrahi and Yoram Moses. "Software defined networks: It's about time". In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524418.

[MM16b] Tal Mizrahi and Yoram Moses. "Time4: Time for SDN". In: *IEEE Transactions on Network and Service Management* 13.3 (2016), pp. 433–446. DOI: 10.1109/TNSM.2016.2599640.

[MW13] Ratul Mahajan and Roger Wattenhofer. "On consistent updates in Software Defined Networks". In: *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*. HotNets-XII. College Park, Maryland: Association for Computing Machinery, 2013. ISBN: 9781450325967. DOI: 10.1145/2535771.2535791. URL: https://doi.org/10.1145/2535771.2535791.

[Sal94] Peter H. Salus. *A quarter century of UNIX*. USA: ACM Press/Addison-Wesley Publishing Co., 1994. ISBN: 0201547775.

[Vin02] A. Vince. "A framework for the greedy algorithm". In: *Discrete Applied Mathematics* 121.1 (2002), pp. 247–260. ISSN: 0166-218X. DOI: https://doi.org/10.1016/S0166-218X(01)00362-6. URL: https://www.sciencedirect.com/science/article/pii/S0166218X01003626.

[XLH] Jiqiang Xia, Julong Lan, and Yuxiang Hu. "P4lof: Scheduling Loop-Free Updates for Multiple Flows in Sdn". In: *Available at SSRN 4106297* ().

[Zha+07] Yanfeng Zhang, Yongqiang Liu, Yong Xia, and Quan Huang. “LeapFrog: Fast, Timely WiFi Handoff”. In: *IEEE GLOBECOM 2007 - IEEE Global Telecommunications Conference*. 2007, pp. 5170–5174. DOI: 10.1109/GLOCOM.2007.980.

[Zhe+17] Jiaqi Zheng, Guihai Chen, Stefan Schmid, Haipeng Dai, and Jie Wu. “Chronus: Consistent Data Plane Updates in Timed SDNs”. In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 319–327. DOI: 10.1109/ICDCS.2017.96.

[Zhe+18] Jiaqi Zheng, Bo Li, Chen Tian, Klaus-Tycho Förster, Stefan Schmid, Guihai Chen, and Jie Wux. “Scheduling Congestion-Free Updates of Multiple Flows with Chronicle in Timed SDNs”. In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 2018, pp. 12–21. DOI: 10.1109/ICDCS.2018.00012.

[Zhe+19] Jiaqi Zheng, Bo Li, Chen Tian, Klaus-Tycho Förster, Stefan Schmid, Guihai Chen, Jie Wu, and Rui Li. “Congestion-Free Rerouting of Multiple Flows in Timed SDNs”. In: *IEEE Journal on Selected Areas in Communications* 37.5 (2019), pp. 968–981. DOI: 10.1109/JSAC.2019.2906741.

A. Appendix: Results

Results from chapter 7.1

Backward

Leaving 5 seconds between rounds

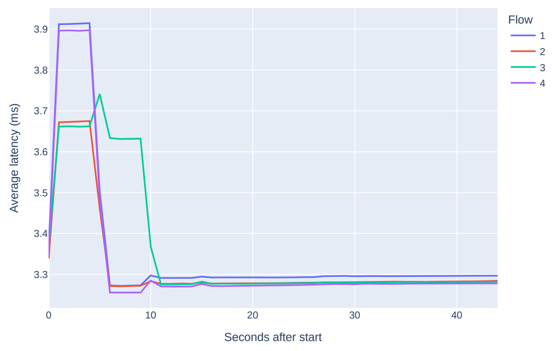


Figure A.3.: Latency between source and destination of all flows during a route change with the Backward algorithm

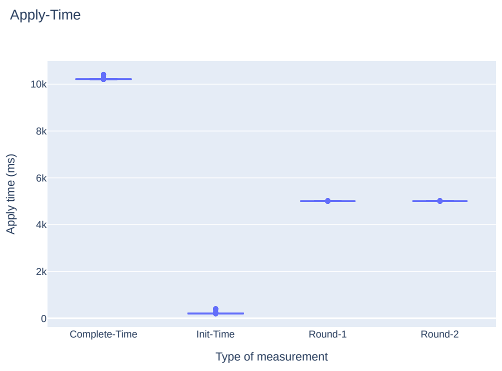


Figure A.1.: Time that it takes to apply the changes using a schedule generated by Backward

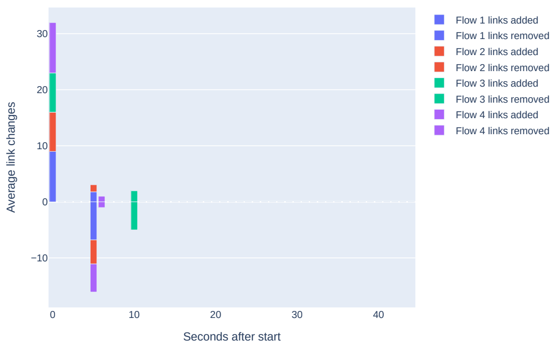


Figure A.4.: Links disappearing and newly appearing in traceroutes when running the Backward algorithm

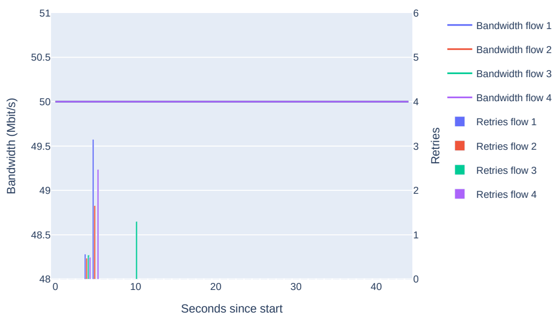


Figure A.2.: Bandwidth and retries during the execution of the Backward algorithm

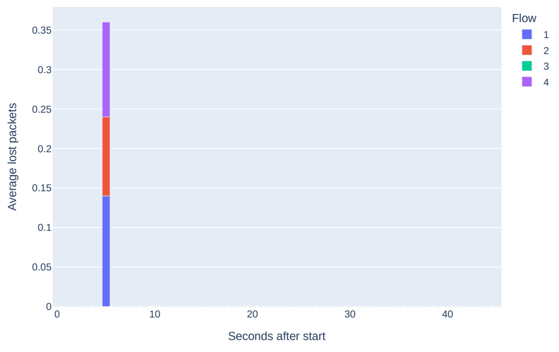


Figure A.5.: Lost packets in our ping test when executing a route change with the Backward algorithm

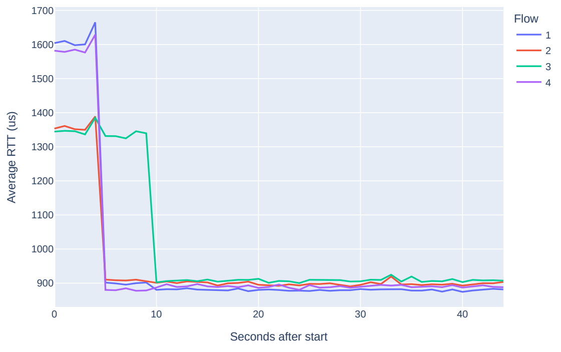


Figure A.6.: Round-Trip-Time between source and destination of all flows during the route changes of the Backward algorithm

Leaving no time between rounds

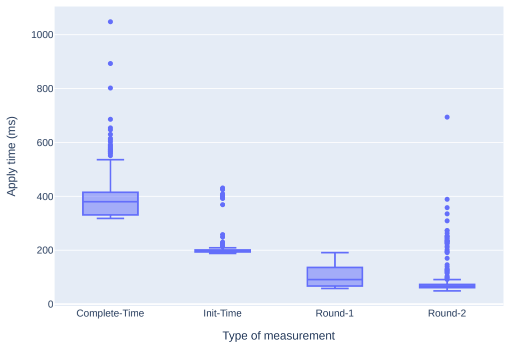


Figure A.7.: Time that it takes to apply the changes using a schedule generated by Backward

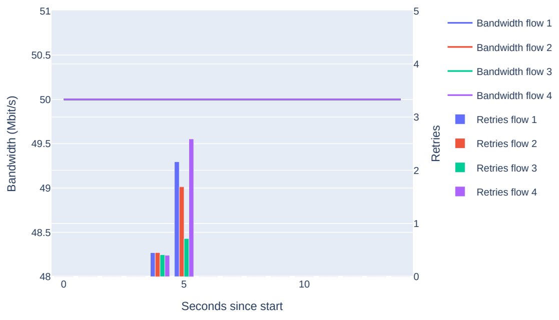


Figure A.8.: Bandwidth and retries during the execution of the Backward algorithm

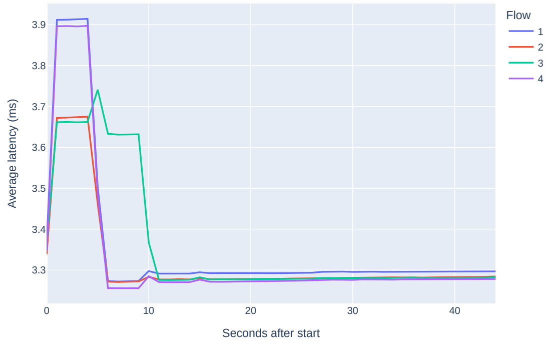


Figure A.9.: Latency between source and destination of all flows during a route change with the Backward algorithm

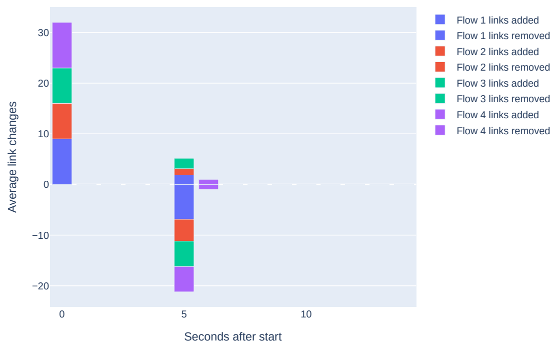


Figure A.10.: Links disappearing and newly appearing in traceroutes when running the Backward algorithm

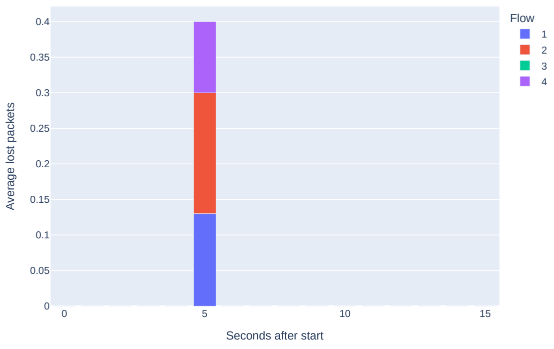


Figure A.11.: Lost packets in our ping test when executing a route change with the Backward algorithm

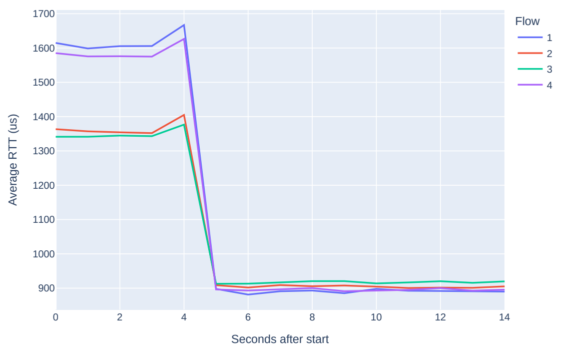


Figure A.12.: Round-Trip-Time between source and destination of all flows during the route changes of the Backward algorithm

Brute-force

Leaving 5 seconds between rounds

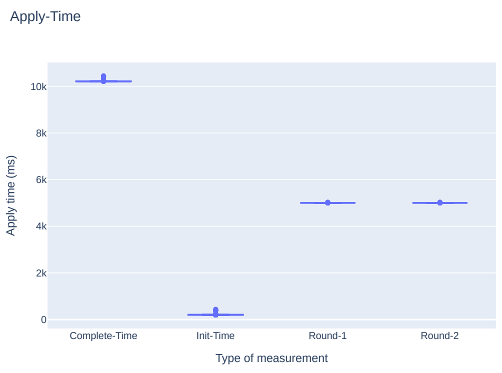


Figure A.13.: Time that it takes to apply the changes using a schedule generated by Brute-force

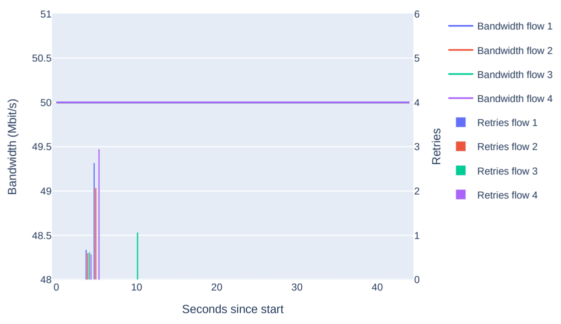


Figure A.14.: Bandwidth and retries during the execution of the Brute-force algorithm

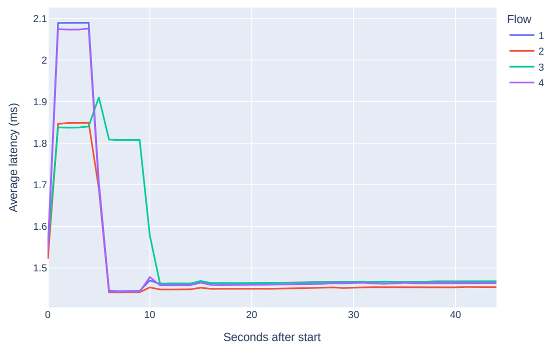


Figure A.15.: Latency between source and destination of all flows during a route change with the Brute-force algorithm

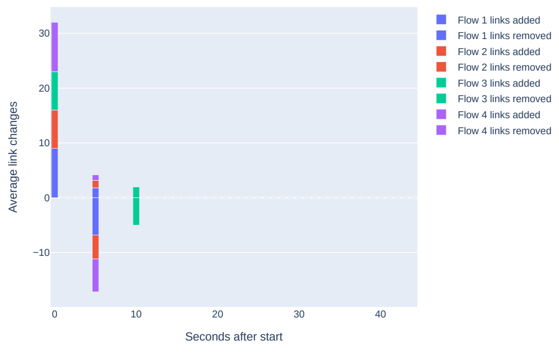


Figure A.16.: Links disappearing and newly appearing in traceroutes when running the Brute-force algorithm

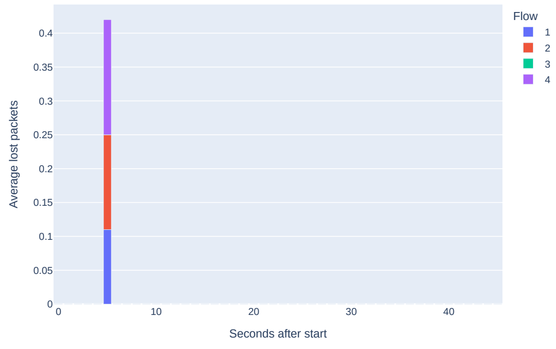


Figure A.17.: Lost packets in our ping test when executing a route change with the Brute-force algorithm

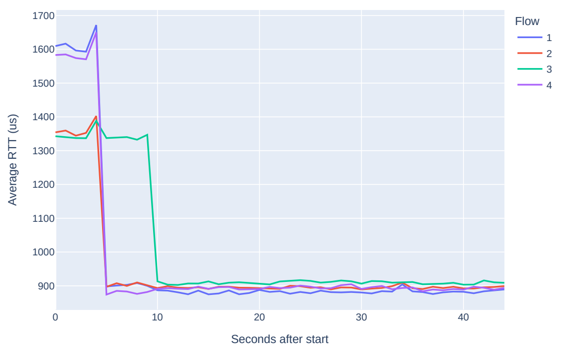


Figure A.18.: Round-Trip-Time between source and destination of all flows during the route changes of the Brute-force algorithm

Leaving no time between rounds

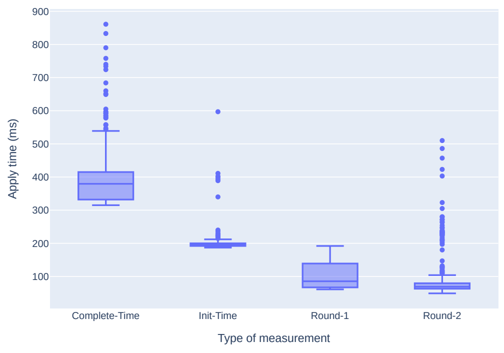


Figure A.19.: Time that it takes to apply the changes using a schedule generated by Brute-force

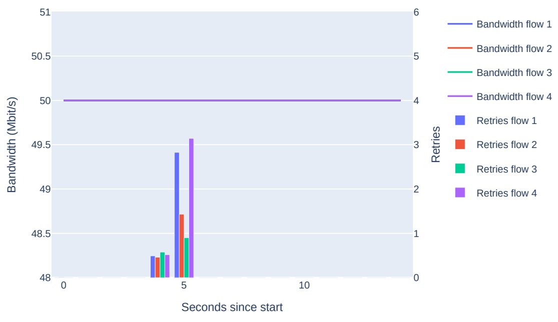


Figure A.20.: Bandwidth and retries during the execution of the Brute-force algorithm

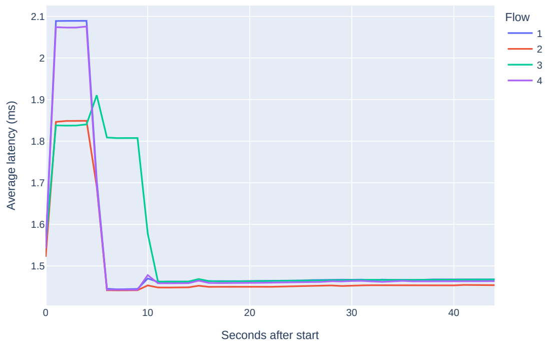


Figure A.21.: Latency between source and destination of all flows during a route change with the Brute-force algorithm

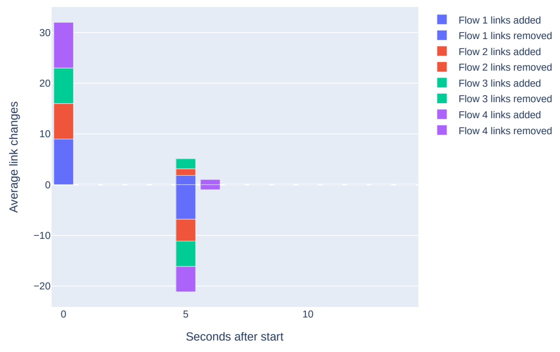


Figure A.22.: Links disappearing and newly appearing in tracer-outes when running the Brute-force algorithm

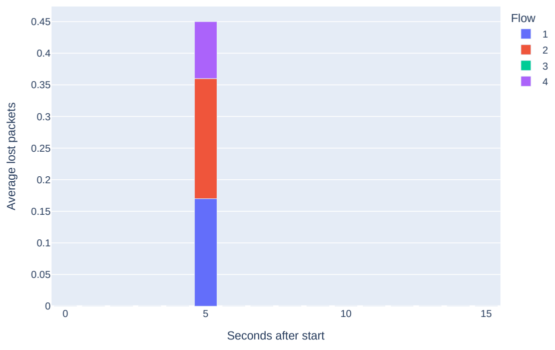


Figure A.23.: Lost packets in our ping test when executing a route change with the Brute-force algorithm

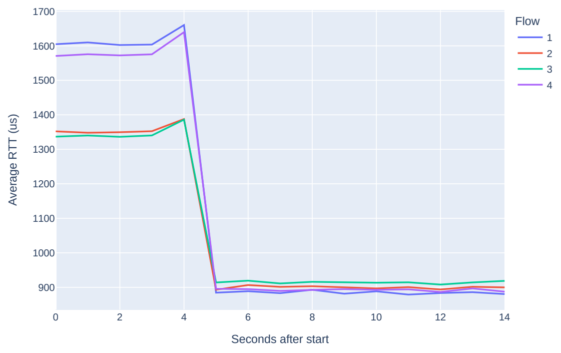


Figure A.24.: Round-Trip-Time between source and destination of all flows during the route changes of the Brute-force algorithm

Chronicle

Leaving 5 seconds between rounds

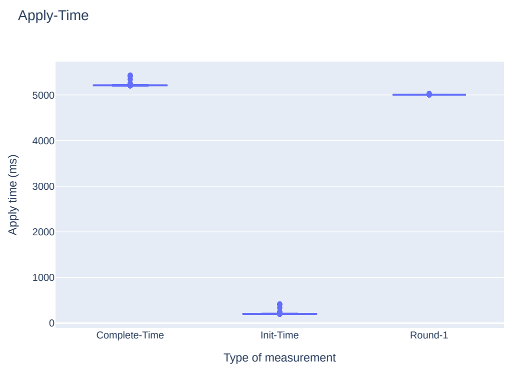


Figure A.25.: Time that it takes to apply the changes using a schedule generated by Chronicle

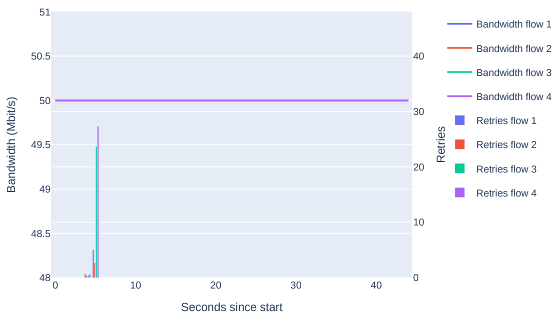


Figure A.26.: Bandwidth and retries during the execution of the Chronicle algorithm

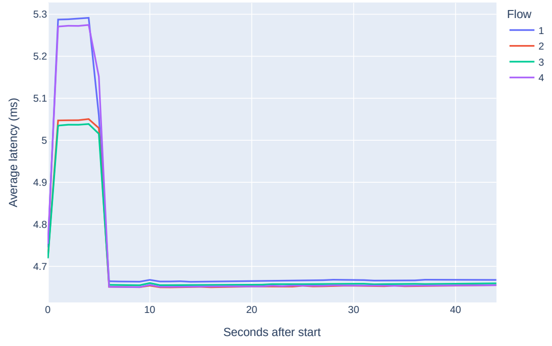


Figure A.27.: Latency between source and destination of all flows during a route change with the Chronicle algorithm

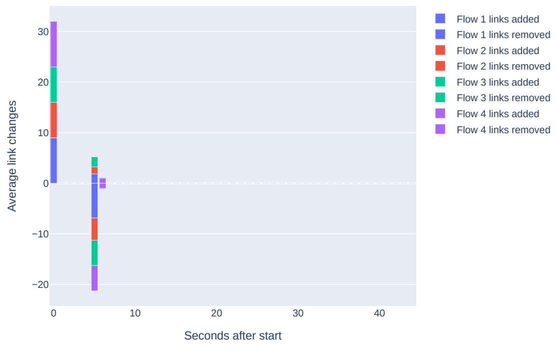


Figure A.28.: Links disappearing and newly appearing in traceroutes when running the Chronicle algorithm

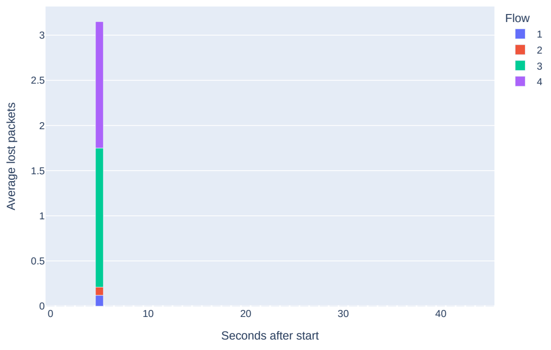


Figure A.29.: Lost packets in our ping test when executing a route change with the Chronicle algorithm

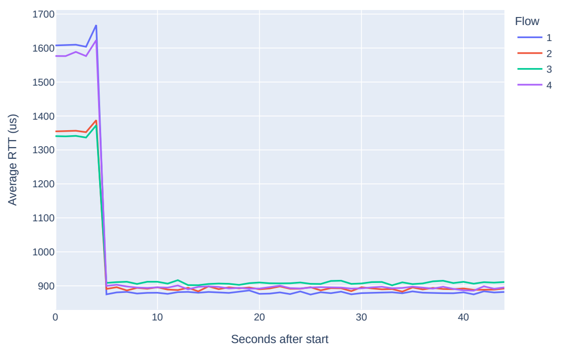


Figure A.30.: Round-Trip-Time between source and destination of all flows during the route changes of the Chronicle algorithm

Leaving no time between rounds

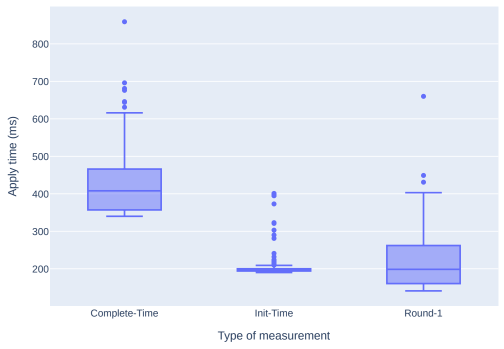


Figure A.31.: Time that it takes to apply the changes using a schedule generated by Chronicle

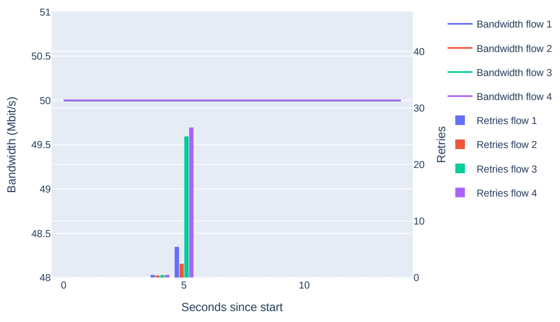


Figure A.32.: Bandwidth and retries during the execution of the Chronicle algorithm

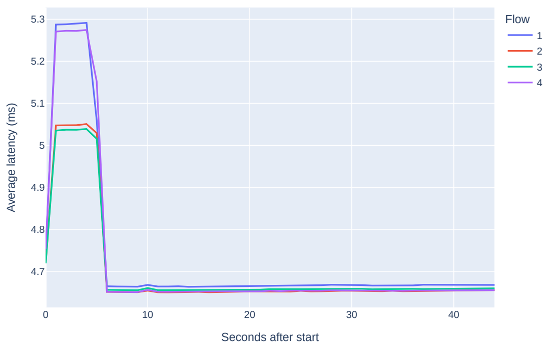


Figure A.33.: Latency between source and destination of all flows during a route change with the Chronicle algorithm

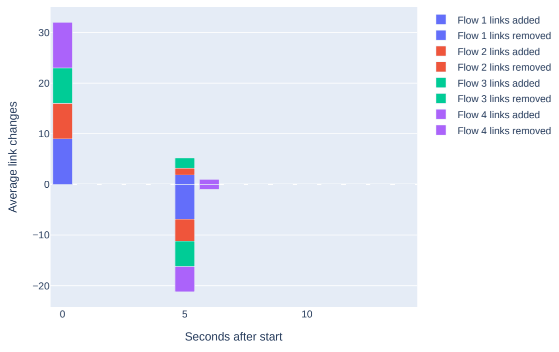


Figure A.34.: Links disappearing and newly appearing in traceroutes when running the Chronicle algorithm

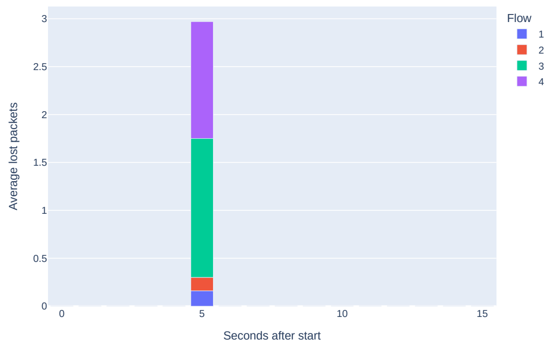


Figure A.35.: Lost packets in our ping test when executing a route change with the Chronicle algorithm

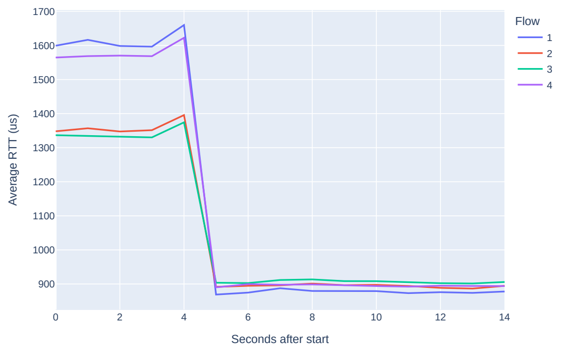


Figure A.36.: Round-Trip-Time between source and destination of all flows during the route changes of the Chronicle algorithm

Greedy

Leaving 5 seconds between rounds

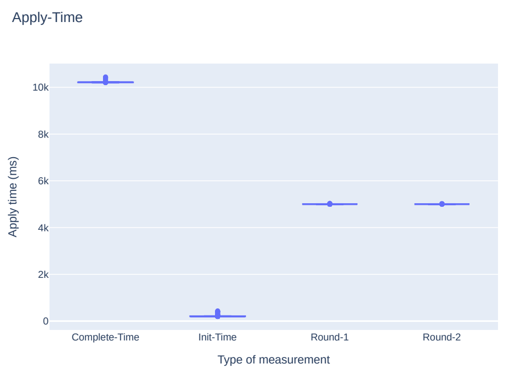


Figure A.37.: Time that it takes to apply the changes using a schedule generated by Greedy

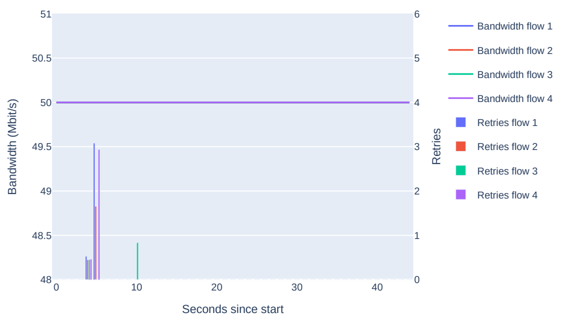


Figure A.38.: Bandwidth and retries during the execution of the Greedy algorithm

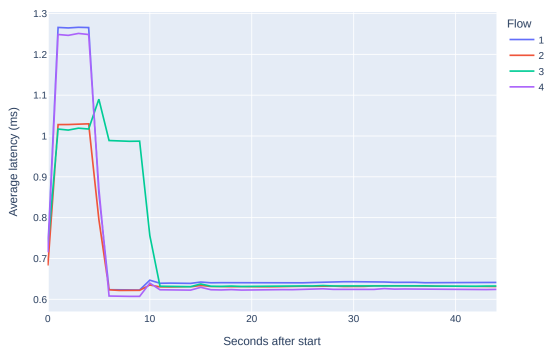


Figure A.39.: Latency between source and destination of all flows during a route change with the Greedy algorithm

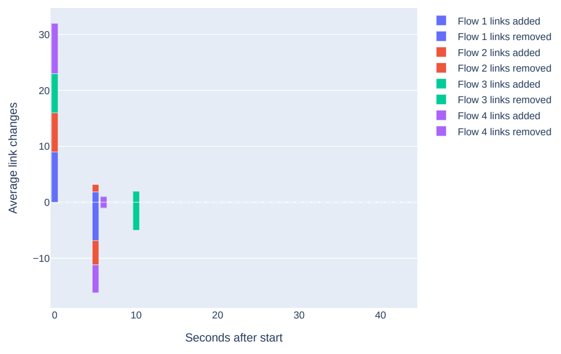


Figure A.40.: Links disappearing and newly appearing in traceroutes when running the Greedy algorithm

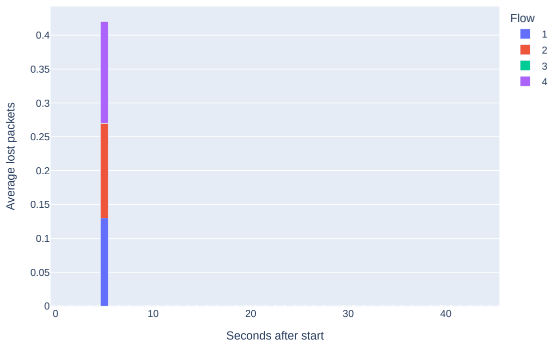


Figure A.41.: Lost packets in our ping test when executing a route change with the Greedy algorithm

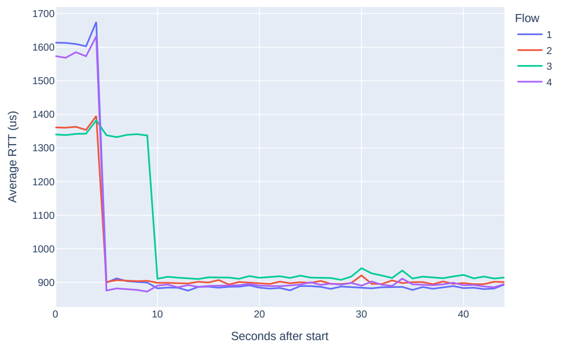


Figure A.42.: Round-Trip-Time between source and destination of all flows during the route changes of the Greedy algorithm

Leaving no time between rounds

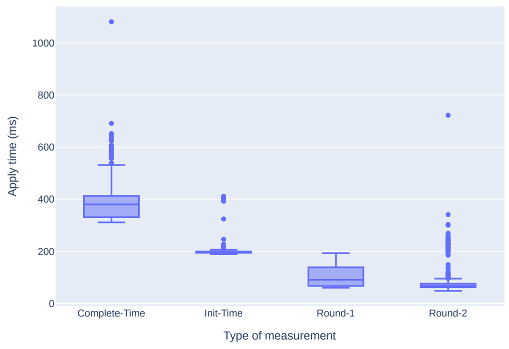


Figure A.43.: Time that it takes to apply the changes using a schedule generated by Greedy

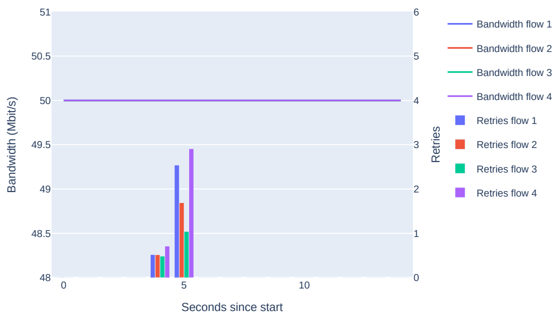


Figure A.44.: Bandwidth and retries during the execution of the Greedy algorithm

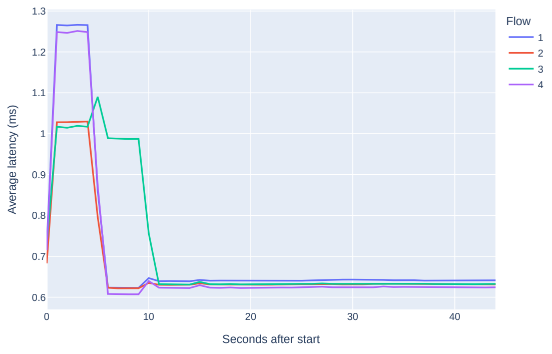


Figure A.45.: Latency between source and destination of all flows during a route change with the Greedy algorithm

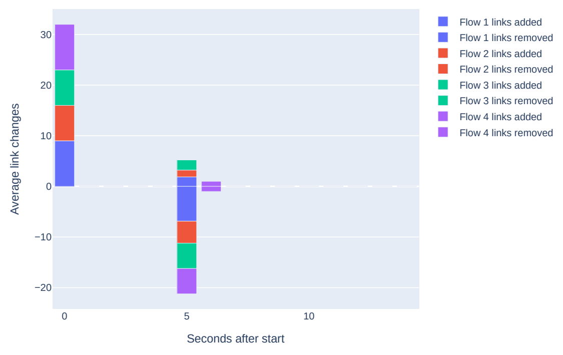


Figure A.46.: Links disappearing and newly appearing in traceroutes when running the Greedy algorithm

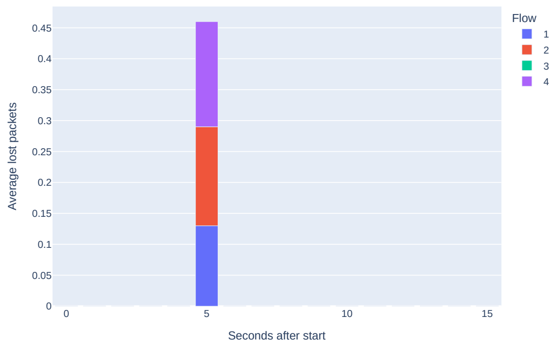


Figure A.47.: Lost packets in our ping test when executing a route change with the Greedy algorithm

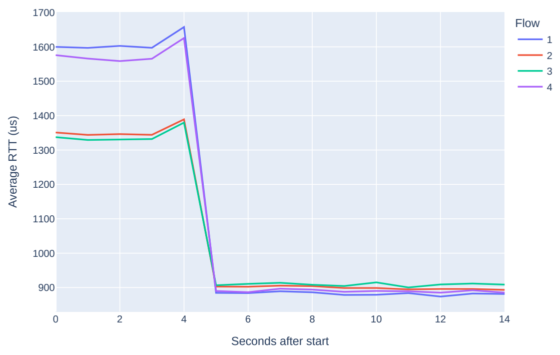


Figure A.48.: Round-Trip-Time between source and destination of all flows during the route changes of the Greedy algorithm

One-Round

Leaving 5 seconds between rounds

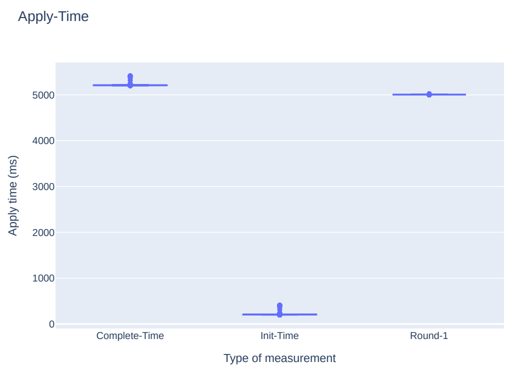


Figure A.49.: Time that it takes to apply the changes using a schedule generated by One-Round

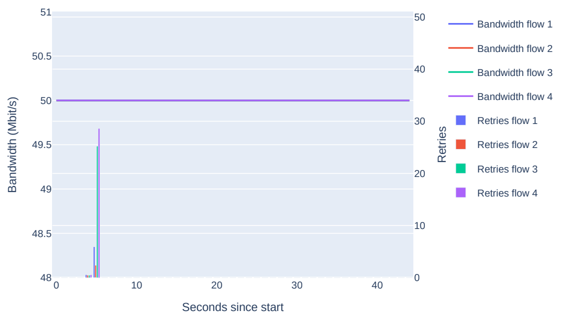


Figure A.50.: Bandwidth and retries during the execution of the One-Round algorithm

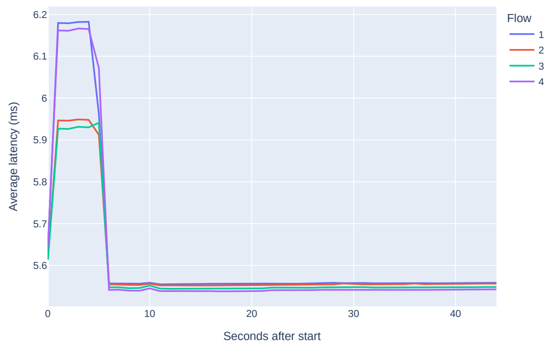


Figure A.51.: Latency between source and destination of all flows during a route change with the One-Round algorithm

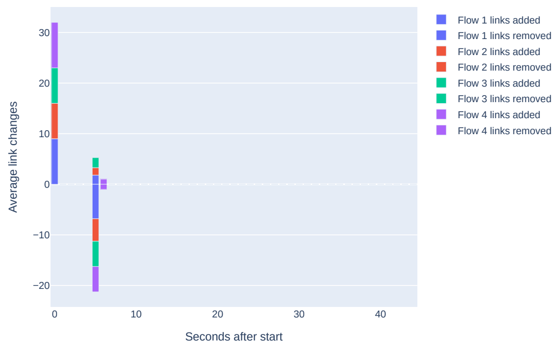


Figure A.52.: Links disappearing and newly appearing in traceroutes when running the One-Round algorithm

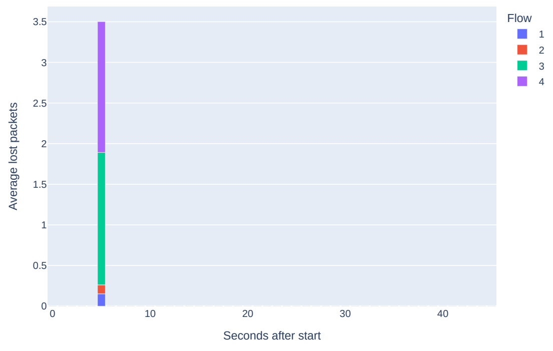


Figure A.53.: Lost packets in our ping test when executing a route change with the One-Round algorithm

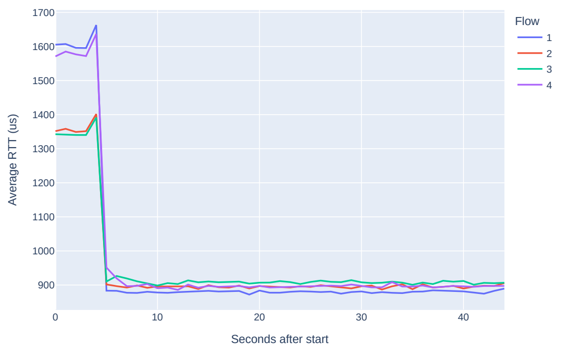


Figure A.54.: Round-Trip-Time between source and destination of all flows during the route changes of the One-Round algorithm

Leaving no time between rounds

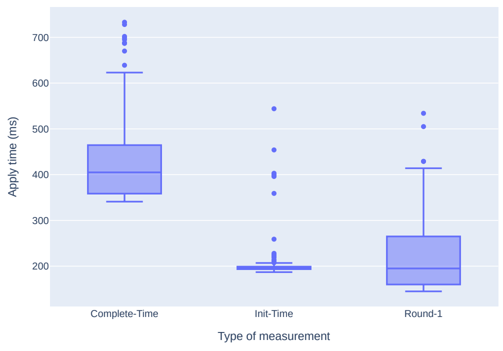


Figure A.55.: Time that it takes to apply the changes using a schedule generated by One-Round

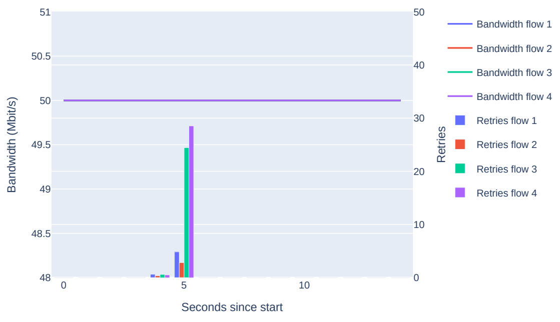


Figure A.56.: Bandwidth and retries during the execution of the One-Round algorithm

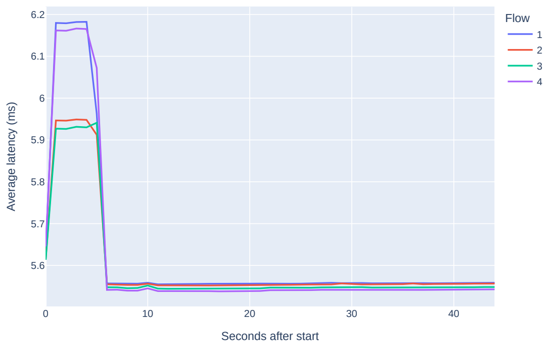


Figure A.57.: Latency between source and destination of all flows during a route change with the One-Round algorithm

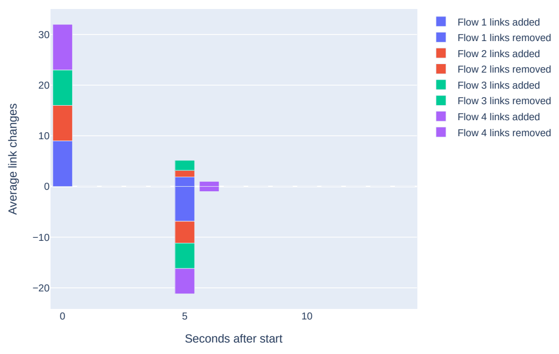


Figure A.58.: Links disappearing and newly appearing in traceroutes when running the One-Round algorithm

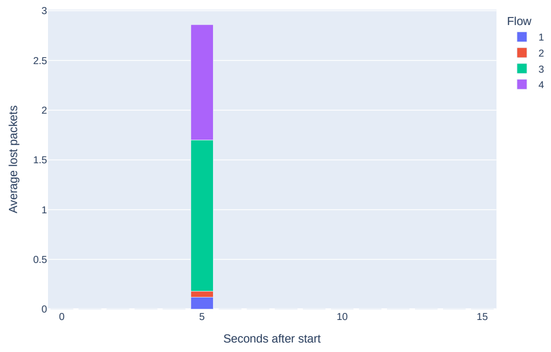


Figure A.59.: Lost packets in our ping test when executing a route change with the One-Round algorithm

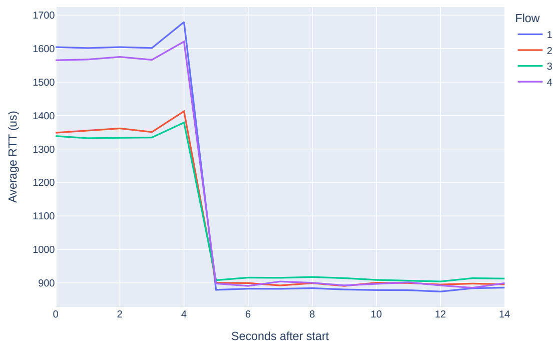


Figure A.60.: Round-Trip-Time between source and destination of all flows during the route changes of the One-Round algorithm

Three-Round

Leaving 5 seconds between rounds

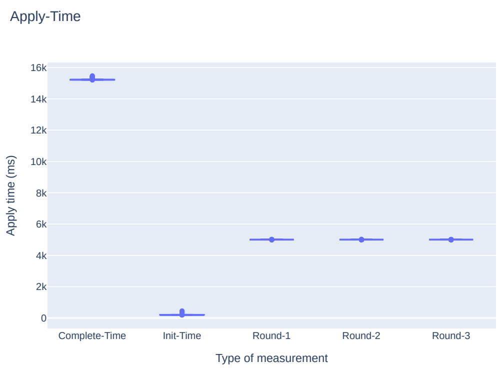


Figure A.61.: Time that it takes to apply the changes using a schedule generated by Three-Round

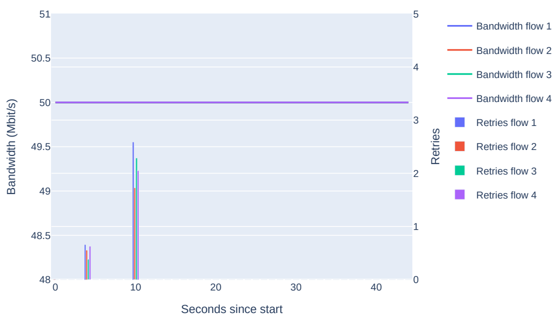


Figure A.62.: Bandwidth and retries during the execution of the Three-Round algorithm

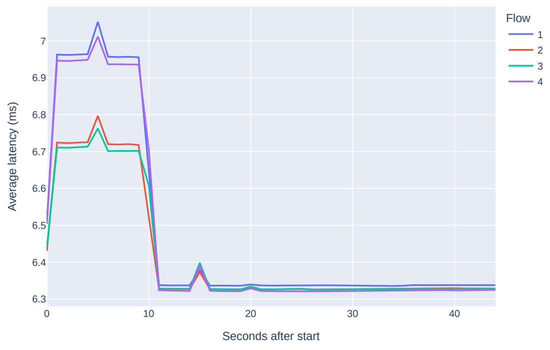


Figure A.63.: Latency between source and destination of all flows during a route change with the Three-Round algorithm

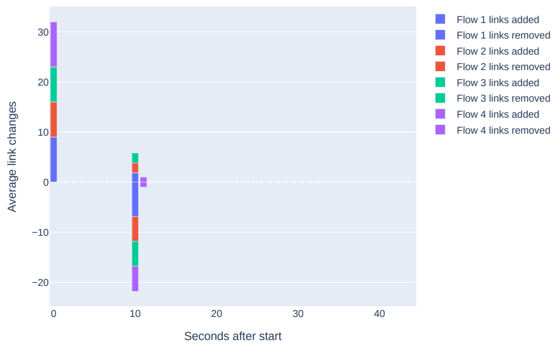


Figure A.64.: Links disappearing and newly appearing in traceroutes when running the Three-Round algorithm

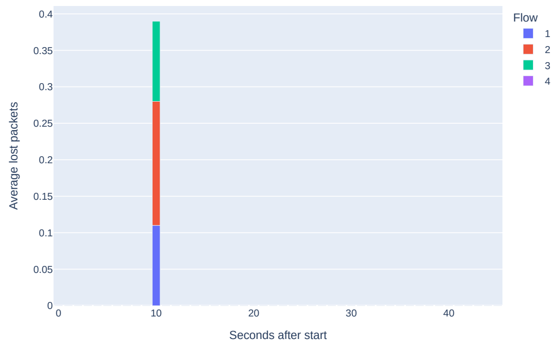


Figure A.65.: Lost packets in our ping test when executing a route change with the Three-Round algorithm

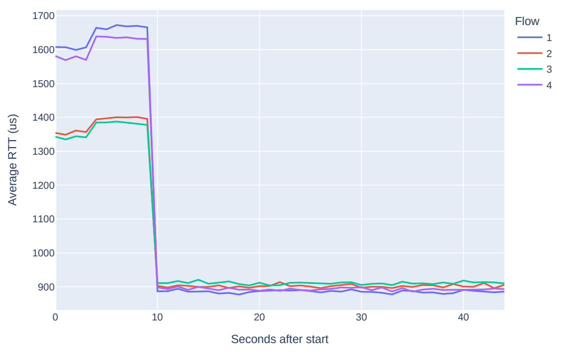


Figure A.66.: Round-Trip-Time between source and destination of all flows during the route changes of the Three-Round algorithm

Leaving no time between rounds

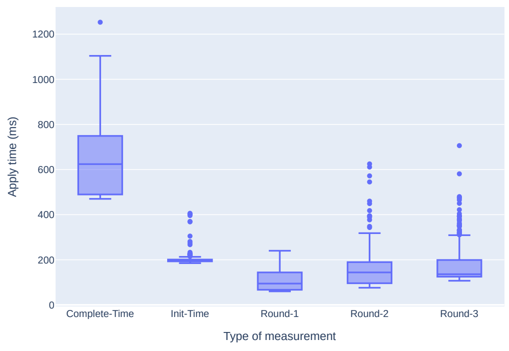


Figure A.67.: Time that it takes to apply the changes using a schedule generated by Three-Round

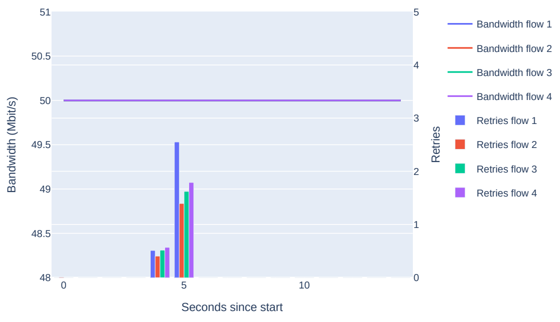


Figure A.68.: Bandwidth and retries during the execution of the Three-Round algorithm

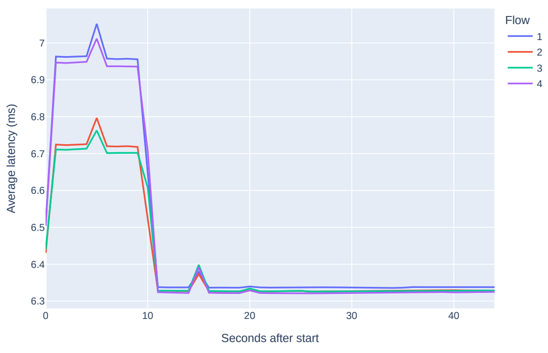


Figure A.69.: Latency between source and destination of all flows during a route change with the Three-Round algorithm

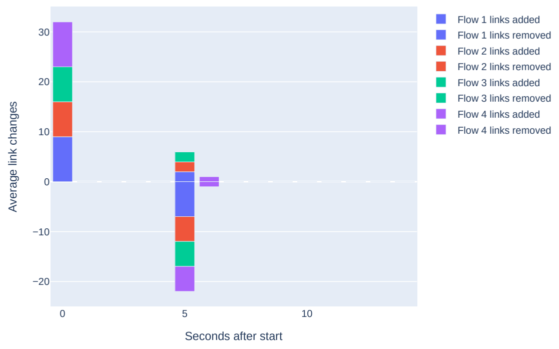


Figure A.70.: Links disappearing and newly appearing in traceroutes when running the Three-Round algorithm

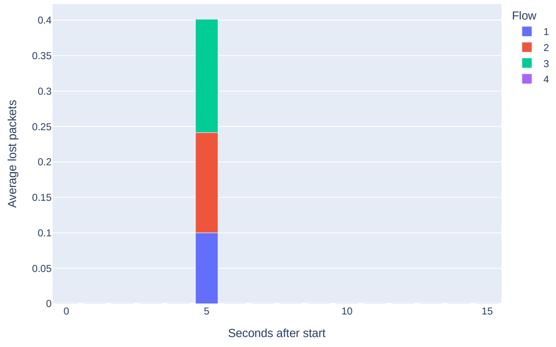


Figure A.71.: Lost packets in our ping test when executing a route change with the Three-Round algorithm

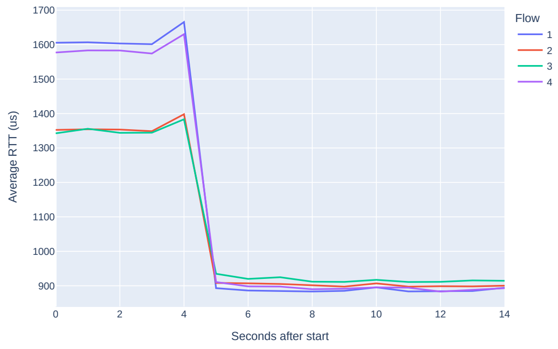


Figure A.72.: Round-Trip-Time between source and destination of all flows during the route changes of the Three-Round algorithm

Results from chapter 7.2

Backward

Leaving 5 seconds between rounds

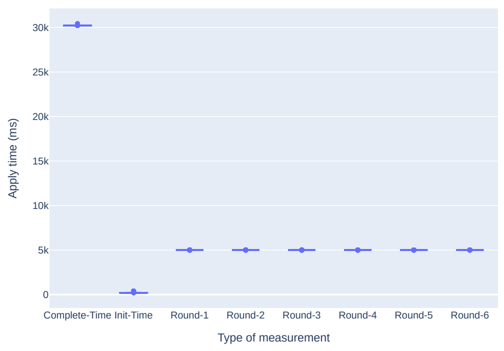


Figure A.73.: Time that it takes to apply the changes using a schedule generated by Backward

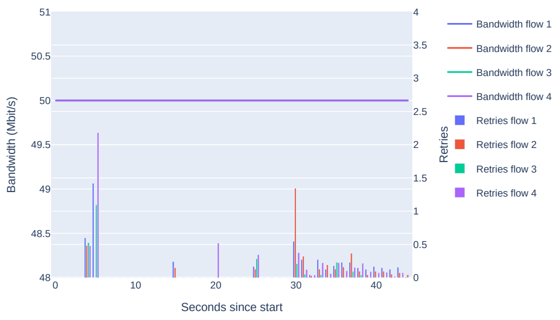


Figure A.74.: Bandwidth and retries during the execution of the Backward algorithm

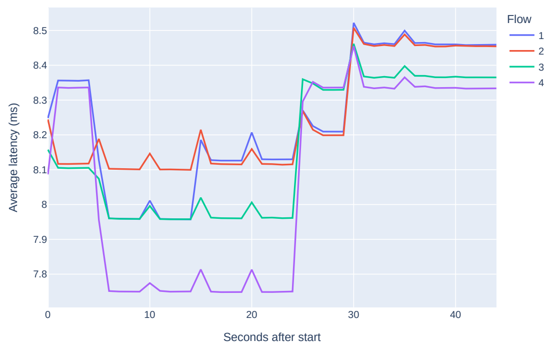


Figure A.75.: Latency between source and destination of all flows during a route change with the Backward algorithm

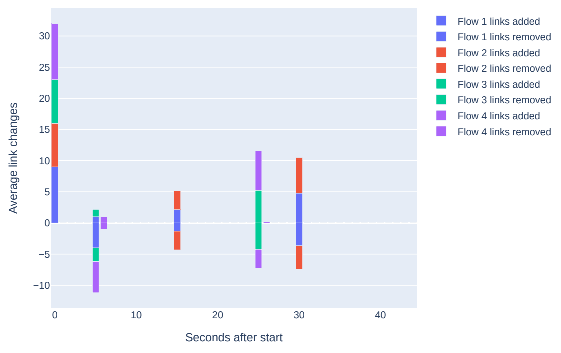


Figure A.76.: Links disappearing and newly appearing in traceroutes when running the Backward algorithm

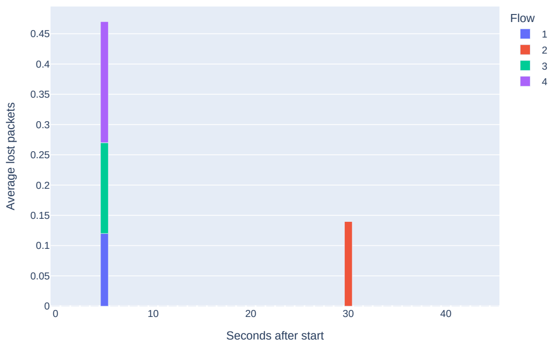


Figure A.77.: Lost packets in our ping test when executing a route change with the Backward algorithm

Leaving no time between rounds

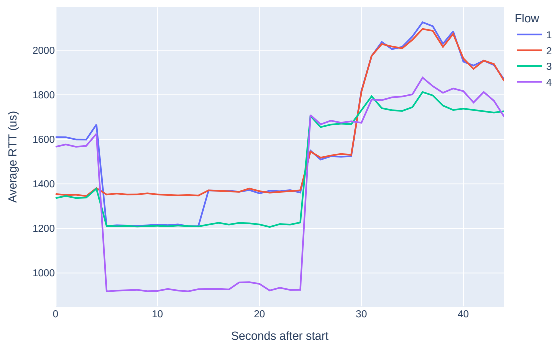


Figure A.78.: Round-Trip-Time between source and destination of all flows during the route changes of the Backward algorithm

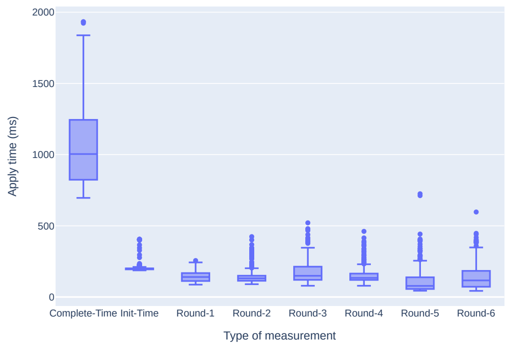


Figure A.79.: Time that it takes to apply the changes using a schedule generated by Backward

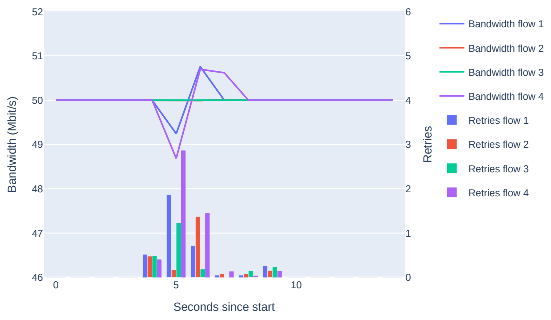


Figure A.80.: Bandwidth and retries during the execution of the Backward algorithm

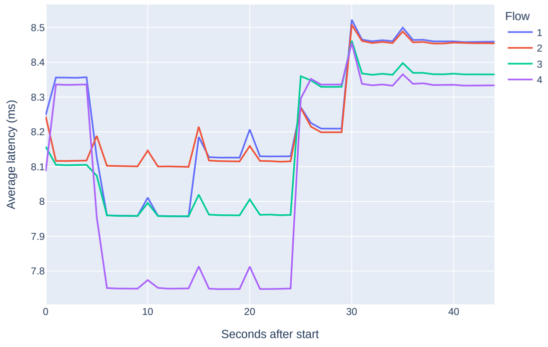


Figure A.81.: Latency between source and destination of all flows during a route change with the Backward algorithm

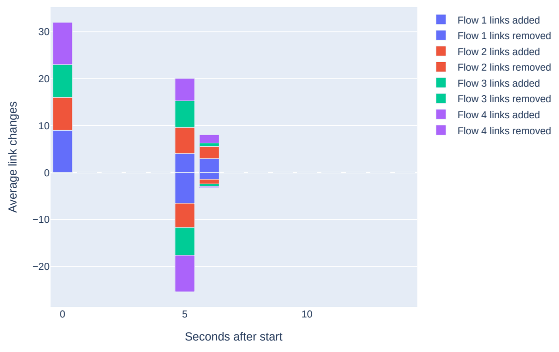


Figure A.82.: Links disappearing and newly appearing in traceroutes when running the Backward algorithm

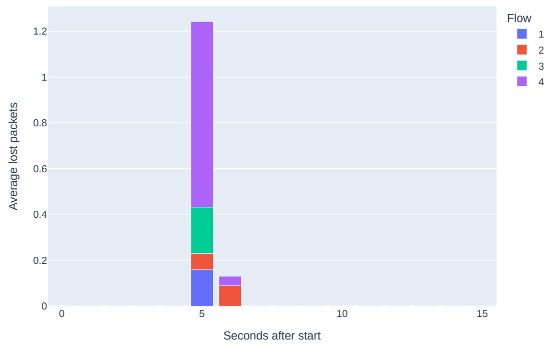


Figure A.83.: Lost packets in our ping test when executing a route change with the Backward algorithm

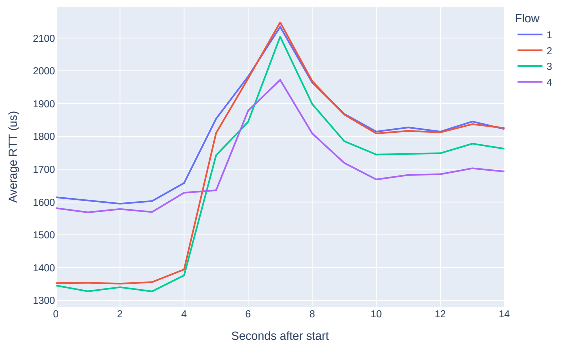


Figure A.84.: Round-Trip-Time between source and destination of all flows during the route changes of the Backward algorithm

Brute-force

Leaving 5 seconds between rounds

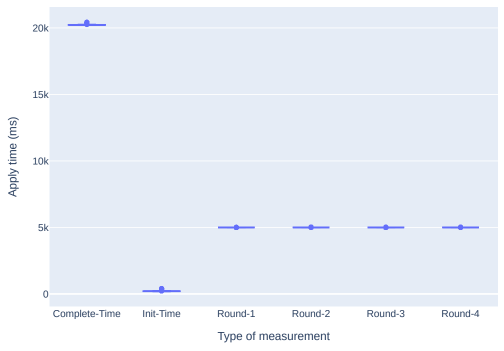


Figure A.85.: Time that it takes to apply the changes using a schedule generated by Brute-force

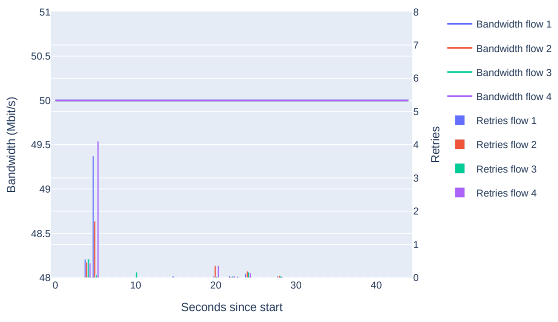


Figure A.86.: Bandwidth and retries during the execution of the Brute-force algorithm

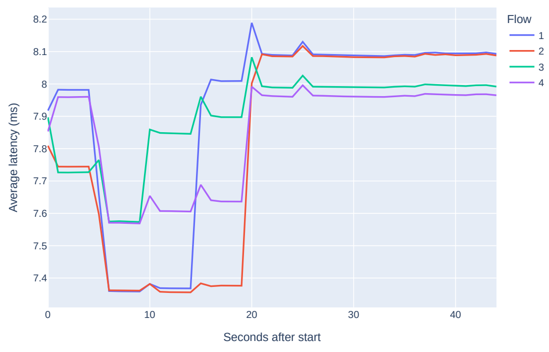


Figure A.87.: Latency between source and destination of all flows during a route change with the Brute-force algorithm

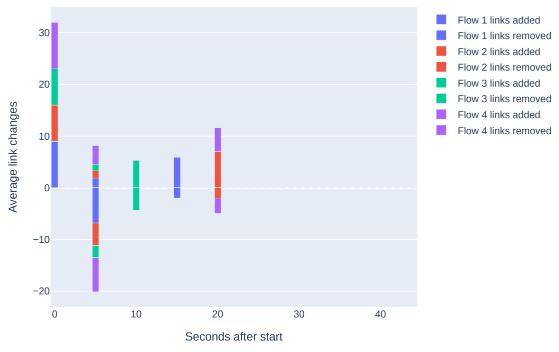


Figure A.88.: Links disappearing and newly appearing in traceroutes when running the Brute-force algorithm

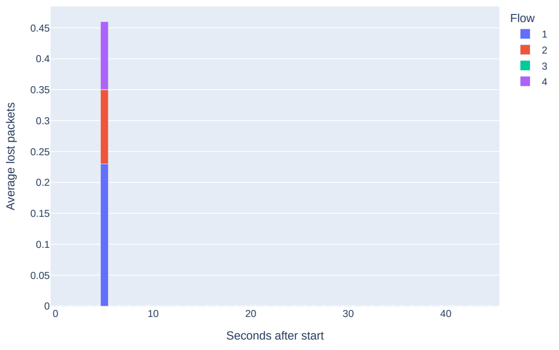


Figure A.89.: Lost packets in our ping test when executing a route change with the Brute-force algorithm

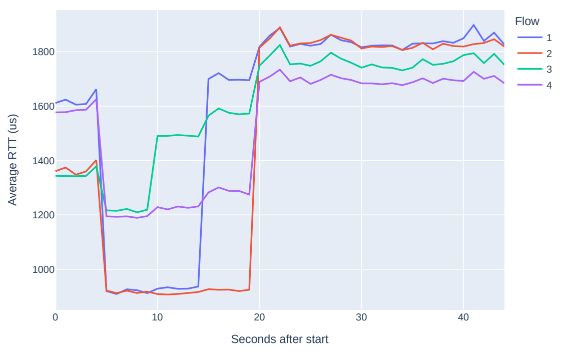


Figure A.90.: Round-Trip-Time between source and destination of all flows during the route changes of the Brute-force algorithm

Leaving no time between rounds

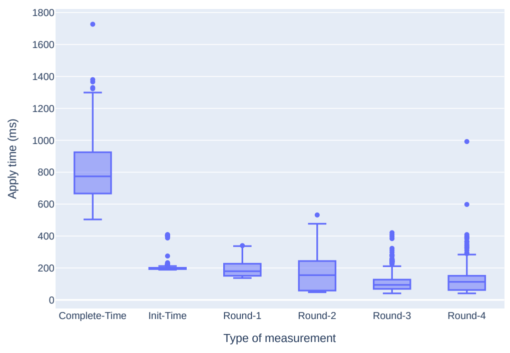


Figure A.91.: Time that it takes to apply the changes using a schedule generated by Brute-force

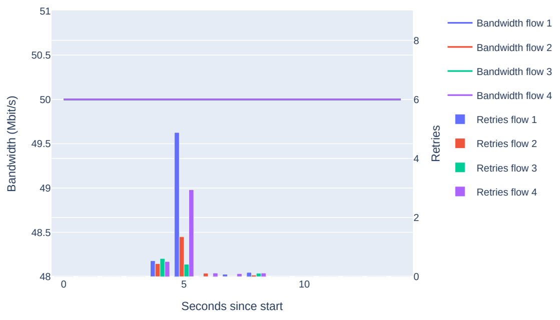


Figure A.92.: Bandwidth and retries during the execution of the Brute-force algorithm

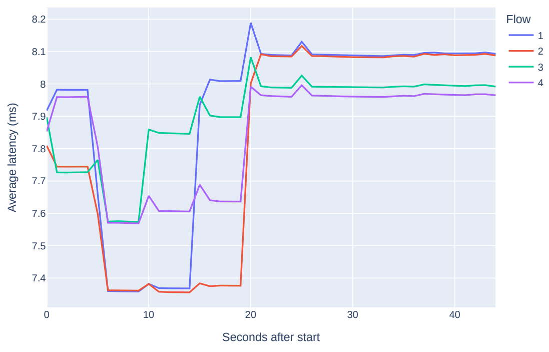


Figure A.93.: Latency between source and destination of all flows during a route change with the Brute-force algorithm

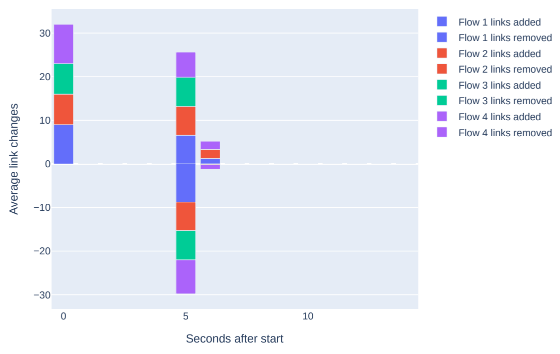


Figure A.94.: Links disappearing and newly appearing in traceroutes when running the Brute-force algorithm

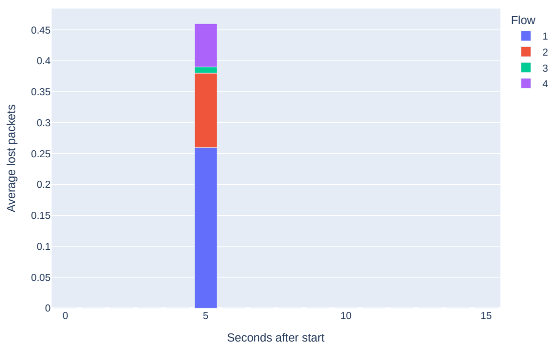


Figure A.95.: Lost packets in our ping test when executing a route change with the Brute-force algorithm

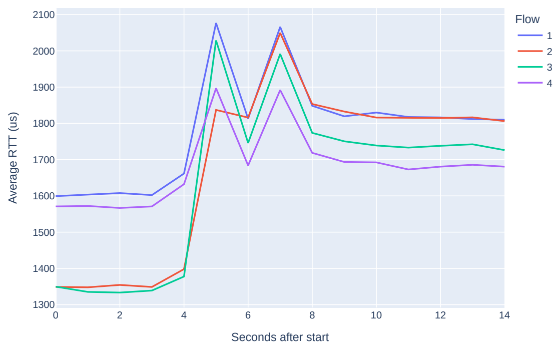


Figure A.96.: Round-Trip-Time between source and destination of all flows during the route changes of the Brute-force algorithm

Chronicle

Leaving 5 seconds between rounds

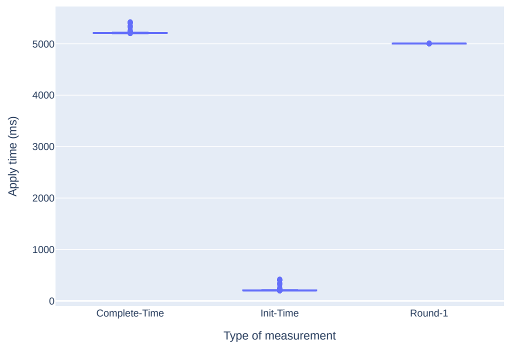


Figure A.97.: Time that it takes to apply the changes using a schedule generated by Chronicle

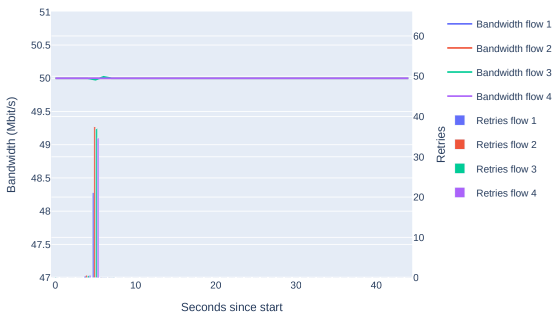


Figure A.98.: Bandwidth and retries during the execution of the Chronicle algorithm

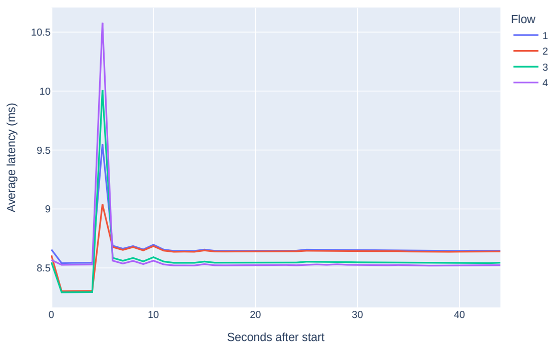


Figure A.99.: Latency between source and destination of all flows during a route change with the Chronicle algorithm

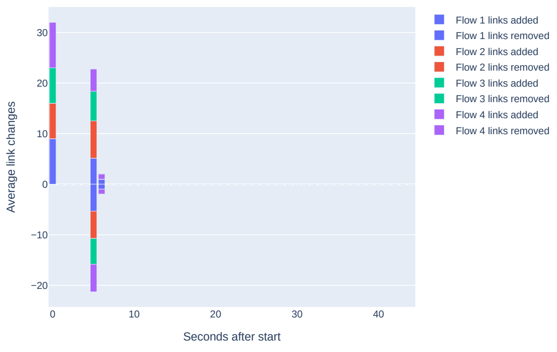


Figure A.100.: Links disappearing and newly appearing in traceroutes when running the Chronicle algorithm

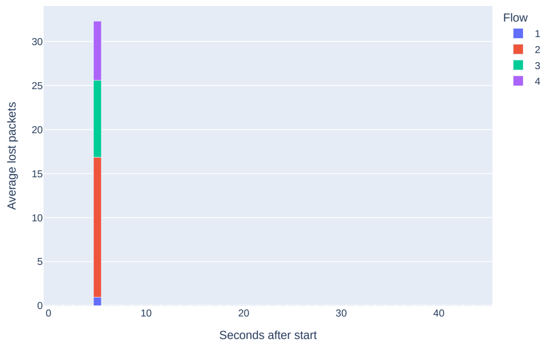


Figure A.101.: Lost packets in our ping test when executing a route change with the Chronicle algorithm

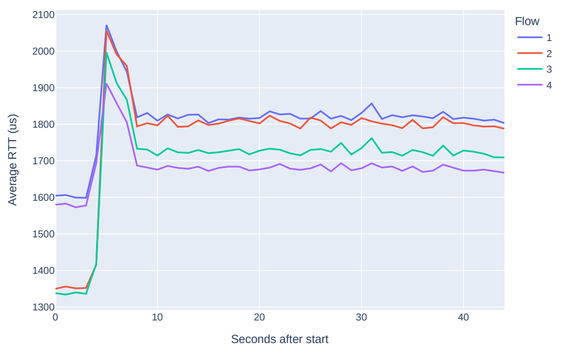


Figure A.102.: Round-Trip-Time between source and destination of all flows during the route changes of the Chronicle algorithm

Leaving no time between rounds

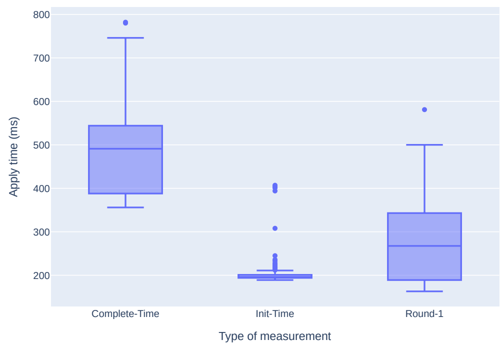


Figure A.103.: Time that it takes to apply the changes using a schedule generated by Chronicle

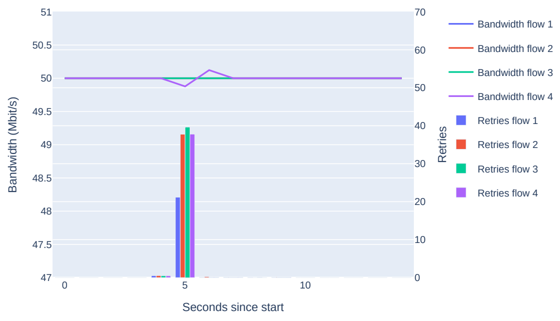


Figure A.104.: Bandwidth and retries during the execution of the Chronicle algorithm

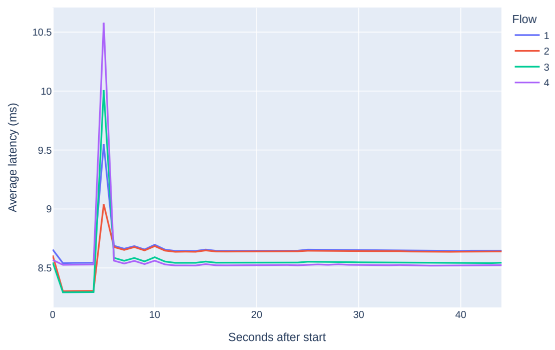


Figure A.105.: Latency between source and destination of all flows during a route change with the Chronicle algorithm

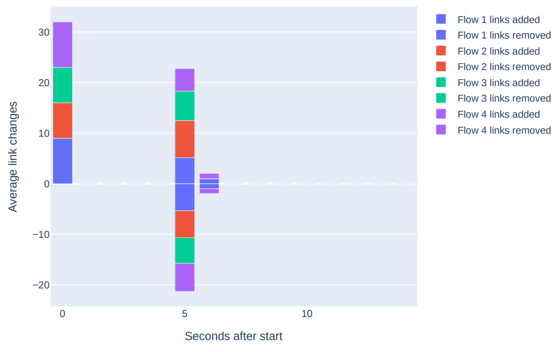


Figure A.106.: Links disappearing and newly appearing in traceroutes when running the Chronicle algorithm

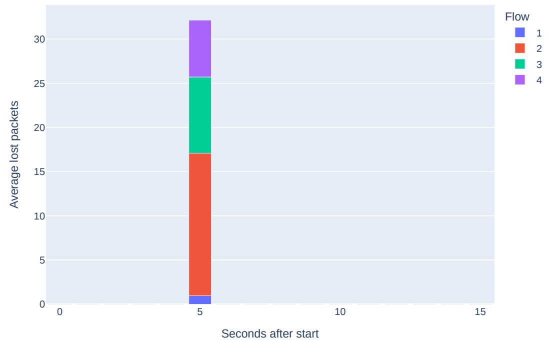


Figure A.107.: Lost packets in our ping test when executing a route change with the Chronicle algorithm

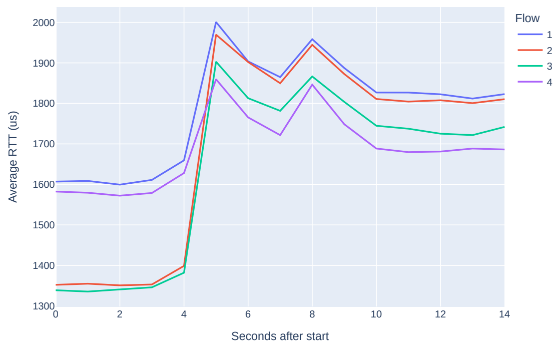


Figure A.108.: Round-Trip-Time between source and destination of all flows during the route changes of the Chronicle algorithm

Greedy

Leaving 5 seconds between rounds

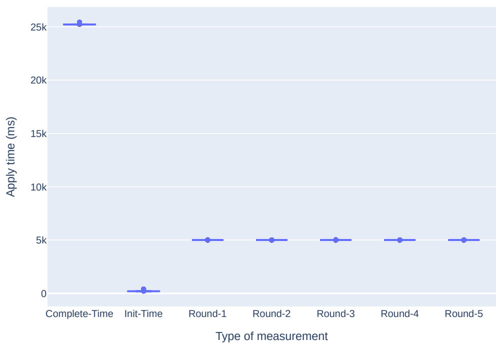


Figure A.109.: Time that it takes to apply the changes using a schedule generated by Greedy

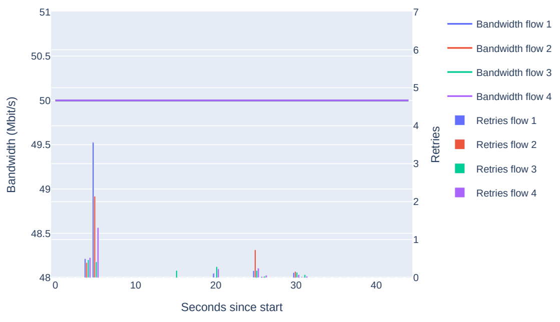


Figure A.110.: Bandwidth and retries during the execution of the Greedy algorithm

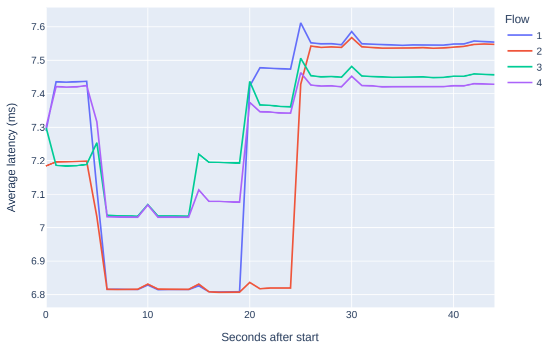


Figure A.111.: Latency between source and destination of all flows during a route change with the Greedy algorithm

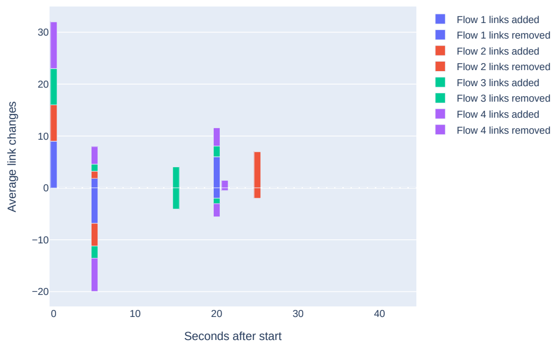


Figure A.112.: Links disappearing and newly appearing in traceroutes when running the Greedy algorithm

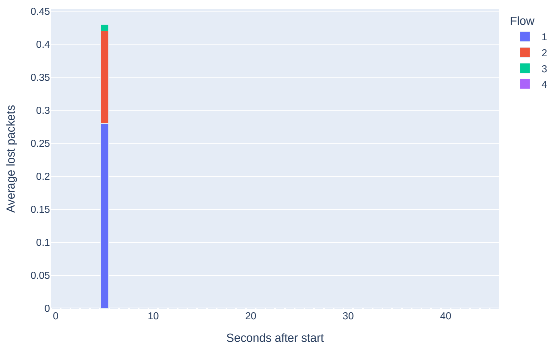


Figure A.113.: Lost packets in our ping test when executing a route change with the Greedy algorithm

Leaving no time between rounds

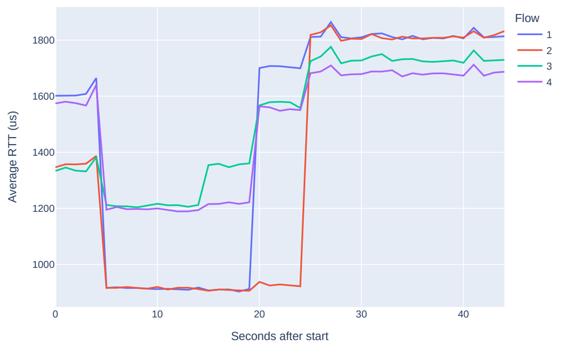


Figure A.114.: Round-Trip-Time between source and destination of all flows during the route changes of the Greedy algorithm

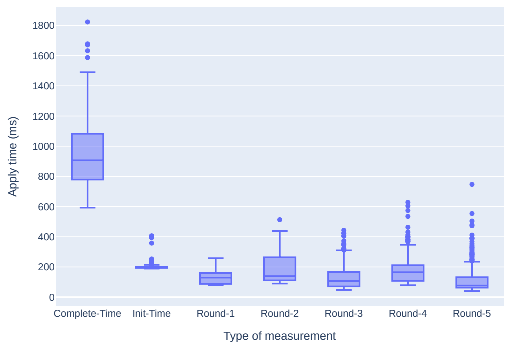


Figure A.115.: Time that it takes to apply the changes using a schedule generated by Greedy

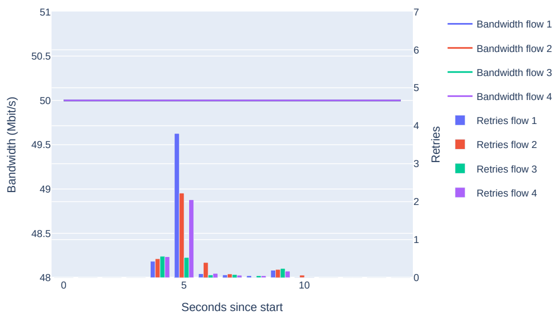


Figure A.116.: Bandwidth and retries during the execution of the Greedy algorithm

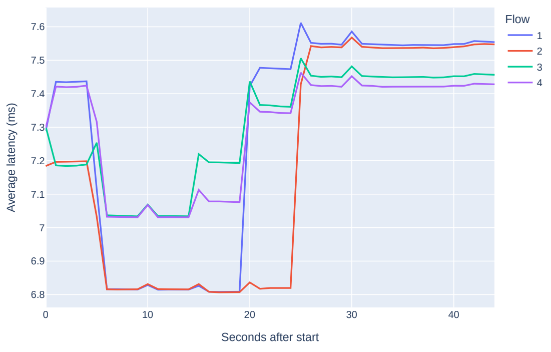


Figure A.117.: Latency between source and destination of all flows during a route change with the Greedy algorithm

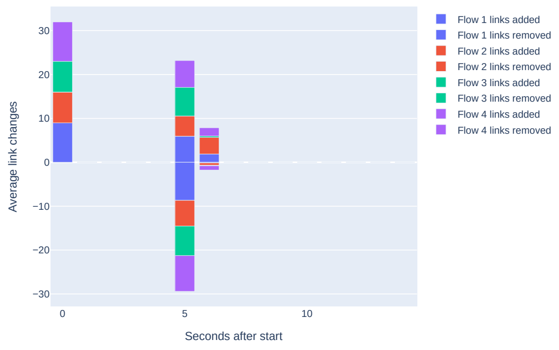


Figure A.118.: Links disappearing and newly appearing in traceroutes when running the Greedy algorithm

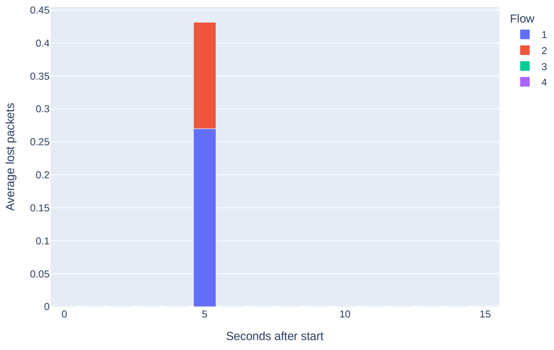


Figure A.119.: Lost packets in our ping test when executing a route change with the Greedy algorithm

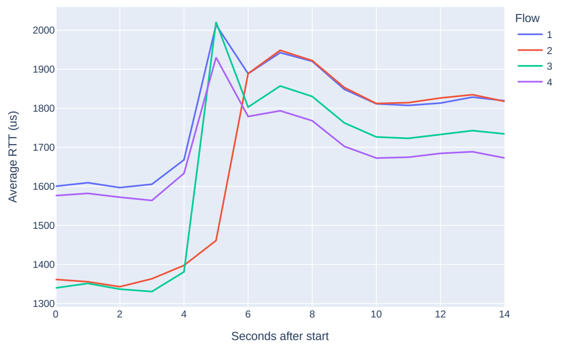


Figure A.120.: Round-Trip-Time between source and destination of all flows during the route changes of the Greedy algorithm

One-Round

Leaving 5 seconds between rounds

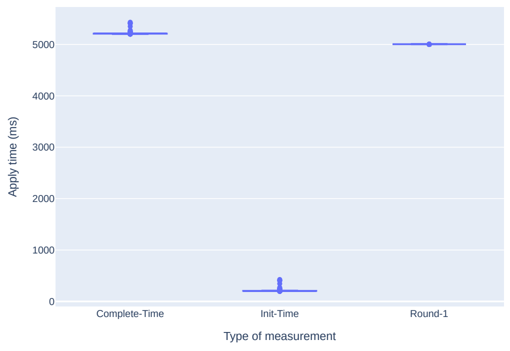


Figure A.121.: Time that it takes to apply the changes using a schedule generated by One-Round

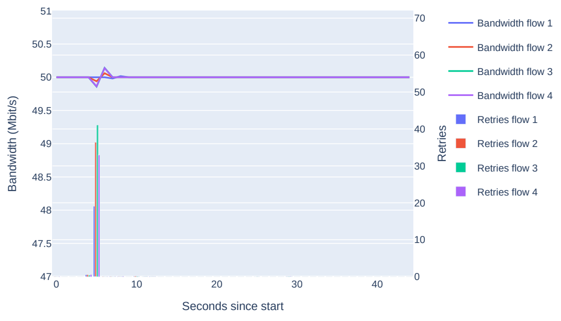


Figure A.122.: Bandwidth and retries during the execution of the One-Round algorithm

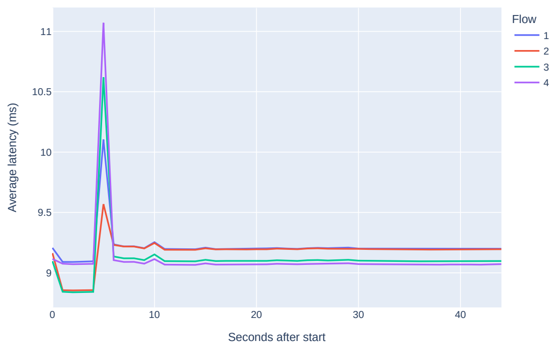


Figure A.123.: Latency between source and destination of all flows during a route change with the One-Round algorithm

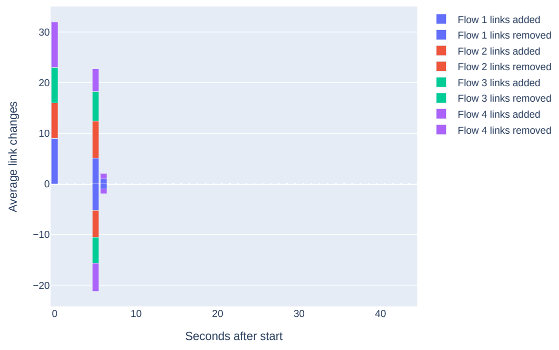


Figure A.124.: Links disappearing and newly appearing in traceroutes when running the One-Round algorithm

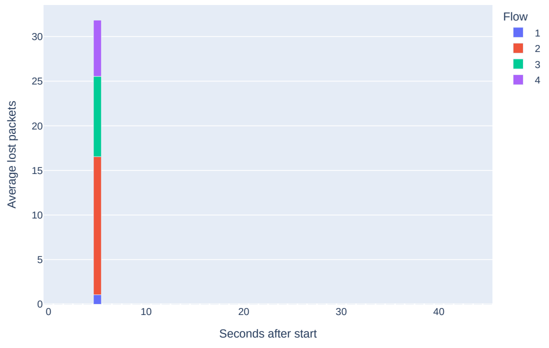


Figure A.125.: Lost packets in our ping test when executing a route change with the One-Round algorithm

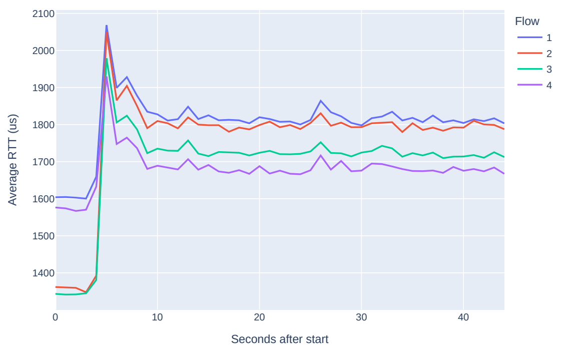


Figure A.126.: Round-Trip-Time between source and destination of all flows during the route changes of the One-Round algorithm

Leaving no time between rounds

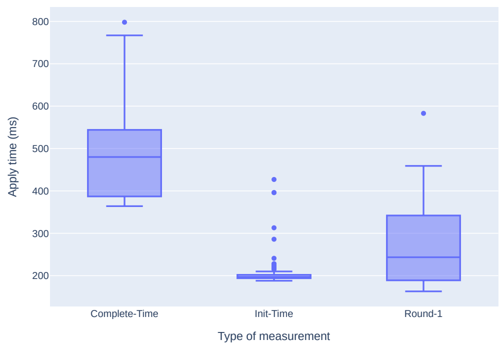


Figure A.127.: Time that it takes to apply the changes using a schedule generated by One-Round

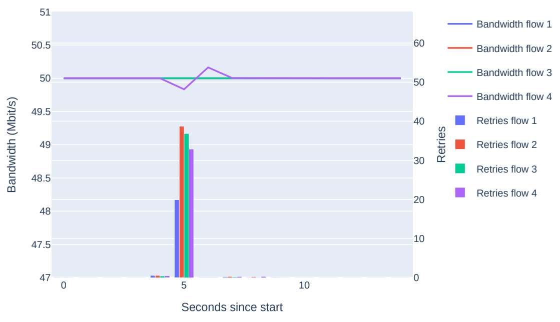


Figure A.128.: Bandwidth and retries during the execution of the One-Round algorithm

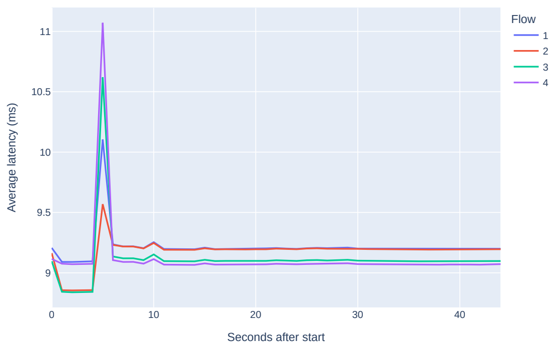


Figure A.129.: Latency between source and destination of all flows during a route change with the One-Round algorithm

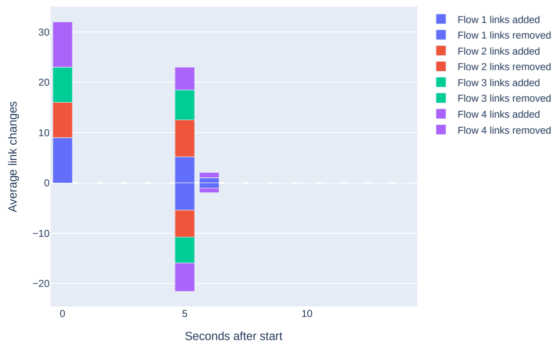


Figure A.130.: Links disappearing and newly appearing in traceroutes when running the One-Round algorithm

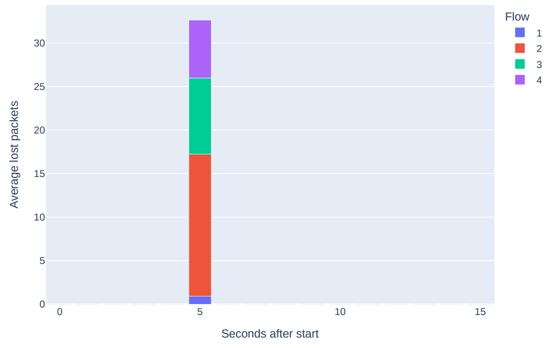


Figure A.131.: Lost packets in our ping test when executing a route change with the One-Round algorithm

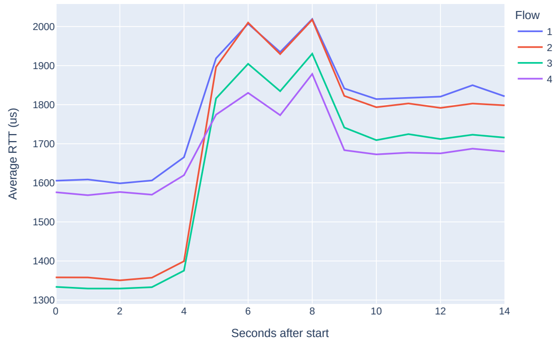


Figure A.132.: Round-Trip-Time between source and destination of all flows during the route changes of the One-Round algorithm

Three-Round

Leaving 5 seconds between rounds

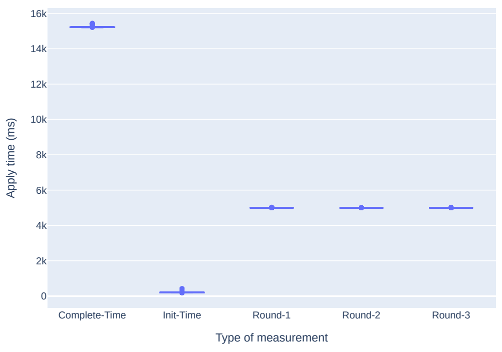


Figure A.133.: Time that it takes to apply the changes using a schedule generated by Three-Round

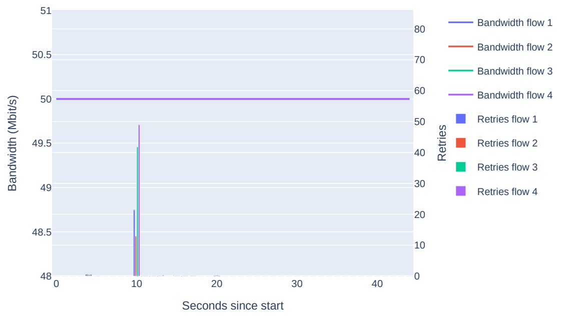


Figure A.134.: Bandwidth and retries during the execution of the Three-Round algorithm

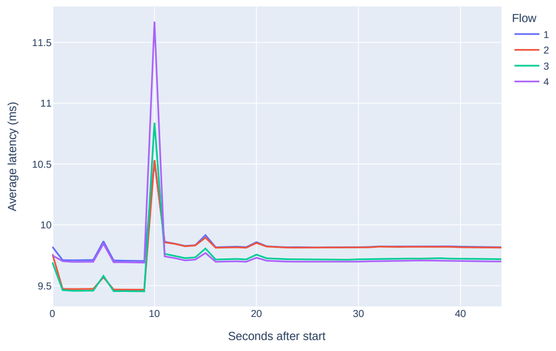


Figure A.135.: Latency between source and destination of all flows during a route change with the Three-Round algorithm

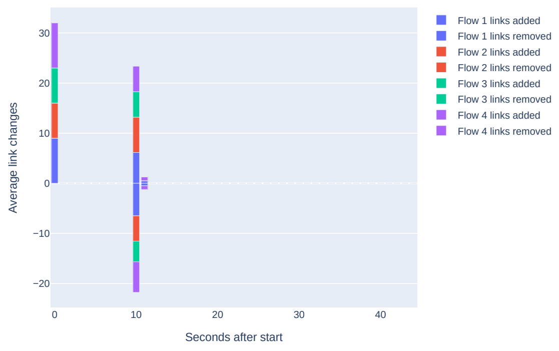


Figure A.136.: Links disappearing and newly appearing in traceroutes when running the Three-Round algorithm

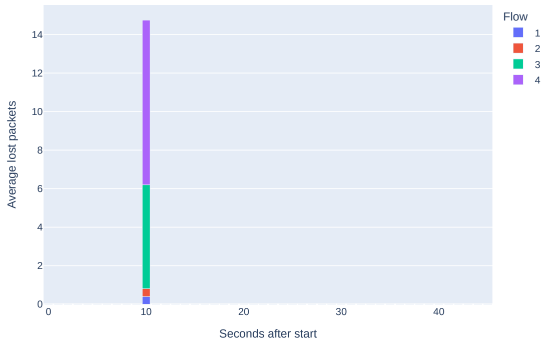


Figure A.137.: Lost packets in our ping test when executing a route change with the Three-Round algorithm

Leaving no time between rounds

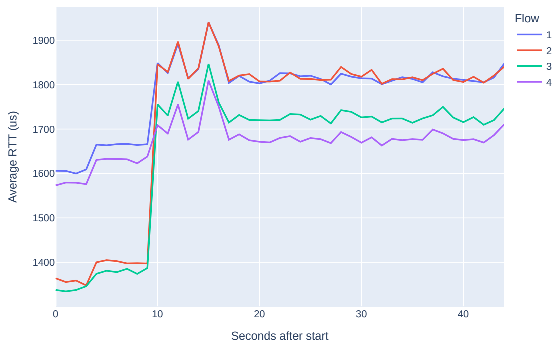


Figure A.138.: Round-Trip-Time between source and destination of all flows during the route changes of the Three-Round algorithm

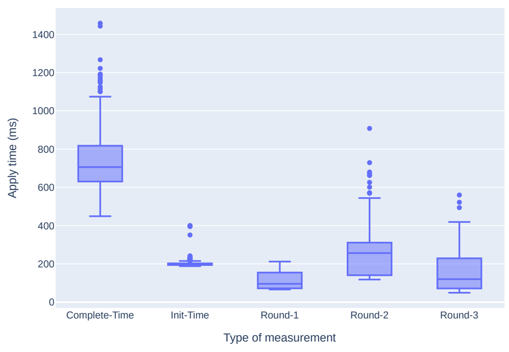


Figure A.139.: Time that it takes to apply the changes using a schedule generated by Three-Round

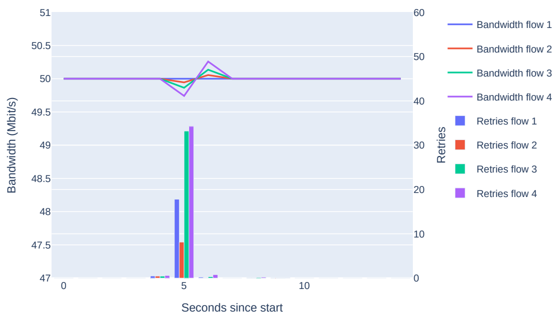


Figure A.140.: Bandwidth and retries during the execution of the Three-Round algorithm

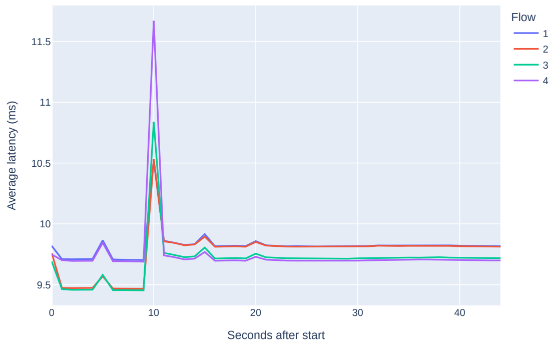


Figure A.141.: Latency between source and destination of all flows during a route change with the Three-Round algorithm

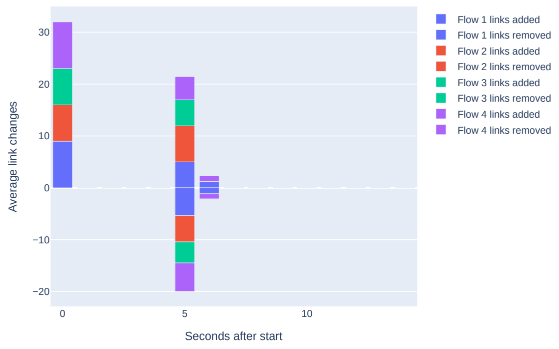


Figure A.142.: Links disappearing and newly appearing in traceroutes when running the Three-Round algorithm

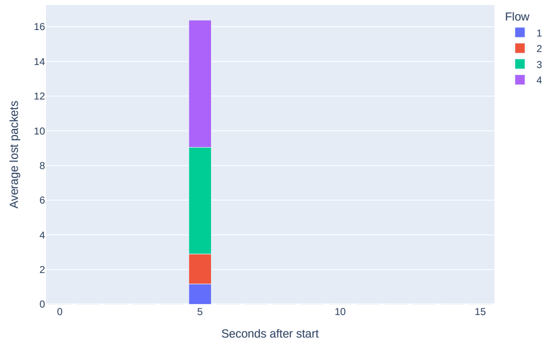


Figure A.143.: Lost packets in our ping test when executing a route change with the Three-Round algorithm

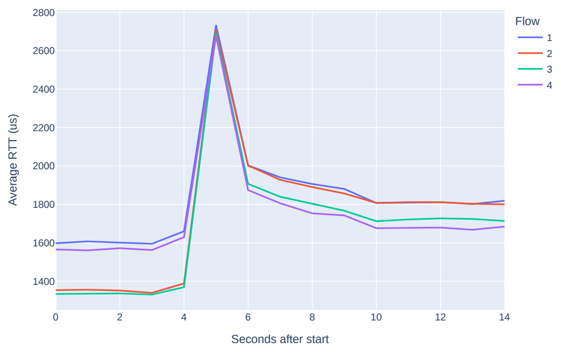


Figure A.144.: Round-Trip-Time between source and destination of all flows during the route changes of the Three-Round algorithm

Results from chapter 7.3

Backward

Leaving 5 seconds between rounds

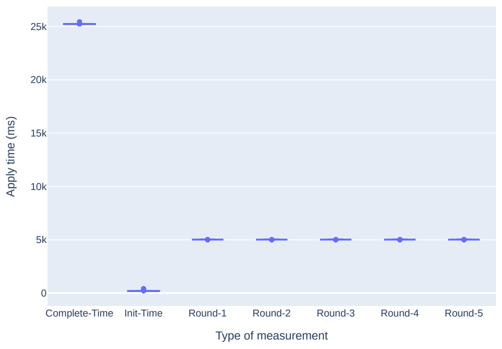


Figure A.145.: Time that it takes to apply the changes using a schedule generated by Backward

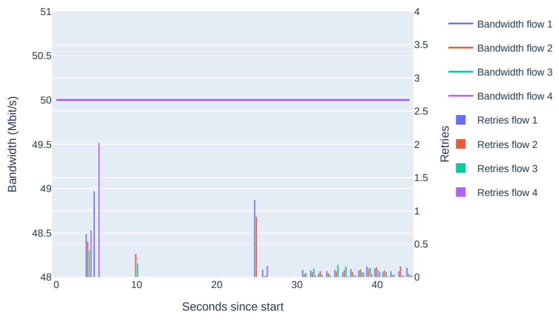


Figure A.146.: Bandwidth and retries during the execution of the Backward algorithm

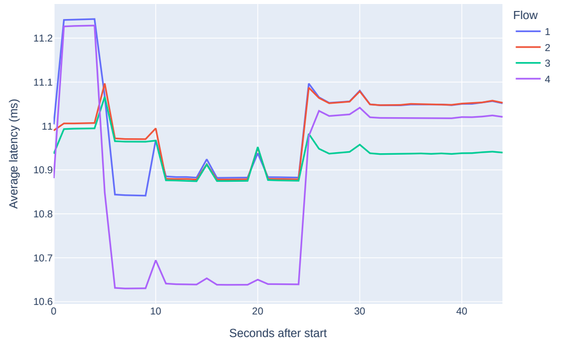


Figure A.147.: Latency between source and destination of all flows during a route change with the Backward algorithm

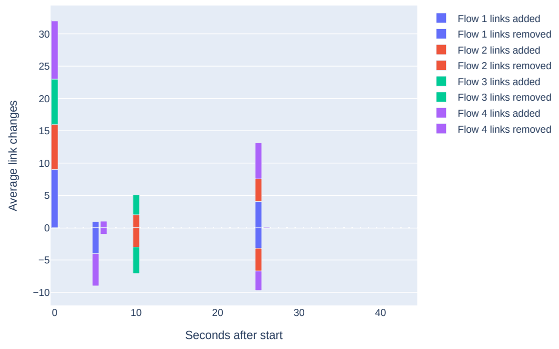


Figure A.148.: Links disappearing and newly appearing in traceroutes when running the Backward algorithm

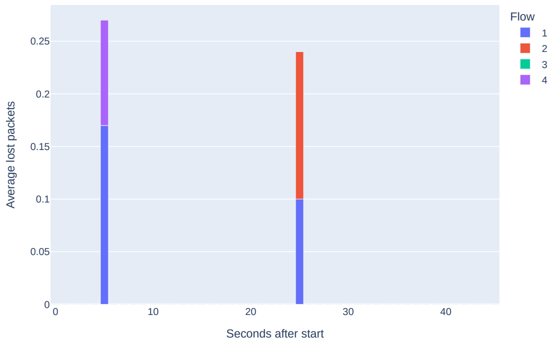


Figure A.149.: Lost packets in our ping test when executing a route change with the Backward algorithm

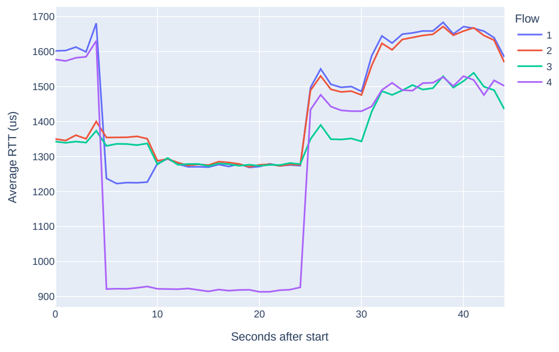


Figure A.150.: Round-Trip-Time between source and destination of all flows during the route changes of the Backward algorithm

Leaving no time between rounds

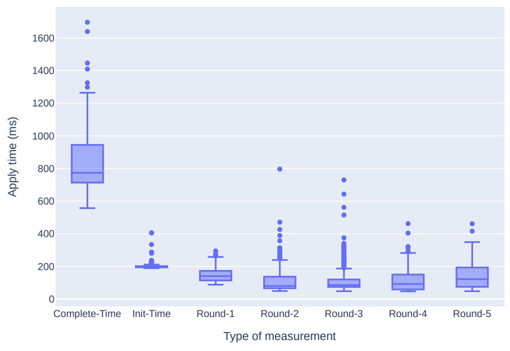


Figure A.151.: Time that it takes to apply the changes using a schedule generated by Backward

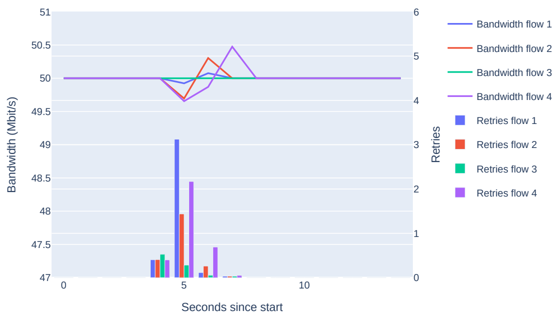


Figure A.152.: Bandwidth and retries during the execution of the Backward algorithm

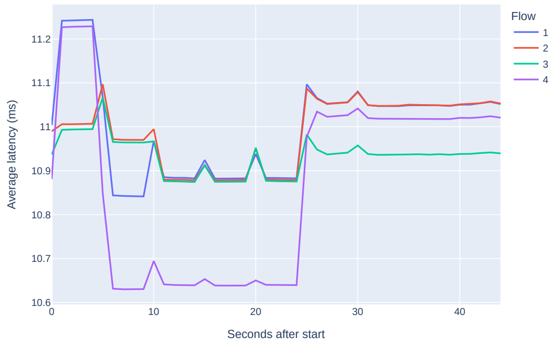


Figure A.153.: Latency between source and destination of all flows during a route change with the Backward algorithm

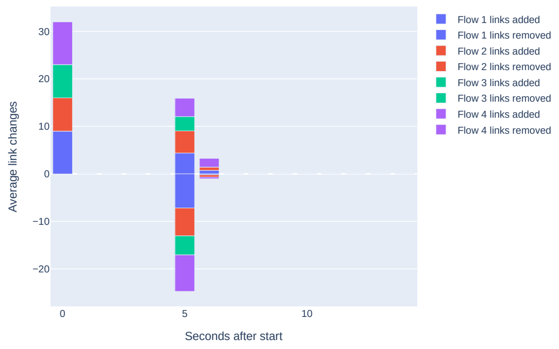


Figure A.154.: Links disappearing and newly appearing in traceroutes when running the Backward algorithm

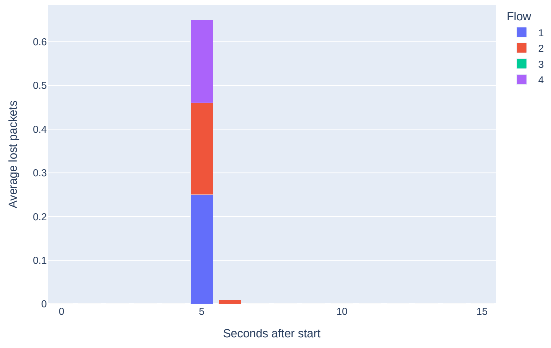


Figure A.155.: Lost packets in our ping test when executing a route change with the Backward algorithm

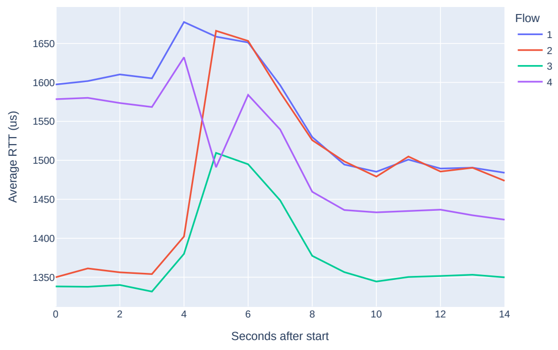


Figure A.156.: Round-Trip-Time between source and destination of all flows during the route changes of the Backward algorithm

Brute-force

Leaving 5 seconds between rounds

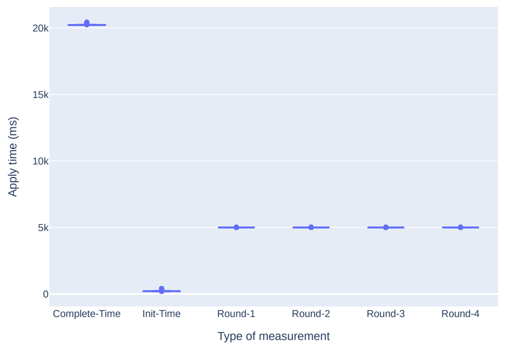


Figure A.157.: Time that it takes to apply the changes using a schedule generated by Brute-force

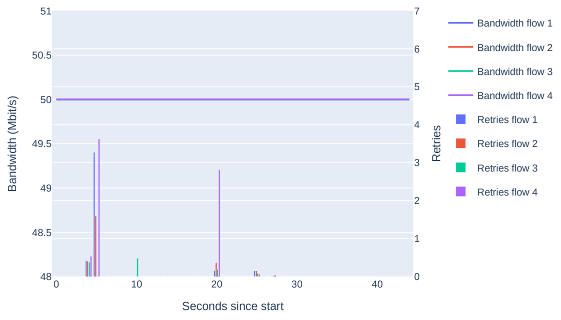


Figure A.158.: Bandwidth and retries during the execution of the Brute-force algorithm

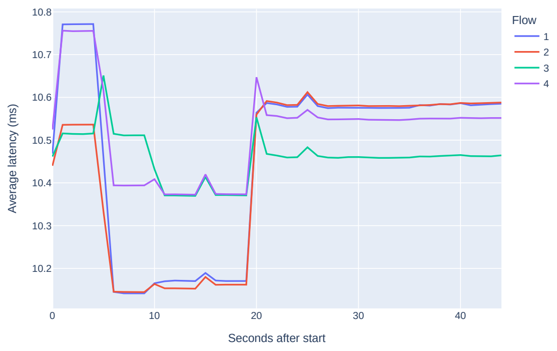


Figure A.159.: Latency between source and destination of all flows during a route change with the Brute-force algorithm

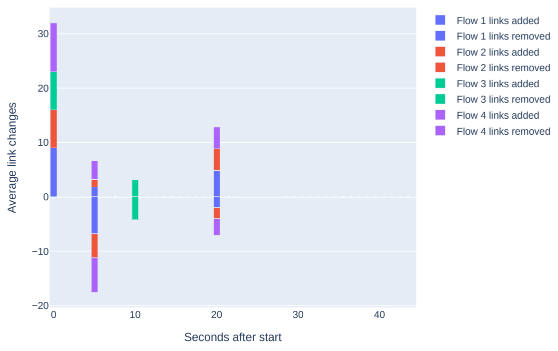


Figure A.160.: Links disappearing and newly appearing in traceroutes when running the Brute-force algorithm

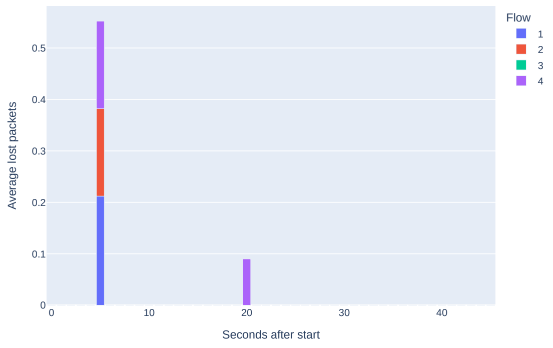


Figure A.161.: Lost packets in our ping test when executing a route change with the Brute-force algorithm

Leaving no time between rounds

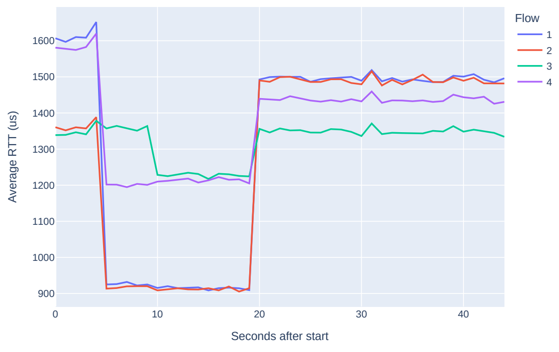


Figure A.162.: Round-Trip-Time between source and destination of all flows during the route changes of the Brute-force algorithm

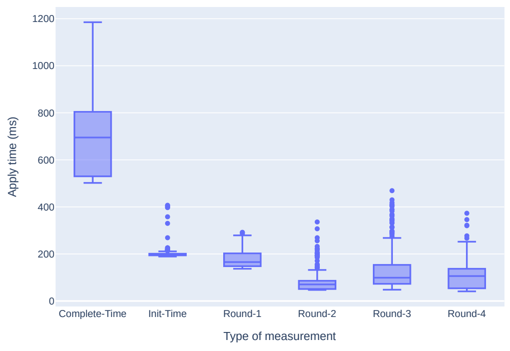


Figure A.163.: Time that it takes to apply the changes using a schedule generated by Brute-force

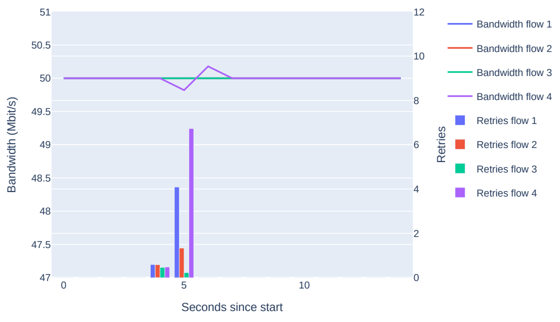


Figure A.164.: Bandwidth and retries during the execution of the Brute-force algorithm

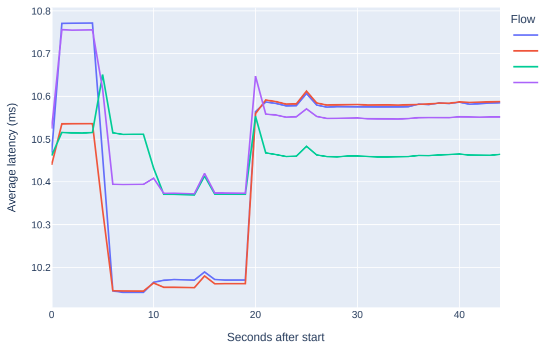


Figure A.165.: Latency between source and destination of all flows during a route change with the Brute-force algorithm

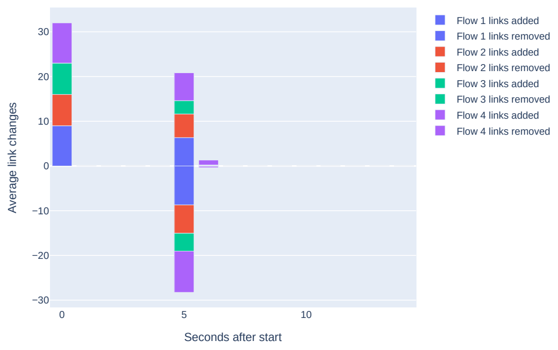


Figure A.166.: Links disappearing and newly appearing in traceroutes when running the Brute-force algorithm

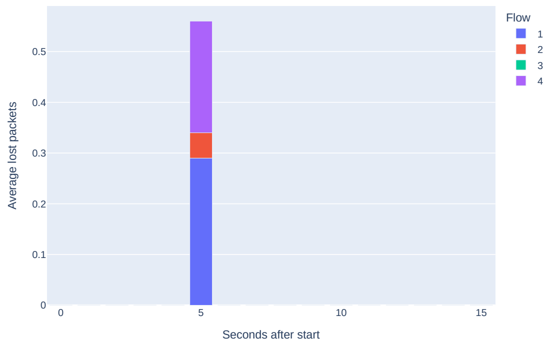


Figure A.167.: Lost packets in our ping test when executing a route change with the Brute-force algorithm

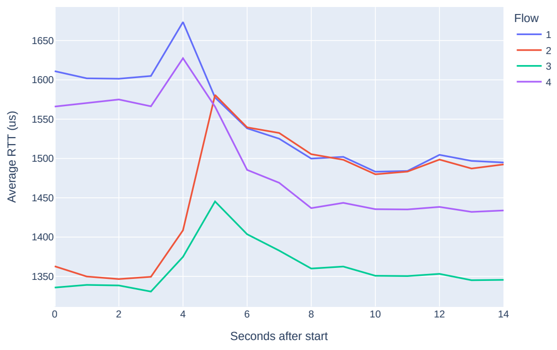


Figure A.168.: Round-Trip-Time between source and destination of all flows during the route changes of the Brute-force algorithm

Chronicle

Leaving 5 seconds between rounds

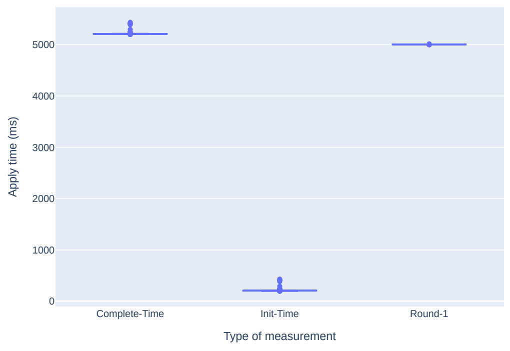


Figure A.169.: Time that it takes to apply the changes using a schedule generated by Chronicle

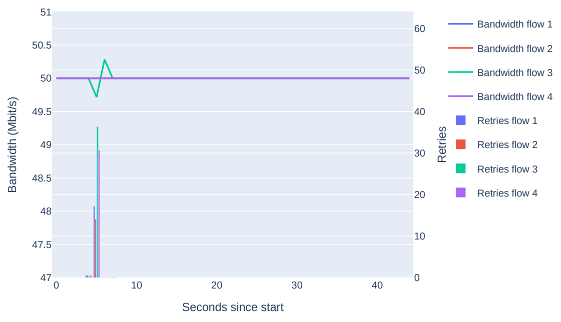


Figure A.170.: Bandwidth and retries during the execution of the Chronicle algorithm

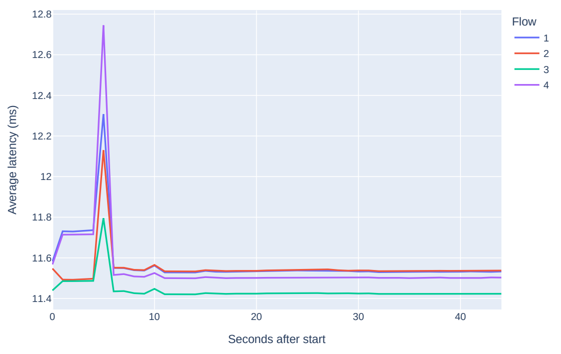


Figure A.171.: Latency between source and destination of all flows during a route change with the Chronicle algorithm

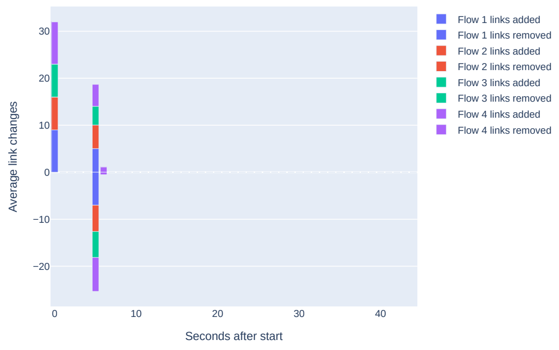


Figure A.172.: Links disappearing and newly appearing in traceroutes when running the Chronicle algorithm

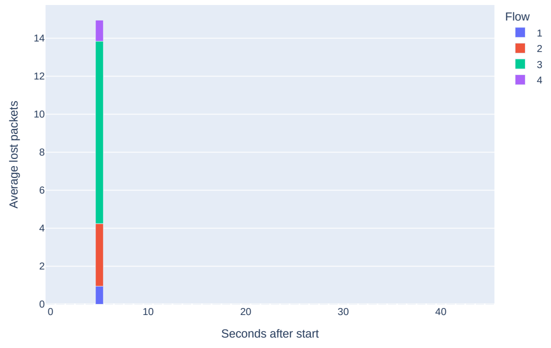


Figure A.173.: Lost packets in our ping test when executing a route change with the Chronicle algorithm

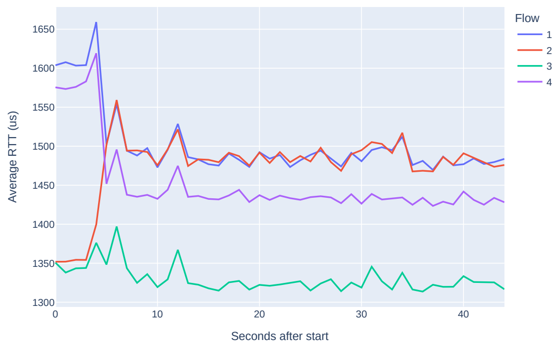


Figure A.174.: Round-Trip-Time between source and destination of all flows during the route changes of the Chronicle algorithm

Leaving no time between rounds

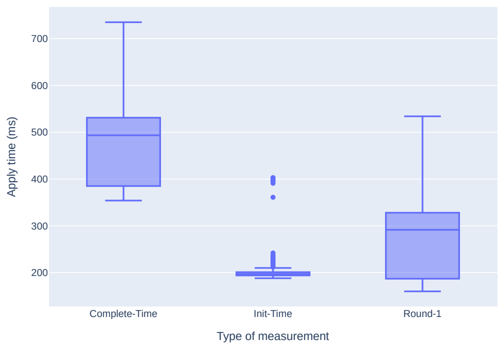


Figure A.175.: Time that it takes to apply the changes using a schedule generated by Chronicle

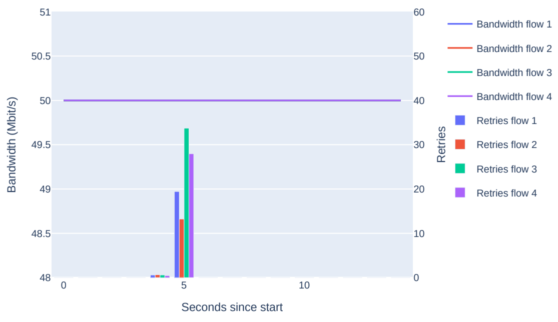


Figure A.176.: Bandwidth and retries during the execution of the Chronicle algorithm

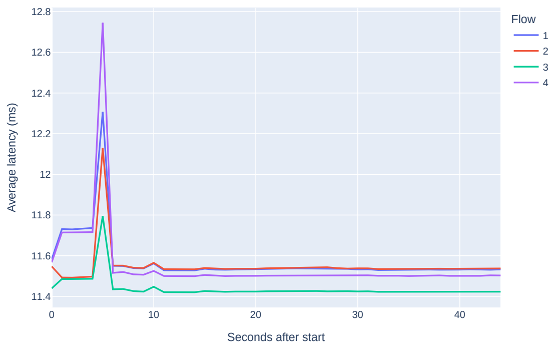


Figure A.177.: Latency between source and destination of all flows during a route change with the Chronicle algorithm

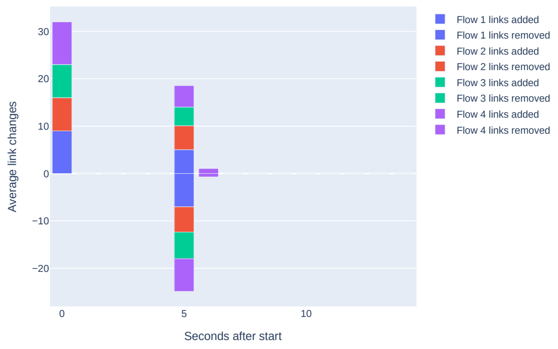


Figure A.178.: Links disappearing and newly appearing in traceroutes when running the Chronicle algorithm

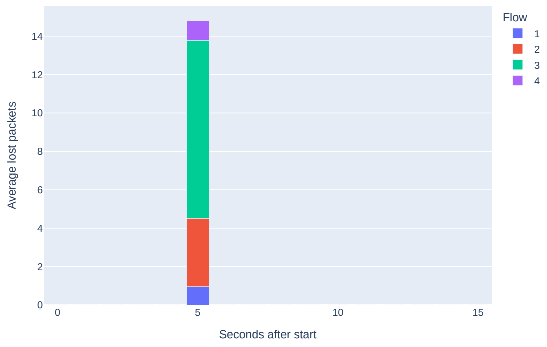


Figure A.179.: Lost packets in our ping test when executing a route change with the Chronicle algorithm

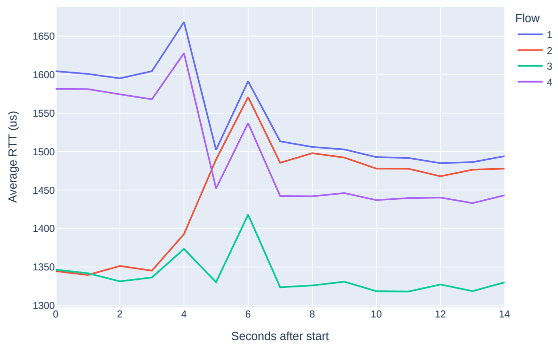


Figure A.180.: Round-Trip-Time between source and destination of all flows during the route changes of the Chronicle algorithm

Greedy

Leaving 5 seconds between rounds

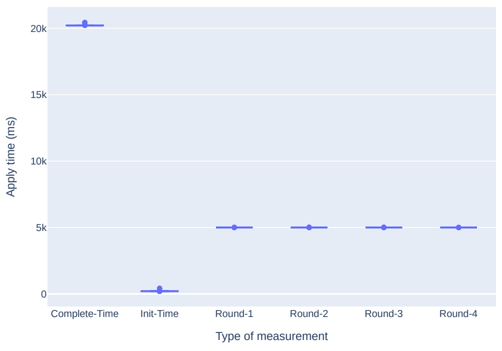


Figure A.181.: Time that it takes to apply the changes using a schedule generated by Greedy

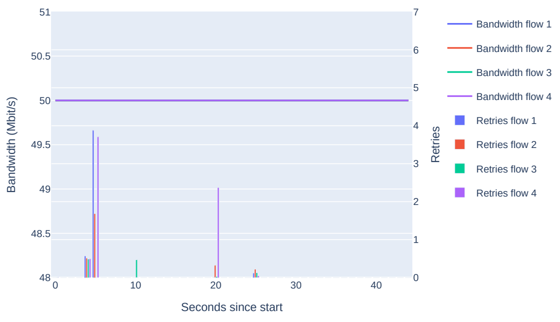


Figure A.182.: Bandwidth and retries during the execution of the Greedy algorithm

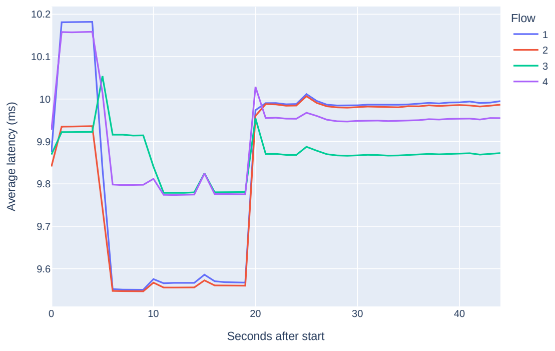


Figure A.183.: Latency between source and destination of all flows during a route change with the Greedy algorithm

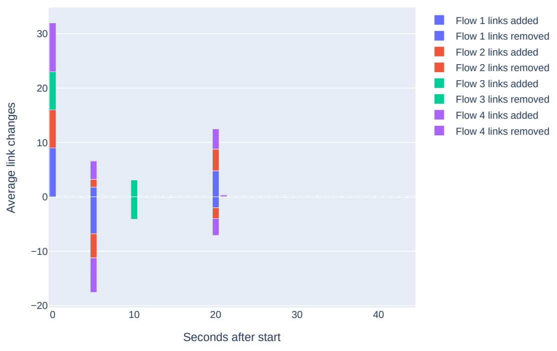


Figure A.184.: Links disappearing and newly appearing in traceroutes when running the Greedy algorithm

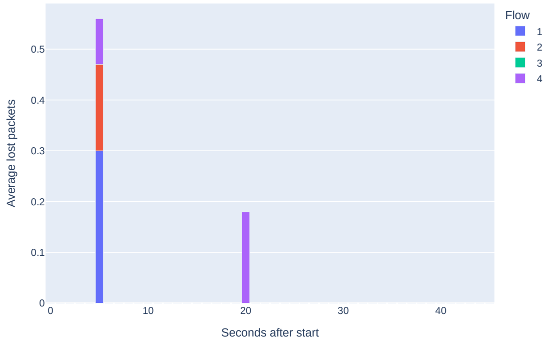


Figure A.185.: Lost packets in our ping test when executing a route change with the Greedy algorithm

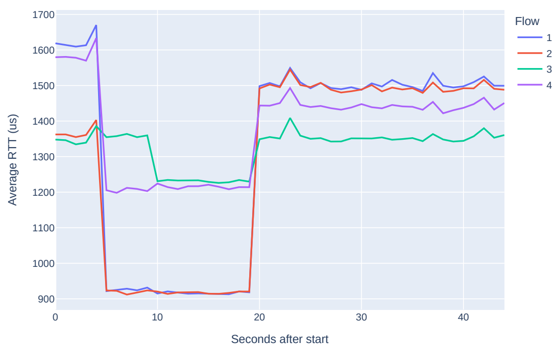


Figure A.186.: Round-Trip-Time between source and destination of all flows during the route changes of the Greedy algorithm

Leaving no time between rounds

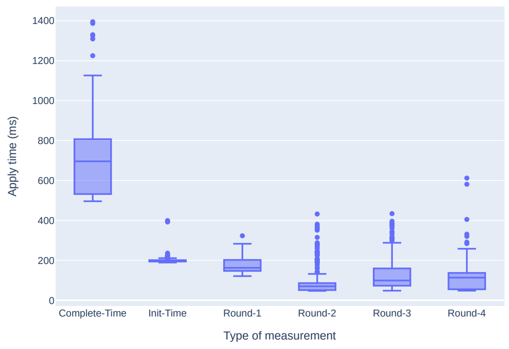


Figure A.187.: Time that it takes to apply the changes using a schedule generated by Greedy

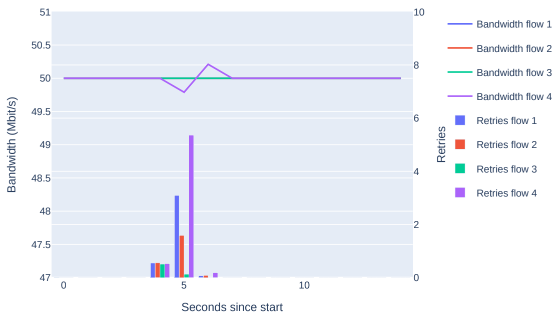


Figure A.188.: Bandwidth and retries during the execution of the Greedy algorithm

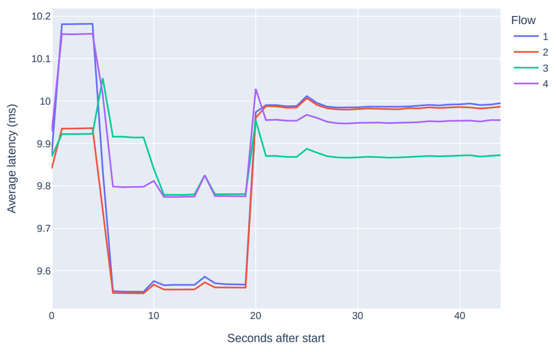


Figure A.189.: Latency between source and destination of all flows during a route change with the Greedy algorithm

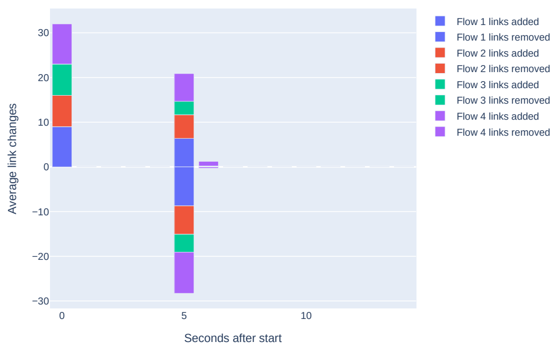


Figure A.190.: Links disappearing and newly appearing in traceroutes when running the Greedy algorithm

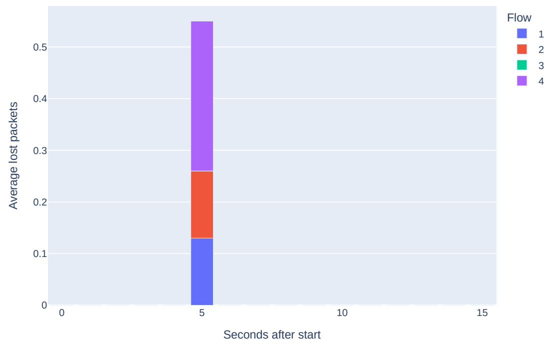


Figure A.191.: Lost packets in our ping test when executing a route change with the Greedy algorithm

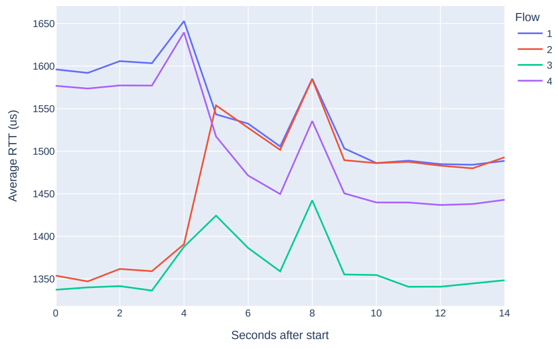


Figure A.192.: Round-Trip-Time between source and destination of all flows during the route changes of the Greedy algorithm

One-Round

Leaving 5 seconds between rounds

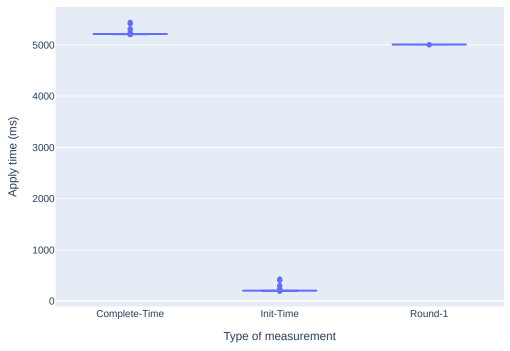


Figure A.193.: Time that it takes to apply the changes using a schedule generated by One-Round

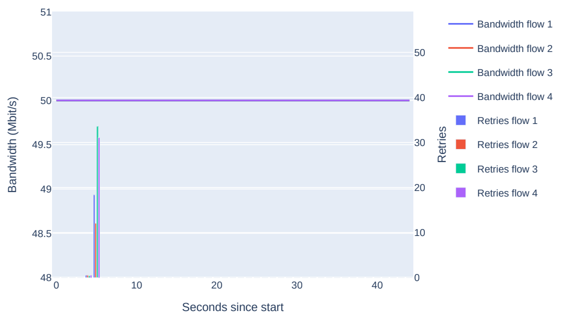


Figure A.194.: Bandwidth and retries during the execution of the One-Round algorithm

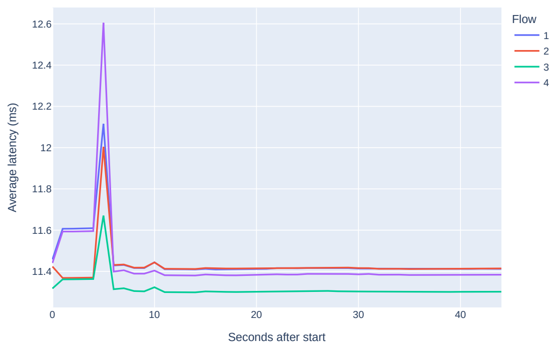


Figure A.195.: Latency between source and destination of all flows during a route change with the One-Round algorithm

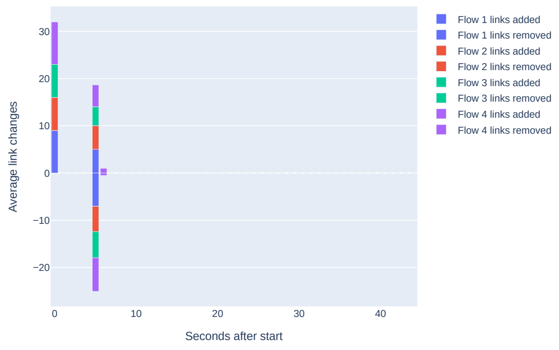


Figure A.196.: Links disappearing and newly appearing in traceroutes when running the One-Round algorithm

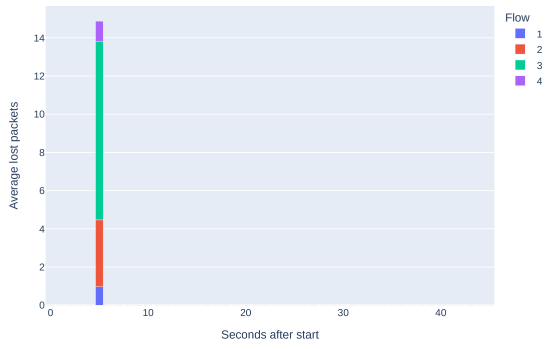


Figure A.197.: Lost packets in our ping test when executing a route change with the One-Round algorithm

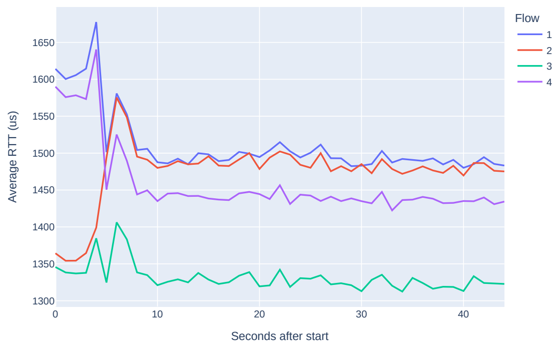


Figure A.198.: Round-Trip-Time between source and destination of all flows during the route changes of the One-Round algorithm

Leaving no time between rounds

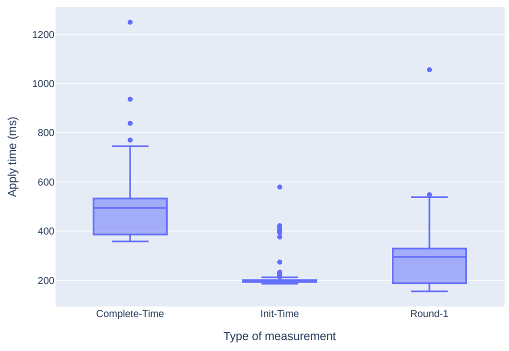


Figure A.199.: Time that it takes to apply the changes using a schedule generated by One-Round

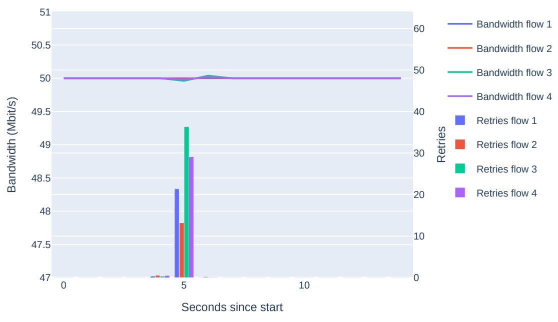


Figure A.200.: Bandwidth and retries during the execution of the One-Round algorithm

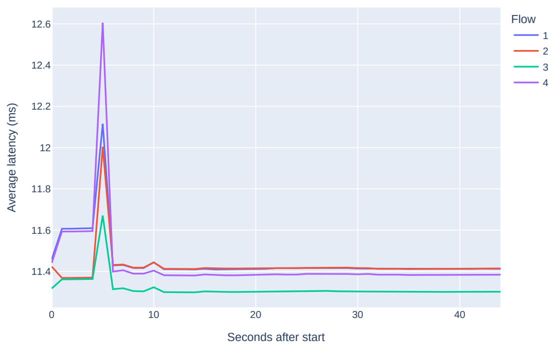


Figure A.201.: Latency between source and destination of all flows during a route change with the One-Round algorithm

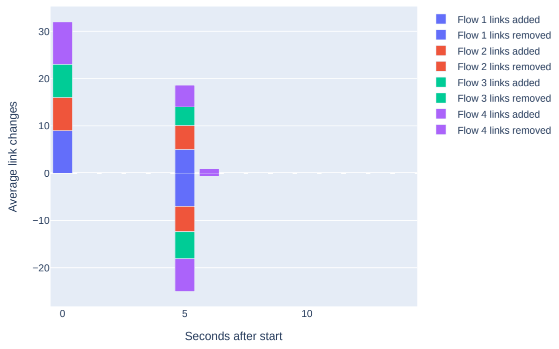


Figure A.202.: Links disappearing and newly appearing in traceroutes when running the One-Round algorithm

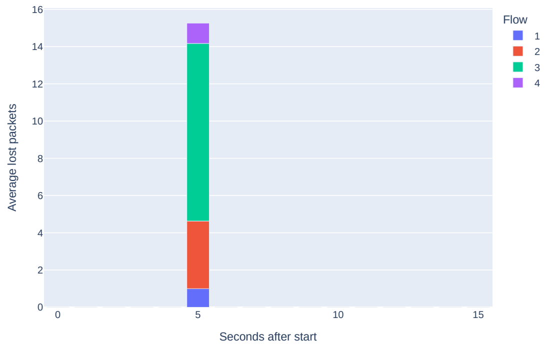


Figure A.203.: Lost packets in our ping test when executing a route change with the One-Round algorithm

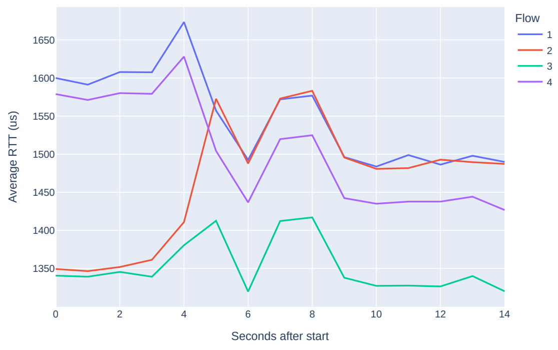


Figure A.204.: Round-Trip-Time between source and destination of all flows during the route changes of the One-Round algorithm

Three-Round

Leaving 5 seconds between rounds

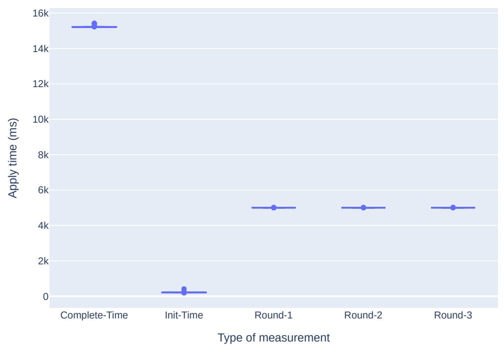


Figure A.205.: Time that it takes to apply the changes using a schedule generated by Three-Round

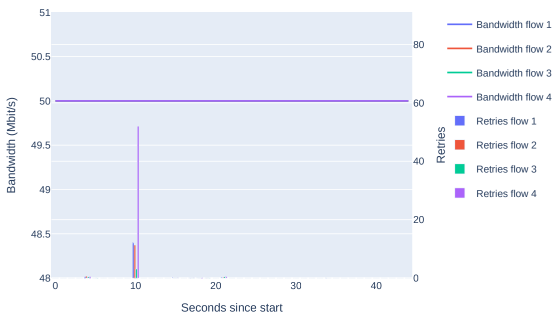


Figure A.206.: Bandwidth and retries during the execution of the Three-Round algorithm

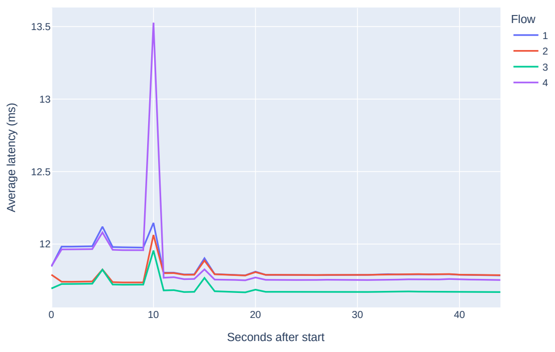


Figure A.207.: Latency between source and destination of all flows during a route change with the Three-Round algorithm

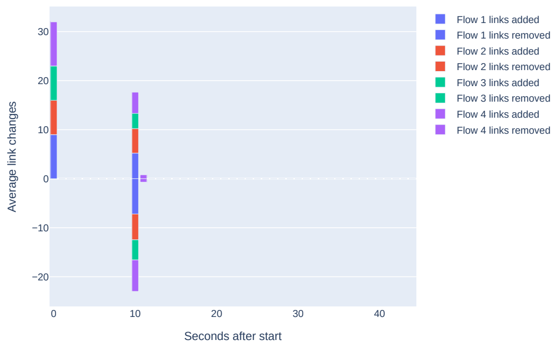


Figure A.208.: Links disappearing and newly appearing in traceroutes when running the Three-Round algorithm

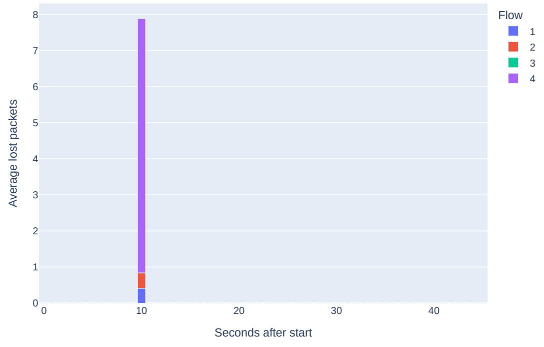


Figure A.209.: Lost packets in our ping test when executing a route change with the Three-Round algorithm

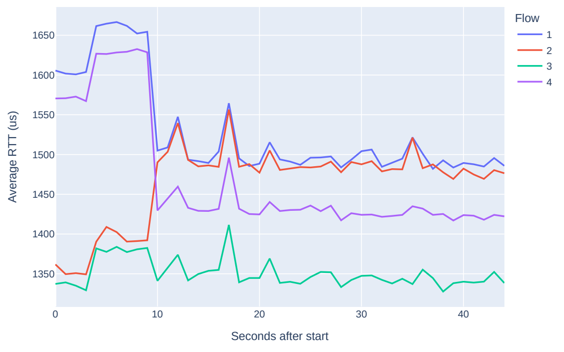


Figure A.210.: Round-Trip-Time between source and destination of all flows during the route changes of the Three-Round algorithm

Leaving no time between rounds

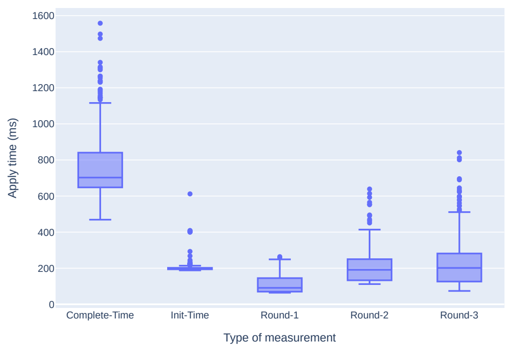


Figure A.211.: Time that it takes to apply the changes using a schedule generated by Three-Round

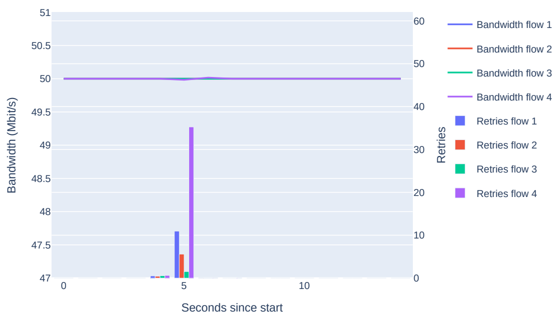


Figure A.212.: Bandwidth and retries during the execution of the Three-Round algorithm

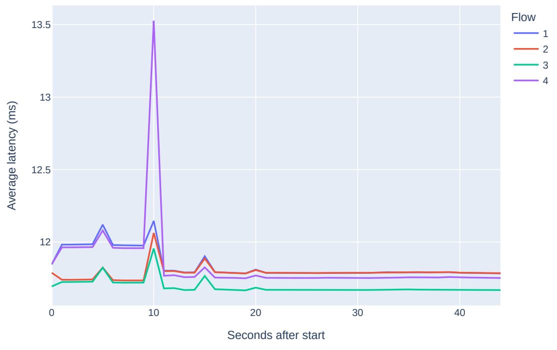


Figure A.213.: Latency between source and destination of all flows during a route change with the Three-Round algorithm

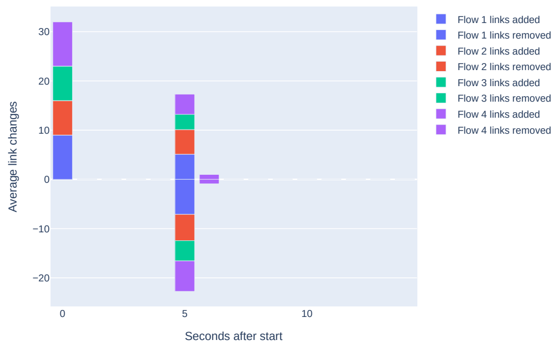


Figure A.214.: Links disappearing and newly appearing in traceroutes when running the Three-Round algorithm

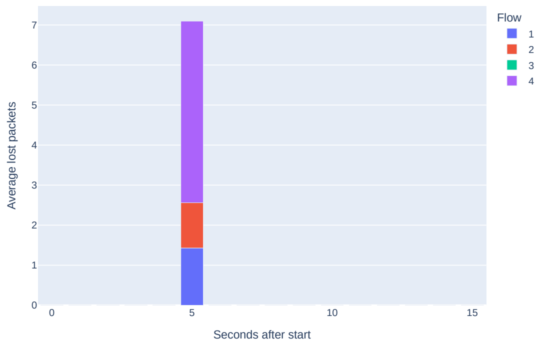


Figure A.215.: Lost packets in our ping test when executing a route change with the Three-Round algorithm

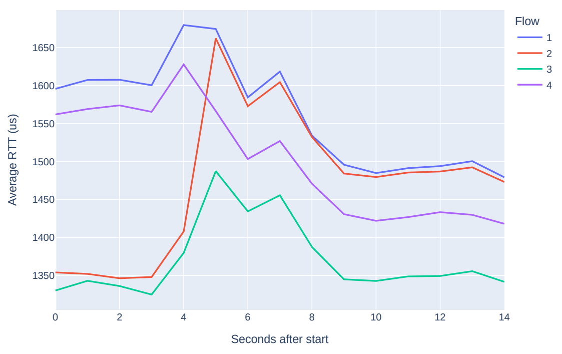


Figure A.216.: Round-Trip-Time between source and destination of all flows during the route changes of the Three-Round algorithm