Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Networked Systems
ETH Zürich — seit 2015

Process Mining for Networking

Semester Thesis / Research in Data Science

Author: Kévin Selänne

Tutor: Dr. Romain Jacob

Supervisor: Prof. Dr. Laurent Vanbever

March to June 2021

Acknowledgments

I wish to express my gratitude to my tutor Dr. Romain Jacob for his insightful guidance and high availability throughout the project. I also want to thank him in particular for his notable work on the collection and processing of the dataset used in the project.

Abstract

Process mining has gained ground in various fields as a methodology for extracting insight from large and complex logs continuously emanating from diverse processes. Thus far it has nevertheless not been applied to a networking context, a gap we bridge with this work by applying process mining techniques to a log based on ordinary network traceroutes. We find that the methods are indeed applicable as we discover and visualize the topology of a network over three months.

Contents

1 Introduction

In the current era of ubiquitous data collection, logs of various kinds feature as an important format of collected data. This is also the case in networking and communication where, for instance, publicly available BGP announcements generate a constant stream of information. In addition, autonomous system (AS) operators set up ample telemetry for monitoring and analyzing the state of their network. It is therefore a current research area to study not only data collection but also the consequent data analysis aspect. In this work, we investigate how the techniques introduced by *process mining* can provide insight in a networking setting.

1.1 Process Mining

Process mining is fundamentally concerned with discovering and analyzing processes based on log data. Discovery produces a concise representation of the process underlying the generation of the log and can be visualized as a diagram such as a Petri net [19]. Comparing a log and a model bring further insight into discovered models, and this has been formalized as conformance and bottleneck analysis. Relevant questions answered through such analysis include: How many deviations from the model does the log exhibit? Where in the model do these deviations occur? Which portions of the process are time-consuming? Are some portions of the process quick on average but slow in a small number of situations? Answers to such questions support problem solving and decision making across a broad range of domains.

Process mining has indeed seen successful application both in academia and the industry; for example in healthcare [16] to analyze steps in patient treatment in a hospital [20], in ICT for improving software development processes [22], and in manufacturing to analyze semiconductor testing processes [21]. A more complete picture of existing uses cases is given in a survey by dos Santos Garcia et al. [7]. Applications to networking have nonetheless remained largely unexplored, prompting this research project.

A key requirement for employing process mining techniques is that the input data be represented as an *event log*. Logical entities flowing through a process are referred to as *cases*, and at each step in the process, a log entry is recorded specifying the name of the *activity* along with a *timestamp*. Table 1.1 presents a simple event log with three cases and four types of activities. Such a format provides the minimal information necessary for modeling a process. It is nonetheless possible to include further event-specific or case-specific attributes for richer analysis, such as the name of a resource executing an activity or separate timestamps for the start and completion of an activity.

Process models can be represented in various notations, with Petri nets [19] having gained popularity in the domain of process mining for their strong theoretical foundation enabling the definition of workflow nets [25], an instance of Petri nets satisfying certain properties. Alternative representations include transition systems (also known as automata) primarily used for theoretical illustrations, along with the Business Process Model and Notation (BPMN) more commonly found in business settings.

Several algorithms have been proposed to discover process models from event logs. Among the first influential algorithms is the Alpha algorithm [28], which considers the *footprint* of an event log, that is, relations between all pairs of activities as inferred from consecutive activities in each trace. These relations can be translated to Petri net components to produce a process model. Figure 1.1 shows the Petri net produced by applying the Alpha algorithm on the log in 1.1. Another discovery algorithm important in this work is "directly follows" modeling [15], which trims down the modeling space from that of the Alpha algorithm's but produces adequate results in simple, well-structured processes.

Table 1.1: A sample log showcasing the minimal event log format.

Case ID	Activity	Timestamp
1	Start	34
2	Start	54
1	Process	59
3	Start	63
2	Reject	70
2	End	72
1	End	78
3	Process	82
3	End	91

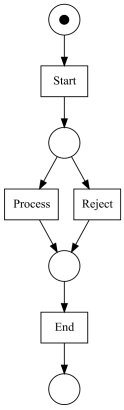


Figure 1.1: A sample Petri net representing the process model mined from the log in Table 1.1 using the Alpha miner. Places are denoted by circles and transitions by rectangles, while the filled black circle represents a token in the start place.

Beyond process discovery, process mining provides value in techniques combining information from an event log and a process model for richer analysis such as conformance checking [17, 27], performance analysis [9, 27], resource analysis [18], and anomaly detection [4]. Conformance checking is concerned with matching traces of events belonging to a case with a process model to quantify fitness, that is, the extent to which the process model can replay cases. An important method in this form of analysis involves alignments [5], where a trace in the event log is matched with the closest valid execution of the process model.

1.2 Network Discovery and Testing

Network discovery itself has been amply studied, particularly in the early 2000s. At the time, measurement was largely limited to passive techniques at the level of a node or a link, or active techniques between end hosts, as categorized in a survey by Lawrence et al. [13]. Passive measurement is primarily concerned with estimating traffic intensity between observed source–destination pairs, whereas active measurement probes the network typically using ICMP echo packets as part of a traceroute. Joining results from measurements between a set of several hosts allows building a more intricate map of the network and its characteristics than local measurements obtained passively, which also assume some level of access into the network unlike active techniques with lighter assumptions. Hence this latter approach has received much more attention in the inference of network topologies.

Active measurement techniques mainly rely on traceroute with research efforts focusing on overcoming its limitations. Key challenges include *alias resolution*—determining the IP addresses belonging to interfaces of a single router—along with reducing measurement overhead to enable scalability. For example, Rocketfuel [23] addresses these issues by exploiting the IP identifier field for alias resolution and by using public BGP routing information to prune likely irrelevant paths for scalability. Skitter [10], on the other hand, consists of a carefully chosen set of vantage points and destinations. Alias resolution is tackled by probing interfaces using UDP packets to an unused port with the intention of triggering a "port unreachable" reply from the router's loopback address. A survey by Donnet and Friedman [6] provides a more complete account of traceroute-based methods.

With the emergence of Software-Defined Networking (SDN) in the early 2010s, interest in discovering topologies formed by a priori unknown routers has diminished. Instead, research interest has increased in settings where access to switches is assumed, such as data centers,

hence giving a link-layer topology against which to compare measurements. In other words, research focus has shifted from network-layer topology discovery towards testing and anomaly detection at the link layer. Classification into active and passive approaches remains nonetheless relevant.

In passive techniques, the general idea is to monitor for interesting packets at switches and when matching packets are encountered, either send information to a collector server or embed information in the packet header to be removed and processed at the edge of the network. Examples falling into this category are NetSight [8], which sends "postcards" containing key information from packet headers to a collector, and PathDump [24], which embeds identifiers of switches crucial for path reconstruction into the header. Everflow [32] also has a passive component where TCP packets are mirrored to a collector.

In active methods, the underlying notion is crafting artificial packets fulfilling conditions of interest and injecting them to switches. Then, SDN traceroute [2], for instance, takes the approach of tagging these packets to forward them to the controller at each hop. Everflow's active component takes another approach by encapsulating these debugging packets to a loopback IP address.

Advances in programmable data planes have opened new avenues for more scalable network measurements under in-band network telemetry (INT) [11]. For example, PINT [3] allows defining queries that result in small chunks of metadata being probabilistically embedded into packet headers throughout the network and aggregated and reconstructed at switches at the edge of the network. Similar to Everflow's passive component, dShark [30] relies on traces being mirrored to collectors and it instead focuses on the subsequent analysis with a distributed architecture aiming for a high throughput. A third INT-based approach, LightGuardian [31], splits small, packet-level data structures called *sketches* further into *sketchlets* to reduce overhead and extends the structure with capabilities to record flow-level statistics.

1.3 Contribution

We explore the potential of process mining in the context of networking. In particular, we look into topology discovery and how a topology varies over the course of three months. To that end, we collect traceroutes from RIPE Atlas [1] and process them into event. As the process mining tool, we use the open-source ProM [29].

We find that process mining techniques work largely as expected, allowing to effortlessly visualize the topology of the network based on the traceroute event log. In terms of how the topology evolves over time, we observe route changes that appear to stem from deliberate configuration changes rather than load balancing. We hypothesize that changes in routes are induced by increased delay on the original path, but that turns out not to be the case. A separate question of interest concerns whether some links transmit packets to different destinations with different delays, which we find to indeed occur in rare yet conclusive instances.

The report is divided into four additional sections. We first present relevant process mining techniques in Section 2, after which we present the dataset collected for this purpose in Section 3. We then show some observations from the dataset in Section 4 before finishing with concluding remarks in Section 5.

2 Methods

2.1 ProM: The Process Mining Toolkit

ProM [29] is the original open-source tool implementing process mining techniques under a user-friendly graphical user interface. The interface is divided into three sections: resources (Figure 2.1), plugins (Figure 2.2), and visualization. The visualizations are specific to each plugin—the Alpha Miner and Directly Follows Visual Miner plugins used in this work are introduced in Sections 2.2 and 2.3. ProM supports importing logs formatted as comma-separated values (CSV). Cases and activities may consist of several attributes in the log, which can be specified when imported.

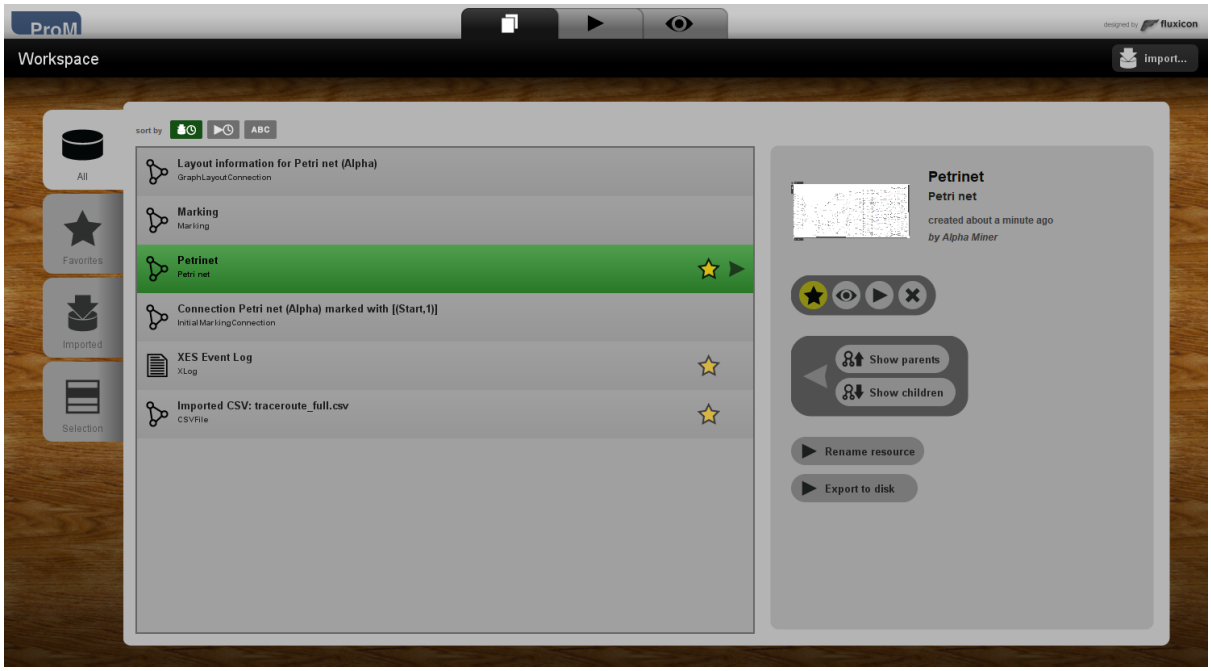


Figure 2.1: The ProM (version 6.10) main screen listing the loaded resources. In this example, a CSV file has been imported and converted to the XES event log format, which is the primary format used when computing models.

2.2 Alpha Miner

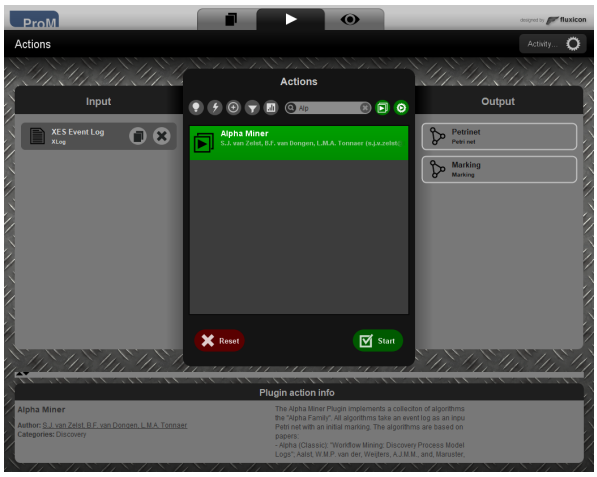
The Alpha Miner plugin implements the Alpha algorithm [28] to construct a Petri net conforming to the provided log. This serves as a simple and general visualization of an event log without needing to configure any parameters.

In detail, the algorithm considers the sequences of activities within every case: one such sequence is referred to as a trace and thus the input is a multiset of traces. For the example log in Table 1.1, the multiset of traces is

$$L = \{(\text{Start}, \text{Process}, \text{End}), (\text{Start}, \text{Reject}, \text{End}), (\text{Start}, \text{Process}, \text{End})\}. \tag{2.1}$$

Notably, the Alpha algorithm neglects frequency and time aspects, such as the prevalence of certain traces or the relative ordering of cases, which leads us to also employ a more complex plugin described in Section 2.3.

The following description of the Alpha algorithm takes inspiration from the Coursera course on process mining by van der Aalst [26]. To simplify notation, we define the following for all pairs of activities x and y within traces:



(a) The Alpha Miner (Section 2.2) is used to build a Petri net from the given log.

(b) The Directly Follows Visual Miner (Section 2.3) builds "directly follows" models that can be interacted with to highlight or filter for specific characteristics.

Figure 2.2: The ProM plugins view lists the available plugins and selecting one shows the required input and resulting output formats.

(i) $x > y \iff x$ is directly followed by y — Direct succession.

(ii) $x \to y \iff x > y$ and not $y > x$ — Causality.

(iii) $x||y \iff x > y$ and $y > x$ — Parallel activities.

(iv) $x\#y \iff$ not $x > y$ and not $y > x$ — Choice or no direct relationship.

Evaluating relationships between all pairs of activities across a set of traces allows building a *footprint*, a matrix summarizing relationships between activities. Table 2.1 shows such a footprint for the example traces in Equation 2.1. Note that the existence of direct succession $>$ is evaluated across all traces, and thus it is enough for a $\to$ relationship to occur in a single trace for it to take precedence over $\#$. Similarly $||$ takes precedence over $\to$.

Table 2.1: Footprint of activities for the multiset of traces in Equation 2.1, that is, the log from Table 1.1. Note the multiple interpretations of #: Process and Reject are mutually exclusive activities, while Start and End have no direct relationship.

	Start	Process	Reject	End
Start	#	$\to$	$\to$	#
Process	$\leftarrow$	#	#	$\to$
Reject	$\leftarrow$	#	#	$\to$
End	#	$\leftarrow$	$\leftarrow$	#

The steps of the Alpha algorithm are outlined below, illustrated with our running example in gray:

0. Assume a multiset of traces L over transitions T. $L = \{(\text{Start}, \text{Process}, \text{End}), \ldots\}$ $T = \{\text{Start}, \text{Process}, \text{Reject}, \text{End}\}$	The multiset of traces is the event log and transitions are activities.
1. $T_L = \{t \in T \mid \exists \sigma \in L : t \in \sigma\}$ $T_L = \{\text{Start}, \text{Process}, \text{Reject}, \text{End}\}$	All occurring transitions.
2. $T_I = \{t \in T \mid \exists \sigma \in L : t \in \text{head}(\sigma)\}$ $T_I = \{\text{Start}\}$	Initial transitions.
3. $T_O = \{t \in T \mid \exists \sigma \in L : t \in \text{tail}(\sigma)\}$ $T_O = \{\text{End}\}$	Final transitions.
4. $X_L = \{(A, B) \mid A, B \subseteq T_L \wedge A, B \neq \emptyset$ $\wedge \forall a \in A, b \in B : a \to b$ $\wedge \forall a_1, a_2 \in A : a_1 \# a_2$ $\wedge \forall b_1, b_2 \in B : b_1 \# b_2\}$ $X_L = \{(\{\text{Start}\}, \{\text{Process}\}),$ $(\{\text{Start}\}, \{\text{Process}, \text{Reject}\}), \ldots\}$	Causally related pairs of internally independent sets: all transitions in A can be followed by all transitions in B, and any two transitions within A or B do not follow each other.
5. $Y_L = \{(A, B) \in X_L \mid \forall (A', B') \in X_L :$ $A \subseteq A', B \subseteq B' \implies (A, B) = (A', B')\}$ $Y_L = \{(\{\text{Start}\}, \{\text{Process}, \text{Reject}\}),$ $(\{\text{Process}, \text{Reject}\}, \{\text{End}\})\}$	Maximal set pairs of X_L: A and B contain all applicable transitions.
6. $P_L = \{p_{(A,B)} \mid (A, B) \in Y_L\} \cup \{i_L, o_L\}$ $P_L = \{p_{(\{\text{Start}\},\{\text{Process},\text{Reject}\})}, \ldots, i_L, o_L\}$	Set of places. i_L and o_L are added source and sink places, respectively.
7. $F_L = \{(a, p_{(A,B)}) \mid (A, B) \in Y_L \wedge a \in A\}$ $\cup \{(p_{(A,B)}, b) \mid (A, B) \in Y_L \wedge b \in B\}$ $\cup \{(i_L, t) \mid t \in T_I\} \cup \{(t, o_L) \mid t \in T_O\}$ $F_L = \{(\text{Start}, p_{(\{\text{Start}\},\{\text{Process},\text{Reject}\})}),$ $(p_{(\{\text{Start}\},\{\text{Process},\text{Reject}\})}, \text{Process}), \ldots,$ $(i_L, \text{Start}), (\text{End}, o_L)\}$	Set of arcs between places and transitions: from transitions to places, from places to transitions, from the source place to initial transitions, and from final transitions to the sink place.
8. $\alpha(L) = (P_L, T_L, F_L)$ Visualized in Figure 1.1.	The resulting Petri net.

While the Alpha algorithm produces adequate results with many datasets thanks to its generality, the context of traceroutes enables further assumptions to be made, leading to a more suitable class of models and a more versatile ProM plugin.

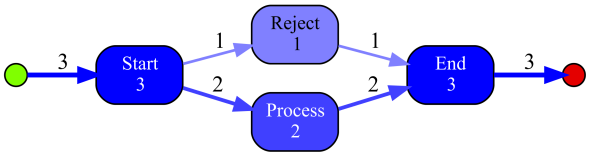
2.3 Directly Follows Visual Miner

Given the Alpha algorithm's limitations in visualizing frequencies and time, we also employ the *Mine with Directly Follows visual Miner* plugin (documentation [14]) which implements a simple directly follows model (DFM) [15]. Such a model only considering direct succession of activities introduces the limitation of not modeling concurrency, which the Alpha algorithm is capable of modeling.

However, this tradeoff is inconsequential in the context of traceroute data: routers and links are ordered sequentially, so a packet can traverse only one link at a time rather than several links concurrently. In return, the visual miner provides means to filter out infrequent traces

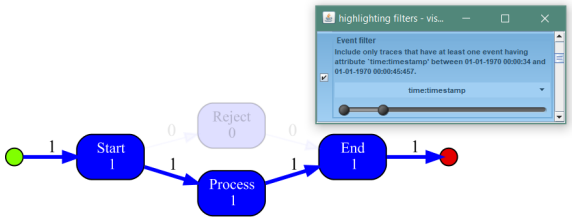
and restrict the visible model to data within a chosen time interval. These features are detailed in Figure 2.3.

Given that the concurrency limitation of DFMs does not apply in our context, the plugin with its rich features is very well suited for analyzing the dataset described in the following section, Section 3.

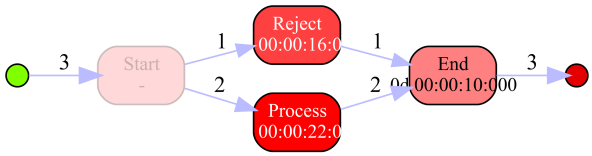


(a) The full DFM. Numbers indicate the number of traces executing each activity and they are also graphically rendered using a blue color scale.

(b) A simplified view of the DFM omitting the less frequent "Reject" activity. A slider in the interface allows pruning less frequent traces. Specifically, the *paths* slider defines a lower bound for the proportion of traces that should be represented in a model omitting as many of the least frequent edges as possible. The model above is produced by any value under 66.7 %.



(c) The full DFM filtered to only include traces having an event within a chosen time interval. The times in Table 1.1 have been interpreted as Unix timestamps, which is why the time filter includes January 1, 1970, as the date.



(d) The plugin also provides a view visualizing *sojourn times*, that is, times elapsed between completions of two consecutive activities. Numbers under activities denote the mean sojourn times, which are also rendered using a red color scale.

Figure 2.3: A directly follows model based on the event log in Table 1.1 and visualized by the Directly Follows Visual Miner. The rounded rectangles denote activities.

3 Dataset

We apply process mining on an event log extracted from traceroutes from one source to ten destinations. The traceroutes originate from hourly dumps of traceroute measurements in RIPE Atlas [1], a geographically distributed network of probes and anchors. Volunteers hosting a probe or an anchor in their network may request measurements such as pings and traceroutes between these devices, and the results are typically made publicly available.

These hourly dumps are filtered to only consider traceroutes with a straightforward interpretation to allow reconstructing a path based on hop information. Namely, we discard traceroutes

(i) where all probes at any hop time out, as we would be missing information about a router,

(ii) with more than one IP address at a given hop, implying two probes took different paths, and

(iii) where the traceroute encounters an error.

After the filtering, we are left with 66 062 traceroutes (on average 31 per hour and 749 per day) to analyze further. Notably, every hop now has exactly one IP address to be considered a router interface, which allows determining the path to a destination as the sequence of these IP addresses. Furthermore, we can estimate the delay from one router to the next by averaging round-trip times at every hop, halving those to estimate one-way delays, and computing the difference of these estimates between consecutive routers. This data suffices to build an event log of the form presented in Table 3.1. The parameters for importing the event log in ProM are in Figure 3.1.

Table 3.1: Excerpt of the constructed event log. The Unix timestamp of a traceroute coupled with its source and destination address uniquely identify a case. The hop-specific source and destination addresses identify the activity starting and ending at the times indicated by the hop-specific timestamps. The second block of lines shows the boundary between two distinct traceroutes: the timestamp increases and the hop attributes restart from the beginning.

Unix timestamp (s)	Source address	Destination address	Hop #	Hop source address	Hop destination address	Hop start timestamp (ms)	Hop end timestamp (ms)
1614557183	51.89.117.81	80.241.3.66	1	51.89.117.81	51.89.116.1	0.00	0.17
1614557183	51.89.117.81	80.241.3.66	2	51.89.116.1	192.168.143.254	0.17	0.21
1614557183	51.89.117.81	80.241.3.66	3	192.168.143.254	10.13.65.254	0.21	0.33
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1614557183	51.89.117.81	80.241.3.66	16	92.47.145.190	88.204.208.2	56.10	56.11
1614557183	51.89.117.81	80.241.3.66	17	88.204.208.2	80.241.3.66	56.11	60.04
1614558581	51.89.117.81	103.126.52.180	1	51.89.117.81	51.89.116.1	0.00	0.17
1614558581	51.89.117.81	103.126.52.180	2	51.89.116.1	192.168.143.254	0.17	0.19
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

From all the publicly available traceroutes, we chose to investigate further traceroutes originating from 51.89.117.81 and targeting ten destinations common during the initial phase of data collection at the end of March 2021. With such a heuristic, we expected to obtain ample data during the three months of data collection, and indeed, seven of the ten destinations had traceroutes for most of the period from March to May 2021 (Table 3.2).

Suitably formatted for process mining, this dataset provides an intuitive mapping between process mining and networking concepts. Each traceroute measurement is a case in process

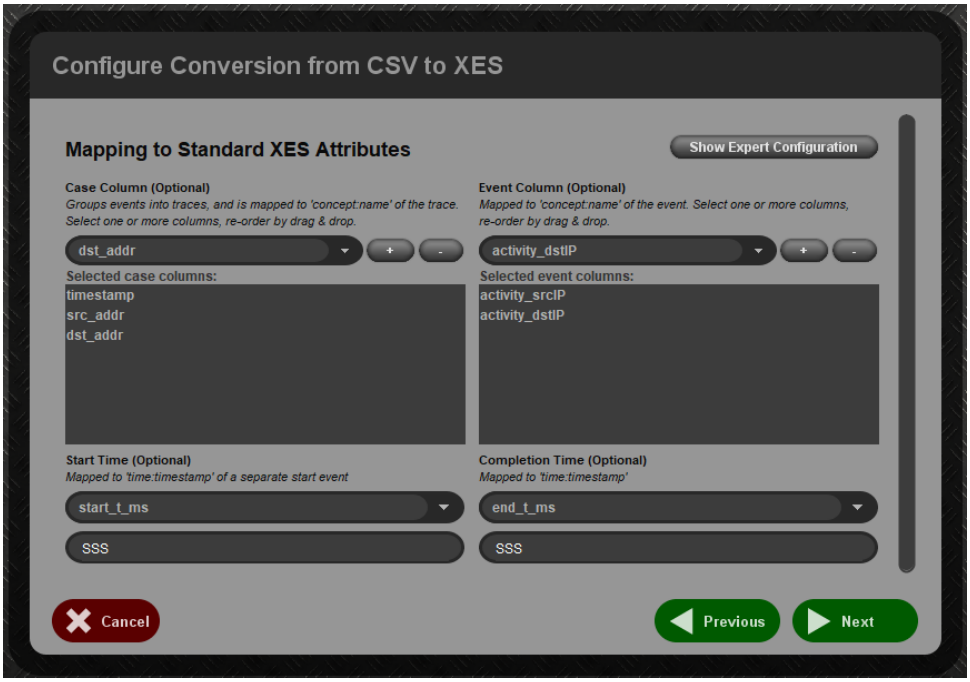


Figure 3.1: Parameters to provide to the CSVImporter plugin in ProM to convert a log of the form shown in Table 3.1 into the XES event log format. `timestamp`, `src_addr`, and `dst_addr` identify the case; `activity_srcIP` and `activity_dstIP` identify activities starting at `start_t_ms` and ending at `end_t_ms`. The `SSS` format for time defines the values as milliseconds since January 1, 1970 at 00:00:00.000. Hence the resulting cases are treated by ProM as if they had all happened in 1970, but this is inconsequential as we are not interested in potentially interleaved traceroutes. The relative ordering of traceroutes can be recovered from the timestamp attribute, which needs to be duplicated to be accessed in the Directly Follows Visual Miner plugin.

mining and a proxy for the path an individual packet would take from the source to the destination. IP addresses returned at each hop are paired to produce activities representing links between pairs of router interfaces; physical links may nonetheless slightly deviate from these. Finally, approximated starting and ending times of traceroute hops on the one hand estimate delay between two routers, and on the other hand serve as starting and ending timestamps for computing activity service times. These characteristics make the dataset well suited for exploring process mining in the context of networking, although the content itself presents no particular significance to us.

Table 3.2: The IP addresses of the chosen traceroute source and ten destinations, and the mean number of hops across all measurements to provide an intuitive impression of the distance between the hosts involved. The last column indicates the proportion of hours in the collected three months of data that had at least one traceroute to the corresponding destination: seven destinations have data in at least 95 % of the 2111 hours between March 1 at 00:00 and May 29 at 23:59 excluding two complete days and one full hour with missing data. Due to a technical glitch, there is no data in the following intervals (UTC): March 30 23:02:44 to April 1 00:31:32, April 3 from 01:36:29 to 03:47:37, and April 7 23:09:37 to April 9 00:53:30.

Source address	Destination address	Mean hop count	Total number of traceroutes	Proportion of hours with ≥ 1 traceroute
51.89.117.81	91.132.8.99	17.0	7886	99.95 %
	159.253.17.19	19.0	7797	99.95 %
	193.141.27.220	18.0	7199	99.91 %
	197.80.104.36	21.2	8092	99.91 %
	80.241.3.66	16.7	12 814	99.72 %
	45.65.244.254	19.3	7898	97.20 %
	103.126.52.180	23.2	4741	95.45 %
	80.65.65.220	20.0	5221	71.80 %
	164.113.94.217	20.8	4285	65.55 %
	194.187.174.6	25.0	129	4.74 %

4 Results

The results consist of five subsections considering the following questions:

4.1 *What does the overall topology of the network over the first two weeks look like, visualized with ProM?*
A visualization with the Alpha algorithm shows that two links divide the network into sections with private and public IP address prefixes.

4.2 *What do the paths to individual destinations look like and do they evolve over time?*
Single paths predominate at short time intervals, implying no load balancing. There are path changes to all destinations, though frequencies and complexities of changes vary significantly between destinations.

4.3 *Are there any time regularities to routing changes?*
No such regularities are discernible, but changes tend to occur among a handful of paths per destination.

4.4 *Is there a correlation between changing routes and delay prior to the change?*
No. While delay typically changes after switching to another path, no intuitive causality is observed.

4.5 *Do any links show systematically different delays for packets to different destinations?*
Against the expectation that link delays would be independent of destination, two links consistently transfer packets to different destinations at different rates for at least some period of time.

For readers in the Networked Systems Group at ETH, interactive versions of some figures are available at `https://gitlab.ethz.ch/nsg/students/projects/2021/sa-2021_processminingnetworking/-/tree/master/report/figures/plotly`.

4.1 Topology overview

What does the overall topology of the network over the first two weeks look like, visualized with ProM?

A unique feature offered by process mining is visualizing an event log in the more interpretable form of a Petri net. Figure 4.1 visualizes paths inferred from the full three months of the traceroute data. However, using the full data leads to too much complexity since a Petri net generated by the Alpha algorithm fits the log perfectly without any simplification or aggregation. This property results in giving equal visibility to high-frequency and less relevant low-frequency traces as well as superimposing all possible paths to a destination, ignoring the dimension of time. We therefore limit the Petri net to the first two weeks of March in Figure 4.2.

This illustration provides an overview of the topology with few details on individual segments. Notably, the first few hops from the source to any destination fall within a private network and all traffic is routed through two specific links and their respective pairs of routers before leaving. As the remaining weeks of the data maintains a similar structure, these routers likely serve a special function such as firewalling or analytics. The portion of the network after these routers is in part hosted within the same AS but gradually shifts to other ASes depending on the destination.

The Petri net capturing the three months of data is too complex to analyze visually. Restricting the visualization to the first two weeks provides a more manageable view revealing a network divided into a private portion and a subsequent public portion.

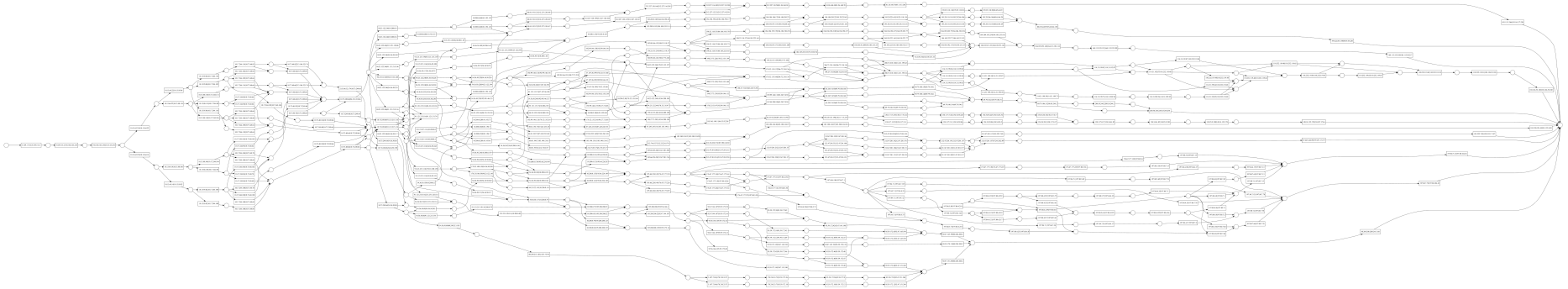


Figure 4.1: The Petri net resulting from applying the Alpha algorithm on the three months of data is too complex to interpret due to every unique trace to ever have occurred being represented.

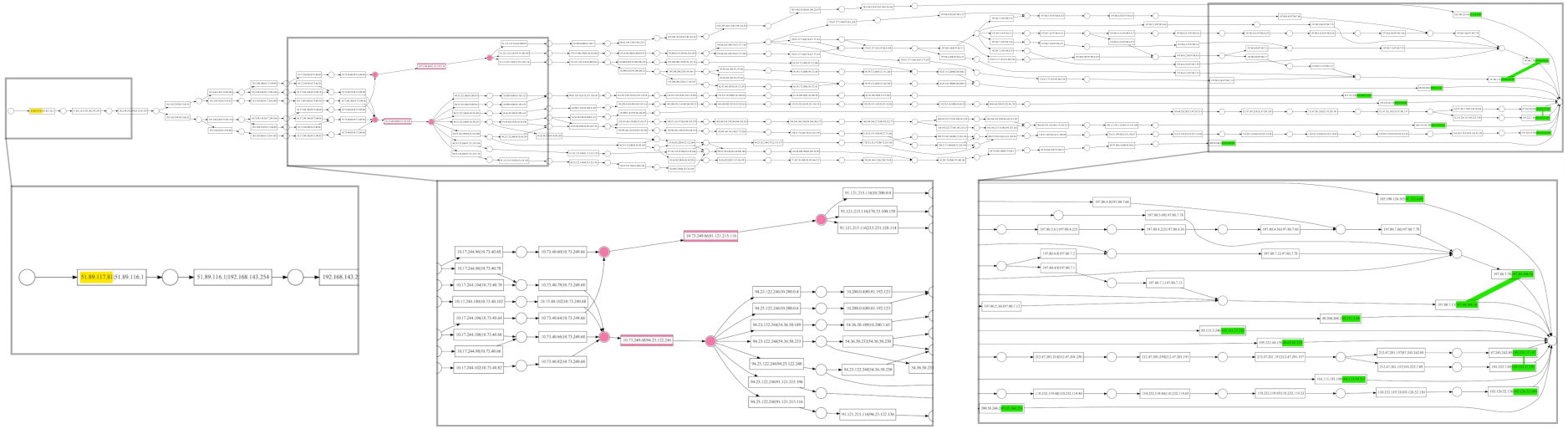


Figure 4.2: A Petri net representing the network as inferred by the Alpha algorithm using the first two weeks of the data. Each rectangle represents a link formatted as `link_source_IP`—`link_destination_IP`. The traceroute source IP address is highlighted in yellow (on the left) and the final destination IP addresses are highlighted in green (on the right). The pink circles in the middle indicate a set of four routers that divide the network into two sections: to the left is a private network with IP addresses under prefixes 192.168.0.0/16 and 10.0.0.0/8, and to the right is a network primarily consisting of public IP address prefixes apart from the occasional 10.0.0.0/8 for some routers still within the source AS.

4.2 Paths by destination

What do the paths to individual destinations look like and do they evolve over time?

The overview of the previous section lacks in capturing frequencies and time evolutions of paths to destinations. The Directly Follows Visual Miner responds to these needs by applying a color scale to the graph and offering an interface to interactively select a time interval to consider in the visualization. Additionally, it provides means to simplify the resulting model by disregarding low-frequency traces.

Figures 4.3, 4.4, and 4.5 present a small case study of what can be observed from traceroutes to 45.65.244.254, and the observations are of similar nature for all other destinations. Each of the 10 destination IP addresses were analyzed as separate graphs to ease the computational load and since overlapping paths between different destinations do not provide further meaningful insight. Towards some destinations, traceroute paths remained very stable with few changes throughout the three months, while there was more variability towards other destinations, as is demonstrated in the following subsection, Section 4.3. Nonetheless, even variable paths remained stable at hourly timescales, indicating that load balancing does not take place at the level of individual source–destination pairs. Instead, changes in paths are better attributed to modified network configurations.

Analyzing paths at a detailed level reveals that one path dominates at any given time, that is, no load balancing is present. Additionally, how often route changes occur and the length of the new segment varies by destination.

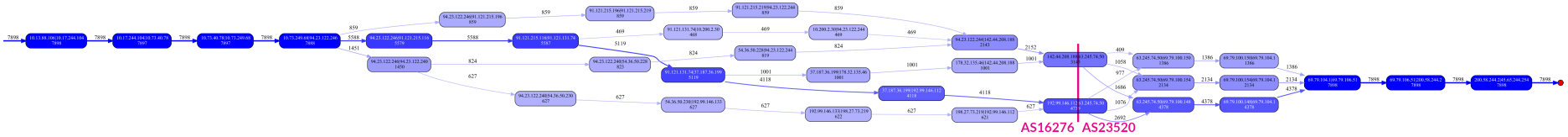


Figure 4.3: Frequencies of links taken when reaching 45.65.244.254 over the three months of data collection. Darker blue shades indicate higher frequencies. Interpreting the IP addresses around the pink AS border between the source AS16276 and AS23520 indicates that the former has two routers connected to an edge router in the latter. The portion of the graph to the left, close to the source, is a linear path cut out of the picture due to size constraints. Additionally, the model has been pruned to use the 98.5 % most frequent traces to reduce clutter by traces occurring only a few times; such omissions are visible in frequencies of incoming and outgoing links not summing up to equal numbers.

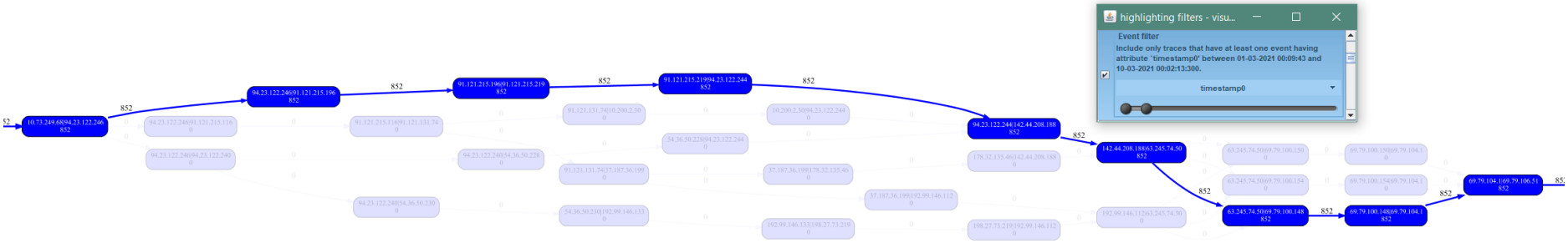


Figure 4.4: Between March 1 and 10, all 852 traceroutes follow a single path to 45.65.244.254. The light-shaded rectangles represent links that are part of the model but remain unused during this interval.

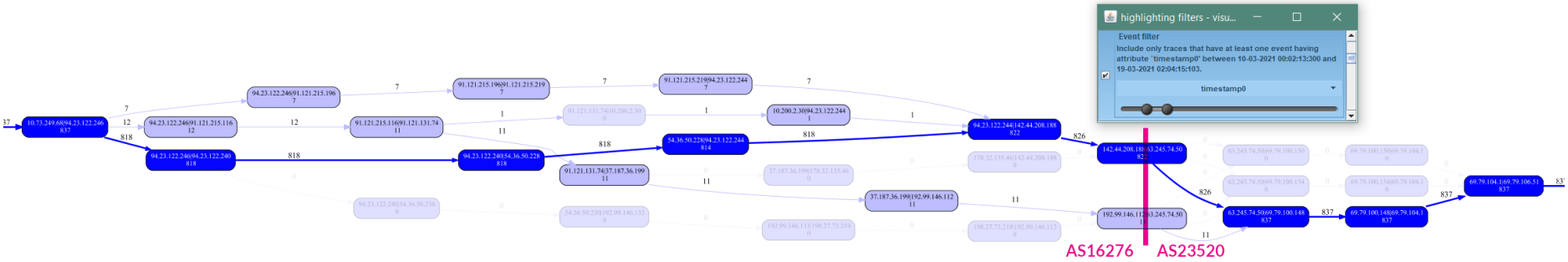


Figure 4.5: Between March 10 and 19, the vast majority of traceroutes follow a single path to 45.65.244.254. A handful of traces take deviating paths; for example, there are 11 traceroutes where the source AS16276 uses a different router to reach AS23520.

4.3 Overview of path changes by destination

Are there any time regularities to routing changes?

Having identified that some destinations experience changes in paths over time, a question of interest is to investigate whether there are any regularities in, for instance, the frequency of route changes. Given that observed routes typically remain constant for at least several hours, we aggregate them over 8-hour intervals and keep the most prevalent routes to represent these intervals in Figures 4.6 and 4.7. We split routes at the two links identified in Figure 4.2 as the changes inside and outside the private network can be considered independent.

In Figure 4.6, we observe that the number of path changes over the three months ranges from 1 to 21 depending on the destination. Nonetheless, even with high variability in paths, the same small set of paths is reused: the number of unique major paths does not exceed 8 for any destination. This is expected as physical connections are altered more rarely than software configurations, which can be updated with less effort.

In Figure 4.7 covering the private portion of the network shared between all destinations, routes remain more stable. When changes do take place, not all destinations using the same path are necessarily affected (pink path) and a path no longer used towards some destination may be picked up by another destination (green path).

There is no regularity such as a certain periodicity for path changes, which take place among a small number of possible paths for each destination. The smaller private network exhibits path changes more rarely than the public part of the network after it.

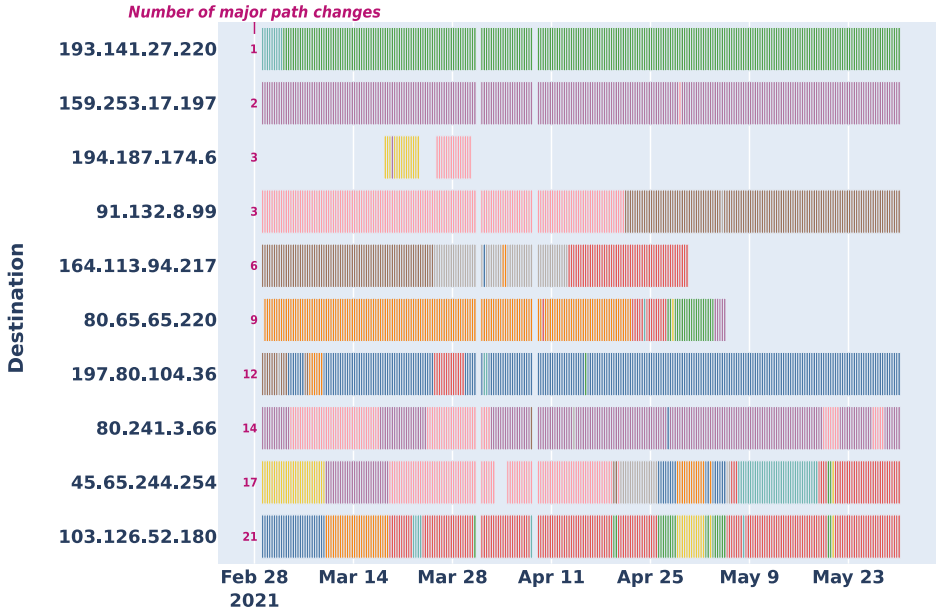


Figure 4.6: Predominant paths during 8-hour intervals from after the private network to each of the ten destinations. Each color represents one path and gaps indicate missing data in the interval, while "major path changes" designate transitions from one predominant path to another. Interactive plot available.

4.4 Correlation between path changes and delay

Is there a correlation between changing routes and delay prior to the change?

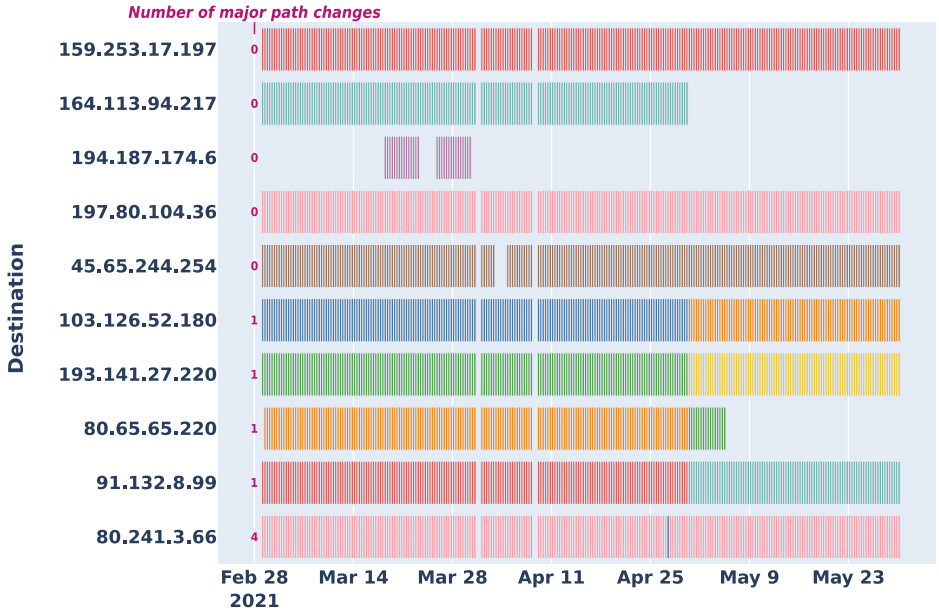


Figure 4.7: Predominant paths during 8-hour intervals within the private network to each of the ten destinations. Five destinations maintain constant paths, four destinations exhibit one switch, and 80.241.3.66 briefly sees two paths differing from the otherwise constant path. In particular, the green and pink paths are the same for both pairs of destinations for which they are used, but the other colors are specific to each destination. Interactive plot available.

A potential explanation for changing routes is that an increase in delay, implying congestion, would trigger rerouting to balance load. We investigate this hypothesis by plotting shares of paths over time overlaid with delay to visually detect such an effect. To reduce noise and improve readability, we reuse the 8-hour time buckets introduced in the previous subsection (4.3) to average both path shares and delay. Additionally, we restrict the delay to the region delimited by the two routers between which all path changes to a destination take place. This is to mitigate changes in delay occurring on individual links elsewhere, unrelated to path changes.

Figure 4.8 looks at this effect for destination 45.65.244.254. The segment for computing delay runs from 94.23.122.246 to 69.79.104.1 as the variable routes are all contained within, as can be verified in Figure 4.3. One key observation in the graph is that there is no visible causality from an increase in delay leading to a path change. However, different paths have different delays, which the graph reflects as expected with changes in delay occurring simultaneously with path changes. Another key observation is that the graph validates the notion mentioned in the previous subsection (4.3) on routes persisting for several hours: the colored path shares indicate 100 % shares in most of the 8-hour intervals.

While only one example is presented in this report, the results are similar for all destinations and the same conclusion holds: from this data we are unable to establish a causality where an increase in delay between two points in a network would induce a change of route between these points.

4.5 Destination-specific differences in delay on a shared link

Do any links show systematically different delays for packets to different destinations?

Separate from the rest of the analysis, we compare delays that packets to different destinations

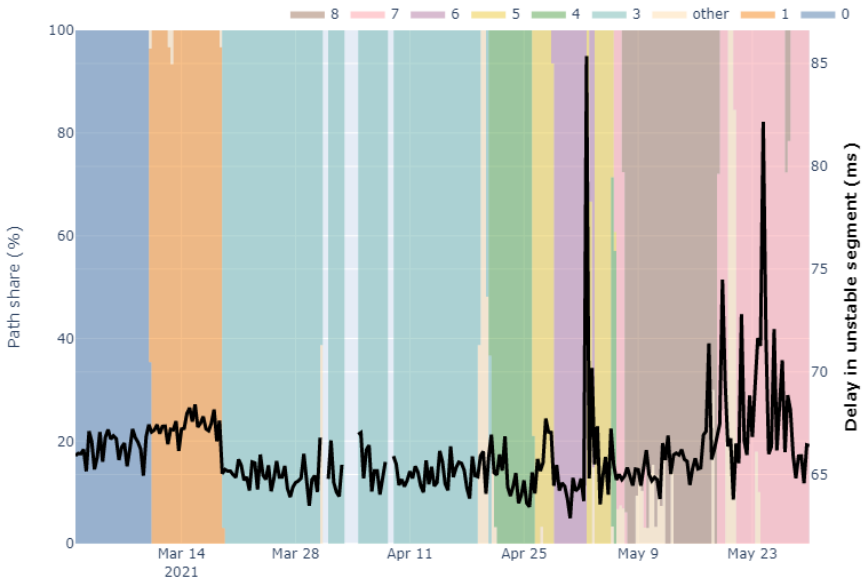


Figure 4.8: Share of distinct paths, represented by colors, to 45.65.244.254 over time between 94.23.122.246 and 69.79.104.1, the segment in which all path changes to this destination occur. The black line denotes delay in this segment (note the axis not starting at 0). Observe, for instance, the clear drop in delay when shifting from path 1 to 3 on March 19. After mid-May, delay and its variability increase with path 7, though at the end of May it seems to stabilize to low levels again without a path change. Rare paths with less than 100 occurrences are grouped under the overflow label "other". Interactive plot available.

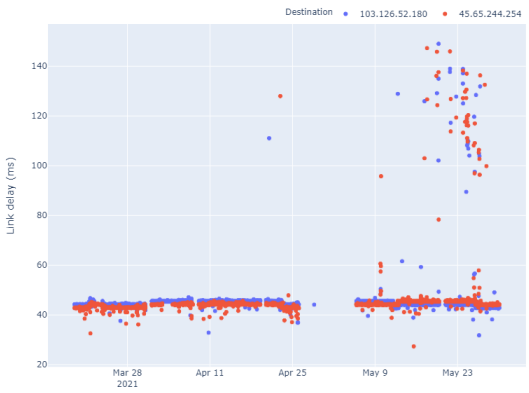
experience on shared links. The expectation is that links should treat packets equally and thus transmit them at equal rates. To this end, we consider delays at every link, grouped by destination. These are further classified into daily buckets—long enough to ensure large samples (about 50 to 100 observations per group) but short enough to also capture temporary effects.

We then employ the Kruskal–Wallis test[1] [12] to find any statistically significant differences in median delays between groups on each link on each day. Of the total of 395 links, 2 links show significant discrepancies in median delay for different destinations: p-values of the Kruskal–Wallis test range from 1.1×10^{-25} to 7.6×10^{-14}.

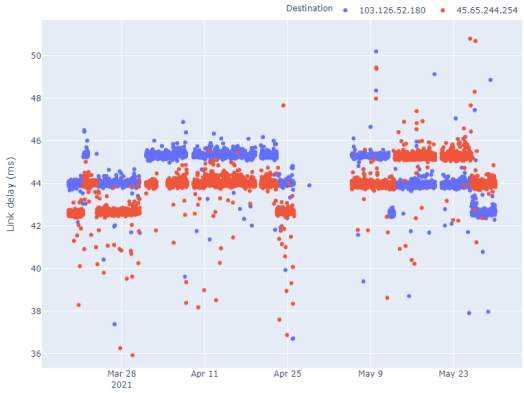
We explore these apparent differences further through plots. Figure 4.9 looks at one link serving two destinations where the difference persists throughout the three months and where, curiously, the sign changes from a certain point onwards. Figure 4.10 shows a similar difference for another link likewise serving two destinations; here the difference only lasts for 10 days at the beginning of the measurement period, but we do not know if this had already been the case prior to beginning the measurements.

Two links carry traffic to different destinations with systematically different delays for at least several days. The cause, however, remains unknown as we lack details of the network to investigate further.

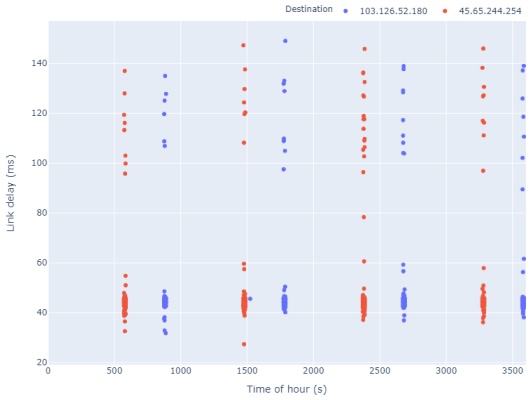
[1]The Kruskal–Wallis test is a non-parametric alternative to the parametric ANOVA, both of which test for equality of locations (median and mean, respectively) among several groups of observations.



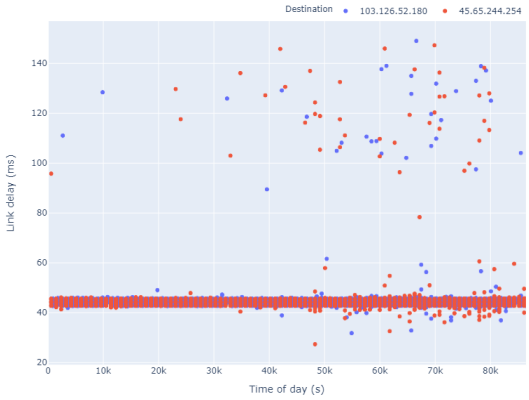
(a) Delays on link 91.121.131.74 to 37.187.36.199 for the two destinations using the link, 45.65.244.254 and 103.126.52.180. Between April 26 and May 5, this link is not on the path to these destinations except for one traceroute. Interactive plot available.



(b) Figure (a) zoomed to values of link delay roughly between 36 and 50 ms to more clearly showcase the discrepancy in delay between traceroutes destined to 45.65.244.254 and 103.126.52.180. Before May 12, packets to 103.126.52.180 travel systematically more slowly than those to 45.65.244.254, while after May 12 the roles are reversed.



(c) The data in Figure (a) modulo one hour. Traceroutes to both destinations consistently take place at four regular times in an hour, so the discrepancy in delay is not explained by hourly time correlation or interference in measurements.



(d) The data in Figure (a) modulo one day. Traceroutes to both destinations happen throughout the day, so the discrepancy in delay is not explained by daily time correlation.

Figure 4.9: Median delays towards 45.65.244.254 and 103.126.52.180 on link 91.121.131.74 to 37.187.36.199 show a discrepancy between the two destinations. The cause however remains unclear. Note the range of the vertical axes not starting at 0.

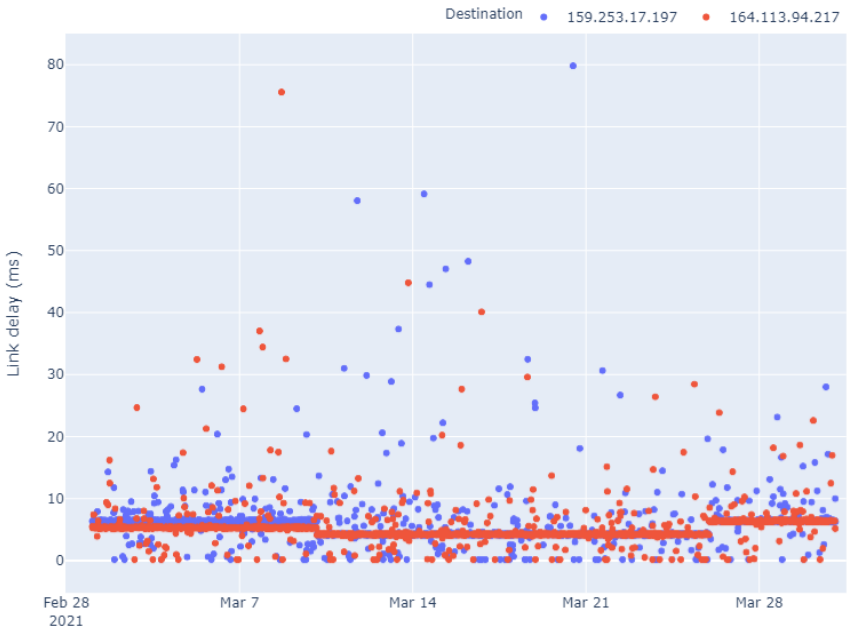


Figure 4.10: Delays on link 94.23.122.246 to 54.36.50.233 for the two destinations using the link, 159.253.17.197 and 164.113.94.217, during March. There is a discrepancy in delays until March 10, after which both destinations see the same delay on this link. Interactive plot available.

5 Conclusion

The goal of the project was to explore how process mining techniques may be applied in a networking context. To do so, we constructed a dataset based on publicly available traceroutes. While the contents of the dataset brought no groundbreaking revelations, we gained a solid picture of how process mining can be applied in practice and found that the domain of networking can indeed benefit from developments in process mining, especially in terms of quick and convenient visualizations.

Some caveats should nonetheless be noted regarding the practical use of the ProM tool and its plugins, which are mainly developed separately by various volunteers. First, this heterogeneous plugin ecosystem correlates with a heterogeneous level of documentation scattered in different locations. Second, there is weak support for saving configurations to reapply them at a later point in time on potentially different data, hindering scientific reproducibility. Similarly, exporting visualizations is poorly supported with the only option being low-resolution raster images of the visible screen. Lastly, scalability for datasets larger than a few tens or hundreds of megabytes appears to require considerable computational resources, depending on the plugin used. In the same vein, it should be added that ProM relies heavily on its graphical user interface and a headless environment has limited support and documentation. All of these considerations suggest a need for a redesigned tool based on modern practice. Specifically, a Python API for process mining would allow gaining wider adoption in the data science community by improving usability and compatibility with existing data science frameworks.

Potential future work with a dataset of the same format as the current one includes comparing paths between two hosts in both directions, comparing delays from two sources reaching a common router before continuing to the same destination, as well as comparing frequencies of routing changes inside and at borders of ASes.

Other, more general possible future work includes visualizing specific network characteristics of interest on top of a topology built with a process mining technique. This would be realized by targeting a specific process mining algorithm to reimplement under a framework compatible with requirements not readily met by ProM, such as online data processing.

References

[1] [n.d.]. RIPE Atlas. `https://atlas.ripe.net/`

[2] Kanak Agarwal, Eric Rozner, Colin Dixon, and John Carter. 2014. SDN traceroute: Tracing SDN forwarding without changing network behavior. In *Proceedings of the third workshop on Hot topics in software defined networking*. 145–150.

[3] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. 2020. PINT: probabilistic in-band network telemetry. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 662–680.

[4] Fábio Bezerra, Jacques Wainer, and Wil van der Aalst. 2009. Anomaly detection using process mining. In *Enterprise, business-process and information systems modeling*. Springer, 149–161.

[5] RP Jagadeesh Chandra Bose and Wil van der Aalst. 2010. Trace alignment in process mining: opportunities for process diagnostics. In *International Conference on Business Process Management*. Springer, 227–242.

[6] Benoit Donnet and Timur Friedman. 2007. Internet topology discovery: a survey. *IEEE Communications Surveys & Tutorials* 9, 4 (2007), 56–69.

[7] Cleiton dos Santos Garcia, Alex Meincheim, Elio Ribeiro Faria Junior, Marcelo Rosano Dallagassa, Denise Maria Vecino Sato, Deborah Ribeiro Carvalho, Eduardo Alves Portela Santos, and Edson Emilio Scalabrin. 2019. Process mining techniques and applications–A systematic mapping study. *Expert Systems with Applications* 133 (2019), 260–295.

[8] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. 2014. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*. 71–85.

[9] Peter TG Hornix. 2007. Performance analysis of business processes through process mining. *Master's thesis, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, Eindhoven* (2007).

[10] Bradley Huffaker, Daniel Plummer, David Moore, and KC Claffy. 2002. Topology discovery by active probing. In *Proceedings 2002 Symposium on Applications and the Internet (SAINT) Workshops*. IEEE, 90–96.

[11] Changhoon Kim, Parag Bhide, E Doe, H Holbrook, A Ghanwani, D Daly, M Hira, and B Davie. 2016. In-band network telemetry (INT). *Tech. Spec* (2016).

[12] William H Kruskal and W Allen Wallis. 1952. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association* 47, 260 (1952), 583–621.

[13] Earl Lawrence, George Michailidis, Vijayan N Nair, and Bowei Xi. 2006. Network tomography: A review and recent developments. *Frontiers in statistics* (2006), 345–366.

[14] Sander JJ Leemans. 2020. visual Miner. (2020). `http://leemans.ch/publications/ivm%20ProM%206.10.pdf`

[15] Sander JJ Leemans, Erik Poppe, and Moe T Wynn. 2019. Directly follows-based process mining: Exploration & a case study. In *2019 International Conference on Process Mining (ICPM)*. IEEE, 25–32.

[16] Ronny S. Mans, Wil M. P. van der Aalst, and Rob J. B. Vanwersch. 2015. *Process Mining in Healthcare: Evaluating and Exploiting Operational Healthcare Processes* (1st ed. ed.). Springer, Cham.

[17] Jorge Munoz-Gama. 2016. *Conformance checking and diagnosis in process mining*. Springer.

[18] Joyce Nakatumba and Wil van der Aalst. 2009. Analyzing resource behavior using process mining. In *International Conference on Business Process Management*. Springer, 69–80.

[19] Carl Adam Petri. 1962. Kommunikation mit automaten. (1962).

[20] Marcella Rovani, Fabrizio M Maggi, Massimiliano De Leoni, and Wil van Der Aalst. 2015. Declarative process mining in healthcare. *Expert Systems with Applications* 42, 23 (2015), 9236–9251.

[21] Anne Rozinat, Ivo SM de Jong, Christian W Günther, and Wil van der Aalst. 2009. Process mining applied to the test process of wafer scanners in ASML. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 39, 4 (2009), 474–479.

[22] Vladimir Rubin, Christian W. Günther, Wil M. P. van der Aalst, Ekkart Kindler, Boudewijn F. van Dongen, and Wilhelm Schäfer. 2007. Process Mining Framework for Software Processes. In *Software Process Dynamics and Agility (Lecture Notes in Computer Science)*, Qing Wang, Dietmar Pfahl, and David M. Raffo (Eds.). Springer, Berlin, Heidelberg, 169–181. https://doi.org/10.1007/978-3-540-72426-1_15

[23] Neil Spring, Ratul Mahajan, and David Wetherall. 2002. Measuring ISP topologies with Rocketfuel. *ACM SIGCOMM Computer Communication Review* 32, 4 (2002), 133–145.

[24] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. 2016. Simplifying datacenter network debugging with pathdump. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 233–248.

[25] Wil van der Aalst. 1998. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems and Computers* 08, 01 (1998), 21–66. https://doi.org/10.1142/S0218126698000043 arXiv:https://doi.org/10.1142/S0218126698000043

[26] Wil van der Aalst. 2016. Process Mining: Data science in Action — Coursera. https://www.coursera.org/learn/process-mining/

[27] Wil van der Aalst, Arya Adriansyah, and Boudewijn van Dongen. 2012. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2, 2 (2012), 182–192.

[28] Wil van der Aalst, Ton Weijters, and Laura Maruster. 2004. Workflow mining: Discovering process models from event logs. *IEEE transactions on knowledge and data engineering* 16, 9 (2004), 1128–1142.

[29] Boudewijn F van Dongen, Ana Karla A de Medeiros, HMW Verbeek, AJMM Weijters, and Wil van der Aalst. 2005. The ProM framework: A new era in process mining tool support. In *International conference on application and theory of petri nets*. Springer, 444–454.

[30] Da Yu, Yibo Zhu, Behnaz Arzani, Rodrigo Fonseca, Tianrong Zhang, Karl Deng, and Lihua Yuan. 2019. dShark: A general, easy to program and scalable framework for analyzing in-network packet traces. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*. 207–220.

[31] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, et al. 2021. LightGuardian: A Full-Visibility, Lightweight, In-band Telemetry System Using Sketchlets.. In *NSDI*. 991–1010.

[32] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. 2015. Packet-level telemetry in large datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 479–491.