Network Path Integrity Verification using Deterministic Delay Measurements

Alfred Arouna
SimulaMet
Oslo, Norway
alfred@simula.no

Steinar Bjrnstad
SimulaMet
Oslo, Norway
steinar@simula.no

Stein Jrgen Ryan
Prima Software
Oslo, Norway
s.j.ryan@vikenfiber.no

Thomas Dreibholz
SimulaMet
Oslo, Norway
dreibh@simula.no

Sobia Rind
SimulaMet
Oslo, Norway
sobia@simula.no

Ahmed Elmokashfi
SimulaMet
Oslo, Norway
ahmed@simula.no

***Abstract*—If an attacker succeeds in planting a device in a network, detecting the alien device can be extremely difficult. With the intuition that every device on the data path contributes to the end-to-end delay, we propose a simple and deterministic measurement-based approach for detecting the insertion of a layer-2 switch on the data path of a network operator. For this purpose, we use commodity hardware and the standard ping tool for collecting ICMP RTTs. To minimise inaccuracies in the measurements, we increase timing determinism on both ICMP source and target by using a real-time kernel on both, a dedicated source (a Linux server) and target (an RPI4 with custom image). Additionally, we manipulate real-time attributes for prioritising the ping process. By using this approach on different loaded networks: lab, campus network, research and education network and an ISP, we are able to reliably detect that a switch was added at the end of the path or within it. Our method yields an excellent performance on networks with considerable cross traffic as well as lightly loaded networks.**

***Index Terms*—network, deterministic RTT, alien switch**

I. Introduction

Protecting against man-in-the-middle attacks is key for ensuring end-to-end security and privacy. While such attacks are often executed using spyware and malware, there have been a number of reports of physical attacks [1]–[4]. These attacks involve the insertion of an alien device into the network that can sniff or manipulate traffic often at layer one or two. Networks that span several physical facilities can be exposed to such incidents. Hardening access to facilities and constant surveillance remain today's prime approaches for defending against physical attacks. However, if an attacker succeeds in planting a device, detecting the alien device can be extremely difficult. In this paper, we propose a measurement-based approach for detecting the insertion of a device in the data path. The key intuition behind our approach is the fact that every device on the data path contributes to the end-to-end delay and delay variation. This intuition applies to physical path changes as well as to the insertion of switch elements that operate at layer-2 and layer-3.

Detecting small changes in delay, however, requires measurement tools with deterministic performance. Current tools typically run in the user space of operating systems (OSes) and are thus subject to jitter related to the way OSes prioritise and schedule different processes [5], [6]. We, therefore, employ real-time Linux kernels on commodity hardware to increase the determinism of Internet Control Message Protocol (ICMP) ping measurements. Having consistent delay measurements allows us to flag changes in network paths in a statistically significant way. This accuracy holds for a broad range of scenarios that involves controlled lab settings, campus LAN and various combinations of WAN. It also holds for insertion of alien devices at different points in the topology. We believe that our approach can be easily deployed to automatically monitor topological changes. We hypothesise that network operators have full control and knowledge of their infrastructure. Thereby, unplanned RTTs changes can be linked to suspicious activity.

The rest of this paper is organised as follows. After introducing switching background in section II, we discuss related work in section III and present our approach and methodology for detecting path changes in section IV and subsequently evaluate the proposed method in section V. In section VI, we discuss our results before we conclude with section VII.

II. Background

In fixed line networks, signals are typically propagated through Ethernet and optical fibre networks. Any change in an all optical path will cause a change in the delay. The delay change will however be a fixed change because optical network elements do not involve electronic processing or buffering. The delay change for the optical path entirely depends on the length change of the path, i.e. 5 µs/km. Internal switching in a node-room, involving only a few metres of change in the path, will therefore impact the delay change in the network path in the order of nanoseconds.

On the other hand, for layer-2 and layer-3 switches, delay will typically vary. For example, a typical layer-2 Ethernet switch performs a set of operations when switching each packet. The switching fabric, which connects incoming ports with outgoing ports, copies packets, determines outgoing ports and then moves packets between queues. Hence, these operations involve buffering in electronic memories as well as per packet address lookup operations and processing. The time it takes to complete these operations will typically vary and depend on the switch hardware architecture, its firmware and network load. Hence, any new switch added in a network path adds a variable delay to the end-to-end delay.

978-3-903176-47-8 ©2022 European Union

The proposed method in this paper relied on the variability of Ethernet switches delay. For instance, a benchmark measurement using hardware-based accuracy device tester, shows – for 10 runs with 64 bytes each – an average of 100 s and 5 ns of latency and jitter respectively for the Mikrotik Gigabit Ethernet switch used in our experiments. The added delay may however be masked by cross traffic as well as measurements artefacts. Hence, for detecting such changes, we need a measurement methodology that both minimise measurements artefacts, e.g. timing inaccuracy, and control for cross traffic.

III. Related Work

The Transmission Control Protocol (TCP) is the most commonly used transport-layer protocol that meets the requirements of many applications [7]. Aikat *et al.* have shown that TCP RTTs values within a connection vary widely [8]. Moreover, the *syn* based method was concluded to provide a good estimate of RTTs that can be exploited in real time analysis of Internet traffic [9]. However, compared to ICMP, TCP is not always the most applicable transport for time-critical applications [7].

Pelsser *et al.* found cases in which ping gave very poor estimates of delay and recommended the use of an adaptation of paris-traceroute which supports delay and jitter estimation [10]. Marchetta *et al.* [11] focused on an IP Pre-specified Timestamp based approach using a single packet to dissect the RTT in chunks mapped to specific portions of a path. They showed that the proposed approach can be applied on more than 77% of the considered paths. On a security perspective, it is possible to develop an efficient ICMP-based algorithm protocol to detect malicious hosts that are performing ARP spoofing attacks [12]. Moreover, Arote *et al.* [13] proposed a technique based on ICMP and voting to detect and prevent Man in The Middle (MiTM) based ARP poisoning.

Another approach to prevent MiTM, is to identify vulnerable routers or abnormal behaviour router in the network [14], [15]. A tool like NMAP (Network Mapper) [16], can help for fingerprinting recognition of routers but has low accuracy. Furthermore, machine learning techniques can be combined with traditional approaches. In [17], the authors proposed Vesper, a tool for detecting a MiTM via ping echo analysis using an artificial neural network (ANN) based on autoencoders [18]. For finding the change or abnormal behaviour in the network, Vesper sends a burst of 50 pings that is modulated according to the MLS (Maximum Length Sequence). It then extracts a set of features to describe the response time series and feed them to the anomaly detector.

Vesper was tested in a LAN and for detecting the insertion of single or multiple switches. In our work we are using a simpler method for delay fingerprinting identifying changes in the network path by performing deterministic delay measurements using a Linux server and an RPI as the target. RPI is well suitable for measurements [19] and is widely used on large scale measurements such as CAIDAs Archipelago infrastructure [20]. Additionally our measurements are not limited to the LAN environment. We perform experiments in several network environments: lab (LAN), campus, research network and an ISP.

IV. Approach and Methodology

We leverage end-to-end measurements, using ICMP ping, to characterise the delay profile of a network path. Subsequently, we insert an Ethernet layer-2 switch and verify whether the new delay profile is statistically different from the baseline.

A. Delay Measurements Determinism

We assume that, given that the network setup is stationary, end-to-end delay measurements will be stable. Monitored paths are under network operators control with full knowledge of the infrastructure and planned changes. Thus unplanned RTTs change – for any traffic profile – may be suspicious. This, however, implies that we are able to control for and minimise effects that are related to the measurement tools. For our measurements, we use the Linux Operating System (OS). By default, the Linux kernel executes processes in a single execution flow (monolithic structure). This leads to delays and inaccuracies in the timing of process scheduling. According to [21], OS latency can be caused by major factors such as scheduling jitter and non-preemptable sections. By assigning the highest real-time priority to a process, the scheduling jitter can be minimised. With kernel or device driver non-preemptable sections disabling preemption, processes with high real-time priority are exposed to extra delays since they can not interrupt. For server applications, as well as for most desktop application use-cases, this is not a challenge. However, such unbounded delays cause jitter for time-critical applications. Particular examples of such applications are music recordings [21], but also fine-granular delay measurements. If, for example, a packet for delay measurement arrives while the preemption is disabled, the time until finishing the currently running tasks will be added to the measured delay.

Real-time ping. We can use a real-time Linux kernel for ensuring deterministic execution time. There are different variants of real-time kernel, including but not limited to real-time micro-kernel (RTLinux) [22] and the RT Linux with the CONFIG_PREEMPT_RT[1]. RTLinux requires a kernel space real-time task to be scheduled via the micro-kernel and real-time applications to be adapted to the micro-kernel. CONFIG_PREEMPT_RT modifies the Linux kernel itself to remove unbounded delays caused by locking and realises interrupts as kernel threads. We adopt CONFIG_PREEMPT_RT, because it provides the most widespread real-time Linux approach. With the real-time Linux kernel, it is also possible to manipulate the real-time attributes of a process. The ping process is set to a high priority (90); eliminating scheduling jitter; and the OS automatically set the process scheduling policy to `SCHED_RR`[2]. This policy is derived from `SCHED_FIFO`

[1] CONFIG_PREEMPT_RT Patch Set: https://rt.wiki.kernel.org/index.php/CONFIG_PREEMPT_RT_Patch.

[2] https://wiki.linuxfoundation.org/realtime/documentation/technical_basics/sched_policy_prio/start

and guarantees fair sharing of CPU time between tasks. This ensures that the ping process will – mostly – not be subject to blocking due to a held lock.

With ping involving two independent systems (i.e. the source and destination), our measurements require improving timing determinism on both ends. To this end, we have built a custom Raspberry Pi 4 Model B (RPI4) image, optimised with minimal packages necessary for the RPI4 to reply to ICMP requests. To build this image for embedded systems, we rely on Yocto and use only two recipes: `yocto/zeus` and `yocto-pinger`[3] from the Open Source Distribution (OSD) project. Our image is mainly based on Yocto Board Support Package (BSP) for Linux on Raspberry Pi[4]. The Yocto BSP layer for RPI provides recipes for RPI4 kernels and we use real-time kernel version 4.19. Instructions to build the image are publicly available[5].

Probe packets. Before assessing the impact of real-time probing, we need to determine the optimal size(s) for probe packets. Different network devices vary in the way they process, fragment and reassemble packets, because of differences in their Application-Specific Integrated Circuit (ASIC)s and firmware. These differences can help inferring the addition of new devices. We perform experiments with a wide range of packet sizes, both in the lab and on a campus LAN. In the lab, the RPI is a) directly connected and b) connected through a switch to the ICMP source. For the campus network experiments, the RPI is placed in an office on the same floor as the ICMP source.

Figure 1 shows the distribution of Round Trip Time (RTT)s for different probe packet sizes with non fragmented payload in grey. Our measurements show that the largest allowed packet size in the campus network is 9600 bytes. Packets larger than this limit result in 100% packet loss. As expected, the RTT increases as the payload increases but not linearly. There are noticeable jumps at sizes that require further fragmentation. Based on these results, we selected five packet sizes for measurements: 56, 1400, 4000, 6800 and 9600 bytes. The first two correspond to packets that will not be fragmented, while the latter three will be fragmented and coincide with the jumps - every 2800 bytes - in Figure 1. We hypothesise that switches handle fragmentation differently, and hence the three large packet sizes can help revealing the presence of new devices.

Evaluating real-time ping. We perform two experiments to verify that using a real-time kernel can indeed result in deterministic delay measurements. We connect the ICMP target (i.e. RPI4) and source, which is a Dell PowerEdge R340 in a simplified topology directly to each other. For each selected payload in both experiments, we sent 10 times, 1000 ICMP packets from the source to the RPI4 using the default ping interval, i.e., every second.

The first experiment compares the default RPI OS[6] to the custom image with real-time kernel, while configuring the ICMP source with a real-time kernel. More specifically, we assign the ping process the highest priority and use the default scheduling policy of `SCHED_RR`. Figure 2a shows that the custom image running on the RPI4 provides more deterministic RTTs (less variability) than the default Raspberry Pi OS. Thus the median of the coefficients of variability for all payloads is 0.0997 and 0.0782 for the default Raspberry Pi OS and the custom image, respectively.

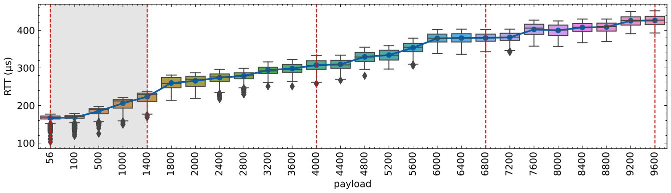


Fig. 1: **Preliminary experiments on the campus network**. Payloads 56, 1400, 4000, 6800 and 9600 have been selected for the rest of the measurements.

The second experiment compares the generic kernel on the source to the real-time one, while running the custom image on the RPI4. More specifically, the source use the generic kernel 5.6.19-generic[7]. For real-time kernel, it loads a patched version of the same kernel, that is its patched PREEMPT_RT version 5.6.19-rt11[8]. At the time of our measurements, kernel version 5.6.19 was the latest stable kernel (with real-time version) compatible with Ubuntu 20.04 LTS. Figure 2b shows the RTT PDF per payload for different kernels running on the source. The RTT timings are more consistent (higher peaks) when using the PREEMPT_RT kernel for all payloads (except for 6800 bytes) with similar median coefficients of variability for all payloads.

In both experiments, using the PREEMPT_RT kernel provides the most consistent measurements with the lowest delay variations. This matches our goal of minimising delay variations in our measurements. The RT OS provides the most deterministic measurements, but not always the lowest RTTs. This is consistent with [23], stating that real-time provides guarantees, not low speed. Previous studies show that real-time kernel comes at the cost of performance degradation [24], [25]. In addition, `PREEMPT_RT` adds a typical jitter of 50s on x86 architecture and is recommended for audio and media [26].

B. *Detecting changes on the path*

After configuring our measurement setup to provide stable delay estimates, we collect end-to-end delay measurements and continuously compare their distributions to flag statistically significant changes. These changes can point to the presence of an alien layer 2 device or a path change. To this end, we employ the two-sided KolmogorovSmirnov (KS) statistical test, for comparing samples [27]. Although comparing percentiles of RTT samples seems easier, it requires the

[3] https://prima.svn.beanstalkapp.com/osd/trunk/

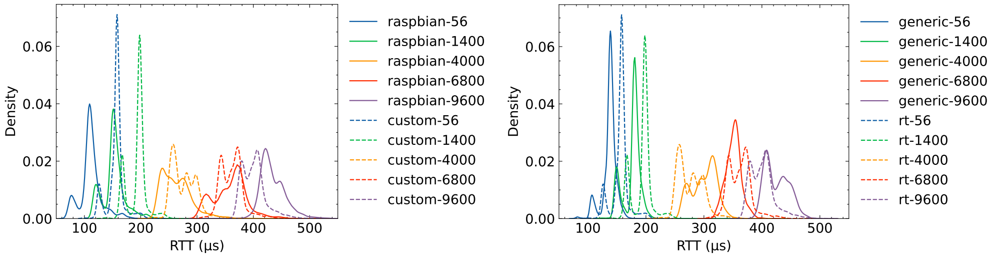
[4] https://github.com/agherzan/meta-raspberrypi/archive/zeus.zip

[5] https://github.com/simula/deterministic-rtt

[6] March 4th 2021 release

[7] https://mirrors.edge.kernel.org/pub/linux/kernel/v5.x/linux-5.6.19.tar.gz

[8] https://mirrors.edge.kernel.org/pub/linux/kernel/projects/rt/5.6/patch-5.6.19-rt11.patch.gz



(a) Custom vs. Raspbian on target

(b) RT Linux vs. Generic on source

Fig. 2: **RTTs distribution based on kernel/OS.** On the source, we compared the Generic and the RT kernel of the same OS. On the target (RPI), we compared the default Raspbian against our custom image (with RT kernel). In both cases, custom image and RT Linux provide more consistent RTTs. For optimising for deterministic delays in the measurement equipment, we use a custom image on the target (RPI) and RT Linux on the source, for the rest of the measurements.

introduction of variable thresholds from a scenario to another. KS measures the distance between the sample distribution and a baseline distribution at a certain confidence level, whether the two samples come from different distributions. The test assumes that the two distributions are identical (null hypothesis). To accept or reject this null hypothesis, we use a range of levels of significance or thresholds: 0.01 (1%) to 0.1 (10%) which include the common 5%. We reject the null hypothesis, if the test p-value is below the threshold, that is the distance between the two distributions can not be attributed to random variations. Since this is a binary classification (accept or reject), it is possible to reduce the results, when running the test multiple times, to a confusion matrix.[9]

Definition 1: The confusion matrix is composed of two dimensions: *actual* and *predicted*. *actual* is our ground truth about the state of the path, which can be same (i.e. no path change) or different (i.e. a layer-2 switch is inserted or there is a reroute). *predicted* is the result of comparison between two distributions. Positive implies that a change is detected. A True Positive (TP) is recorded if *actual* is different and also the *predicted* shows a different path. If the *actual* is same and the *predicted* shows different, the result is a False Positive (FP). False Negative (FN) occurs when the *actual* is different and the *predicted* shows the same path. True Negative (TN) means that the *actual* path is the same and the *predicted* is also the same.

We use the full RTTs distribution collected per payload and per path as an input to the KS when comparing different paths. For same path comparisons, we randomly split the dataset into two subsets 1000 times and compute the KS per random split.

For both different and same paths, the p-value is evaluated according to the thresholds and we compute the ratio of TP, TN, FP and FN. Accordingly, one can set thresholds for the TP and TN rates to infer path changes and no path changes, respectively. In this sequel, instead of picking such a threshold, we show the ratios. This approach works very well, when comparing different paths, in the lab and on the campus network but the ratio of False Negative (different *actual* and same *predicted*) increases with the noise level (i.e. cross traffic). We discussed cross traffic impact on False Negative vs. False Positive in section VI. The added queuing delay then becomes a major contributor to the overall roundtrip delay. One way to minimise FN results, is to consider only packets that experience minimum delay, which we call *lucky* packets.

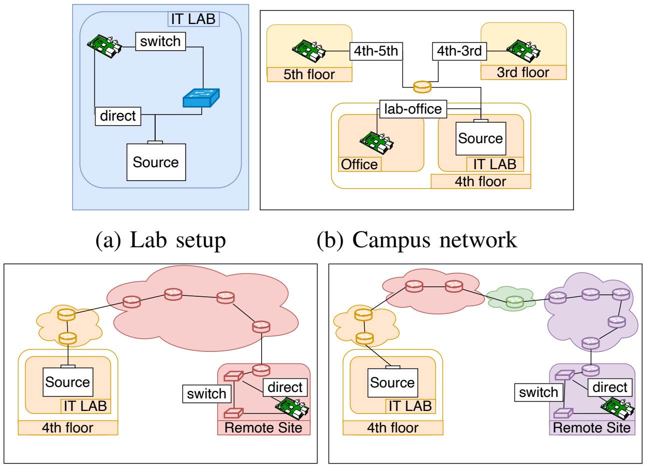
Definition 2: A *lucky* packet is a packet forwarded through a network path without being subject to queuing delays caused by contention with other packets. In a busy network, the *lucky* packets are expected to have the lowest RTTs.

To identify the appropriate percentage of *lucky* packets to minimise FN results in busy networks, we tried several ratios of lowest RTTs (10% to 90%) with regard to the expected results: True Negative for identical path subset and True Positive for different paths.

We used paths with cross traffic (ISP) and paths without cross traffic (lab). In both cases, we compared *direct* and *switch* as in Figure 3a for the lab and Figure 3d for the ISP. We noticed that, starting from 50%, the percentage of predicted True Positives continuously drops for the noisy networks. Wide area connections carry significant traffic thus our packets will likely experience added queuing delays. Therefore, we use the limit of 40% of *lucky* packets to evaluate identical and different paths prediction. This ratio of *lucky* packets can be adjusted according to network operator traffic profiles.

In summary, our approach involves the following steps. First, we collect deterministic round trip measurements between two devices that run real-time Linux, which we place at the beginning and end of the segment that we are interested in monitoring. Due to security and network usage policy, we were not allowed to add our own switch on the campus network and have only access to the source and the target (RPI). The campus network is a case where the switch is in the middle of the network path as in Figure 6. Second, we pick the luckiest 40% packets, in terms of round trip delay, from each test and compare delay distributions across tests using the KS test. Lastly, after conducting n measurement rounds, we estimate the corresponding similarity rates (i.e. our confusion matrix), which we use to infer changes on the path.

[9] We use the python scipy package implementation https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ks_2samp.html.



(a) Lab setup (b) Campus network

(c) Research Network (d) ISP

Fig. 3: **Measurement trials**. Two main setup: a) lab trial, field trials with b) campus network trial, c) research and education network trial and d) ISP trial. Each router represents a layer-3 hop.

TABLE I: Source and target

	Source	**Target**
Hardware	Dell PowerEdge R340	Raspberry PI 4 (4G)
OS	Ubuntu 20.04.2 LTS	OSD Linux 1.0
Kernel	5.6.19-rt11	4.19.71-rt24
Tool	ping (iputils s20190709)	NA
Priority	90 (`SCHED_RR`)	NA

V. Evaluation

Next, we evaluate the viability of using accurate end-to-end delay measurements for detecting insertions of a single alien switch or path changes.

A. Measurement setup

In all experiments, we use a Dell Server as a source and a RPI4 with our custom image as a target (see Table I for their specifications). Further, we use a widely affordable and off-the-shelf switch: MikroTik RB1000. The switch is inserted - on the part of the network under our control - to emulate the addition of an alien switch. We perform identical measurements with a Juniper EX3300 resulting in similar outcomes. Due to space limitation, we show only the results from the Mikrotik.

We evaluate our approach in four different scenarios, which we illustrate in Figure 3. These are a) a controlled lab setup, b) a campus LAN, c) a WAN over the same Research and Education Network and d) a WAN that crosses two different ISPs. Cases c) and d) allow us to evaluate the effectiveness of our approach. Each scenario involves two types of experiments: : α) *direct* where the RPI4 is directly connected to the network and β) *switch* where we add one – under control – switch on the path between the RPI4 and the source.

All measurements are automated using bash scripts to send ICMP `ECHO_REQUEST` with following parameters: a) default – 1s – ping interval; b) 10 experiments per payload; c) variable payloads from 56, 1400, 4000, 6800 and 9600 per experiment; d) 1000 ICMP packets per experiment. In total, we collect 450k RTTs: 100k RTTs from the lab, 150K RTTs from the campus network, 100k RTTs from the research and education network and 100k RTTs from the ISP. Finally, we use the approach we describe in subsection IV-B, with a lucky packet threshold of 40%, for measuring inference accuracy.

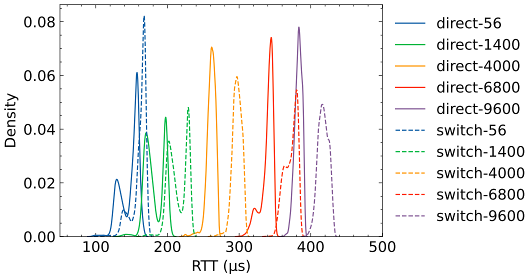


Fig. 4: **Lab direct vs. switch measurements**. As expected the Mikrotik switch adds some delay to the RTTs and the two distributions are different.

TABLE II: **Percentage of False and True outcomes per experiments in the lab.** These results are promising showing that our approach succeeds in flagging same and different paths.

	Payloads									
	56		**1400**		**4000**		**6800**		**9600**	
	FP	**TN**	**FP**	**TN**	**FP**	**TN**	**FP**	**TN**	**FP**	**TN**
D-D	2	98	3	97	3	97	3	97	3	97
S-S	3	97	3	97	3	97	3	97	3	97
	FN	**TP**	**FN**	**TP**	**FN**	**TP**	**FN**	**TP**	**FN**	**TP**
D-S	0	100	0	100	0	100	0	100	0	100

B. Lab Trial

The lab setup represents the simplest case without cross traffic, since we fully control ongoing traffic. While adding the switch, fragmented payloads (payloads higher than the MTU) inflates latency by a maximum of ≈ 50 s as in Figure 4. This makes discriminating the two distributions easier, a fact that is further confirmed by the KS based scoring of accuracy. For all payload sizes, the p-value is consistently lower than 0.01. Thus, we reject the null hypothesis that the two distributions come from the same population.

Table II summarises the scores we obtained from the lab tests, where we look at two cases: direct connectivity (D) and connectivity in presence of an alien switch (S). In total, we have three combinations. First, (D-D), here all comparisons are expected to yield a 100% match, since the path has not changed, that is 100% True Negative (TN). Second, (S-S) which is similar to (D-D) in that we expect 100% TN. Finally, (D-S), which should yield a 100% TP. The values in the table indicate we are able to achieve 100% recall as far as the third combination is concerned. We only mistake 3% of the first two cases for the third one. These results are promising showing

TABLE III: **Percentage of False and True outcomes per experiments at campus network.** Comparing the same experiments subsets show a high ratio of TN. Comparing different paths provides expected results (TP) except for 3rd and 5th floors. According to campus network network topology, 5th and 3rd floor share the same virtual switch.

	Payloads									
	56		**1400**		**4000**		**6800**		**9600**	
	FP	**TN**	**FP**	**TN**	**FP**	**TN**	**FP**	**TN**	**FP**	**TN**
3-3	3	97	4	96	3	97	2	98	3	97
4-4	2	98	2	98	2	98	3	97	3	97
5-5	3	97	4	96	2	98	4	96	3	97
	FN	**TP**	**FN**	**TP**	**FN**	**TP**	**FN**	**TP**	**FN**	**TP**
4-3	0	100	0	100	0	100	0	100	0	100
4-5	0	100	0	100	0	100	0	100	0	100
5-3	10	90	100	0	50	50	40	60	0	100

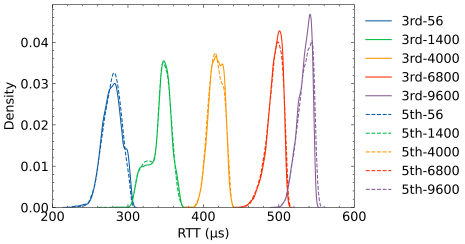


Fig. 5: **Campus Network 3rd vs 5th floor**. RTTs distribution appears very similar, but median KS p-value trends differ with regards to fragmentation.

that our approach succeeds in flagging the insertion of layer-2 devices on the path.

C. Field trial

We focus on three scenarios with an increasing level of uncertainty due to cross traffic and topology complexity: campus network, WAN over the same research and education network and a WAN through two ISPs.

1) Campus Network: Our campus network measurements are collected from three different floors in the same building. With this production grade network, we are not allowed to insert our own switch in the network. We rely on the switching infrastructure provided by the campus network. The campus network presents a case where the switch is not - inserted - at the end of the network path. The ICMP source is located on the 4th floor and the ICMP target was connected at different locations: another office on the 4th, 5th and 3rd floor, respectively. Table III presents the scoring results ratio for all comparisons. We compare measurements within each individual floor and across floors. The idea is to detect cases where the ICMP target was connected to two different floors. Accordingly, measurements from the same floor should yield a high TN rate, while measurements from different floors should yield a high TP rate. The first five cases confirm this. Comparing the 3rd and 5th floors, however, gives mixed results.

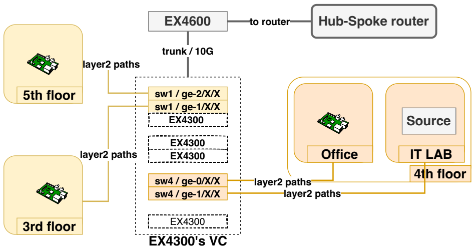


Fig. 6: **Campus Network simplified diagram**. The 3rd and 5th floors are on the same network range while the 4th floor is on another. All switches are part of the same virtual chassis.

To better understand this, we take a closer look at the measured RTTs as well as how they compare to each other. Figure 5 compares the distribution of the luckiest 40% of all packets for the 3rd and 5th floors and for different payload sizes. Apart from the payload size of 1400 bytes, we observe differences between the two distributions albeit very slight.

According to the campus network topology, these two floors are managed by two different physical Juniper switches that are combined in a virtual chassis (see Figure 6). The setup is almost identical, however, using a range of payloads can help flagging the slight differences. This is especially true when using payload sizes that are likely to be fragmented.

2) Research and Education Network (REN) : To verify whether our approach will succeed beyond the relatively simple cases above, we experiment with wide area paths over a Research and Education Network (REN). Here, we collect RTTs between the campus network lab (with the source) and a remote site (under our control with the target). More specifically, we have carried out two types of experiments: *direct* where the RPI is directly connected to the main switch at the remote site and *switch* where an additional switch is added between the main switch at the remote site and the RPI. Running traceroute between the two sites shows that there are seven routers in between. Table IV shows the success percentage when comparing Direct vs. Direct, Switch vs. Switch and Direct vs. Switch, respectively. For the first two cases, we predominantly detect that the compared distributions are from the same population; that is no network change. For the third case, we also correctly detect the insertion of the switch; that is the compared distributions come from different populations. Hence, we are able to detect the insertion of an alien device even when the network path is long and involves several intermediate layer 2 and layer 3 devices.

3) ISP: From the campus network to the remote site, we rely also on another ISP, ISP A, which peer with the research and education network at an Internet eXchange Point, which is depicted by the green cloud in Figure 3d. Note that we have a link to ISP A terminated at the remote site. This network path is longer than the Research and Education Network and is expected to be more noisy.

We run similar measurements as the ones from the Research and Education Network, comparing the same paths and different paths. The first two cases confirm the expected results

TABLE IV: **Percentage of False and True outcomes per experiments from research and education network .** High ratio of expected results for both same and different paths comparisons.

	Payloads									
	56		1400		4000		6800		9600	
	FP	**TN**	**FP**	**TN**	**FP**	**TN**	**FP**	**TN**	**FP**	**TN**
D-D	4	96	3	97	2	98	2	98	2	98
S-S	2	98	2	98	2	98	2	98	3	97
	FN	**TP**	**FN**	**TP**	**FN**	**TP**	**FN**	**TP**	**FN**	**TP**
D-S	0	100	0	100	0	100	0	100	0	100

TABLE V: **Percentage of False and True outcomes per experiments from ISP.** High ratio of expected results for both same and different paths comparisons.

	Payloads									
	56		1400		4000		6800		9600	
	FP	**TN**	**FP**	**TN**	**FP**	**TN**	**FP**	**TN**	**FP**	**TN**
D-D	4	96	2	98	3	97	2	98	2	98
S-S	2	98	2	98	2	98	2	98	2	98
	FN	**TP**	**FN**	**TP**	**FN**	**TP**	**FN**	**TP**	**FN**	**TP**
D-S	0	100	0	100	0	100	0	100	0	100

from comparing the same path: high ratio of True Negative. Comparing *direct* vs. *switch* yields 100% of True Positive as expected (Table V). The lucky packets approach minimises as much as possible RTTs artefacts related to cross traffic and provides consistent results while comparing the same path or different paths.

4) Research and Education Network direct vs. ISP Direct: From our campus network to the remote site, we used two different paths: the first path goes through the Research and Education Network and the second goes through ISP A. While this case can easily be mapped using traceroute, we can envision a scenario where the whole path is opaque, e.g. using MPLS, and we only control the end points. This test mimics such a scenario allowing us to check whether our method can detect that a different network is used between the same end points. Figure 7 shows the RTTs distribution from both paths. As expected, the ISP path with four more routers has higher RTTs for all payloads. The lucky packets comparison approach provides 100% of expected True Positive: both paths are clearly flagged as different for all payloads.

VI. Discussion

ICMP-related artefacts. Our measurement approach may be vulnerable to ICMP blocking and rate-limiting that some providers do [28]. However, our methodology can be used on an internal network while filtering external ICMP requests from the Internet. Yet, forward and reverse paths asymmetry can lead to False Positive and limit the accuracy of our method. But, we assume that both paths are under the operator control and any unplanned RTTs changes can be considered as suspicious. In addition, our methodology is using the default ping interval and commonly accepted payloads. Its impact on network performance is thus minimal and will likely bypass filtering rules which are typically volume/load/frequency based.

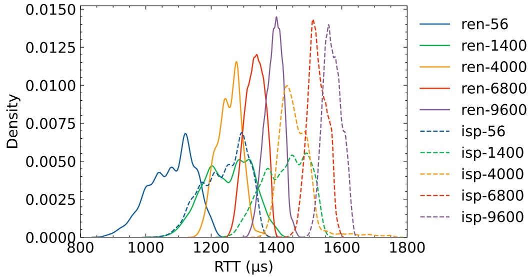


Fig. 7: **Research and Education Network direct vs. ISP direct**. ISP path produce higher RTTs and paths are different

Effects of measurement time. Measurements have been performed in different time-slots, including rush hours, off-peak hours, working days and weekends over the span of several months. These different measurement environments could significantly impact results from the campus network to the ISP through the research and education network. To minimise this impact, we have run each experiment ten times per payload. Moreover, the automated bash script goes sequentially through the chosen payloads. If the measured network is heavily congested for a long time, it will affect all our selected payloads for the current run. Similarly, rush hours measurement impact is shared across all payloads for the run. For paths with stable delay profiles, planned cross traffic will result in False Positives. However, it can lead to False Negatives for jittery paths. Thus, running measurements in off-peak hours can increase confidence in the obtained results.

Lucky packets. By using *lucky* packets, filtering of RTTs related to network high load/activity is achieved, leaving only RTTs that faithfully describe an unloaded measured path. This approach works well when comparing different paths, but its utility is marginal when comparing the same path. The repeated random splits helps to minimise corner cases where lower RTTs from normal traffic are in one data-subset and higher RTTs from high traffic are in the second data-subset.

Deployment considerations. Our approach runs on commodity hardware and does not require specialised equipment. Moreover, it is suitable for the stationary network of an operator. To deploy it, one may consider building out a network of probe pairs to cover network segments of interests. These segments can cover different locations and can help to monitor the integrity of the network paths by detecting unplanned RTTs changes. A practical use of the method may be to continuously run a series of measurements, e.g. once an hour, and then compare the delay characteristics of each series. Clearly, the choice of number of probes presents a trade-off between setup complexity and ability to narrow down locations of alien devices.

Multiple alien devices. It is expected that multiple alien devices on the same path will increase the RTTs latency and jitter. We can speculate that our approach could help to detect unplanned RTTs changes, but is limited as far as the evaluation of the number of inserted alien devices is concerned.

Measured paths. Both source and target used Ethernet ports and IPv4 addresses. All tested paths used wired links and covered a range of hops with the longest path having 11 hops as in Figure 3d. The campus network used Juniper switches with fiber optical links in the node room and we assumed that other network operators used similar infrastructure. Intuitively, using our approach on a shared wireless link will be challenging.

VII. Conclusions and Further Work

In this paper, we have developed a method for detecting the insertion of a switch in the network path from the perspective of a network operator. The method is based on using commodity hardware and the freely available ping tool. The ping process determinism is first increased. Secondly, we identified packet sizes suitable for the measurements. Lastly, we filtered out packets subject to high delay. We have evaluated our approach in a controlled lab as well as on different networks that include a campus local area network, wide area path across a research and education network as well as an end to end wide area path with two ISPs. Our results have demonstrated that we are able to reliably detect the insertion of an alien switch for all tested scenarios. The detection produces binary outcomes not focusing on identifying the type of switch inserted. Further work may include network device signature detection and/or fingerprinting. We also plan to extend our work by a) examining ping determinism on different real-time kernels and/or OS and/or network cards; b) looking at other paths changes detection methods e.g. tracking lucky packets over time to ease the implementation and reduce overhead; c) evaluating sensitivity accuracy with different protocols and multiple alien devices.

VIII. Acknowledgements

We would like to thank our shepherd Theresa Enghardt and the anonymous reviewers for their insightful comments. We are grateful to the network operators who allowed us to conduct our measurements on their production grade networks. This work was funded by the Norwegian Research Council grant # 288744 (GAIA: Digital vulnerability and national autonomy).

References

[1] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2027–2051, 2016.

[2] R. Barnes, B. Schneier, C. Jennings, T. Hardie, B. Trammel, C. Huitema, and D. Borkman, "Confidentiality in the face of pervasive surveillance: A threat model and problem statement," *Request for Comments: 7624*, 2015.

[3] "NSA spying row: Denmark accused of helping US spy on European officials," https://www.bbc.com/news/world-europe-57302806.

[4] "New Snowden Documents Show NSA Deemed Google Networks a Target," https://slate.com/technology/2013/09/shifting-shadow-stormbrew-flying-pig-new-snowden-\documents-show-nsa-deemed-google-networks-a-target.html.

[5] A. Dubey, G. Karsai, and S. Abdelwahed, "Compensating for timing jitter in computing systems with general-purpose operating systems," in *2009 IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*. IEEE, 2009, pp. 55–62.

[6] C. Stylianopoulos, M. Almgren, O. Landsiedel, M. Papatriantafilou, T. Neish, L. Gillander, B. Johansson, and S. Bonnier, "On the performance of commodity hardware for low latency and low jitter packet processing," in *Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems*, 2020, pp. 177–182.

[7] S. D. Strowes, "Passively measuring tcp round-trip times," *Communications of the ACM*, vol. 56, no. 10, pp. 57–64, 2013.

[8] J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay, "Variability in tcp round-trip times," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, 2003, pp. 279–284.

[9] S. Shakkottai, N. Brownlee, A. Broido *et al.*, "The rtt distribution of tcp flows on the internet and its impact on tcp based flow control," Citeseer, Tech. Rep., 2004.

[10] C. Pelsser, L. Cittadini, S. Vissicchio, and R. Bush, "From paris to tokyo: On the suitability of ping to measure latency," in *Proceedings of the 2013 conference on Internet measurement conference*, 2013, pp. 427–432.

[11] P. Marchetta, A. Botta, E. Katz-Bassett, and A. Pescapé, "Dissecting round trip time on the slow path with a single packet," in *International Conference on Passive and Active Network Measurement*. Springer, 2014, pp. 88–97.

[12] G. Jinhua and X. Kejian, "Arp spoofing detection algorithm using icmp protocol," in *2013 International Conference on Computer Communication and Informatics*. IEEE, 2013, pp. 1–6.

[13] P. Arote and K. V. Arya, "Detection and prevention against arp poisoning attack using modified icmp and voting," in *2015 International Conference on Computational Intelligence and Networks*. IEEE, 2015, pp. 136–141.

[14] Y. Vanaubel, J.-J. Pansiot, P. Mérindol, and B. Donnet, "Network fingerprinting: TTL-based router signatures," in *Proceedings of the 2013 conference on Internet measurement conference*, 2013, pp. 369–376.

[15] H. Pucha, Y. Zhang, Z. M. Mao, and Y. C. Hu, "Understanding network delay changes caused by routing events," *ACM SIGMETRICS performance evaluation review*, vol. 35, no. 1, pp. 73–84, 2007.

[16] G. F. Lyon, *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure. Com LLC (US), 2008.

[17] Y. Mirsky, N. Kalbo, Y. Elovici, and A. Shabtai, "Vesper: Using echo analysis to detect man-in-the-middle attacks in LANs," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1638–1653, 2018.

[18] W. Wang, Y. Huang, Y. Wang, and L. Wang, "Generalized autoencoder: A neural network framework for dimensionality reduction," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014, pp. 490–497.

[19] P. Membrey, D. Veitch, and R. K. Chang, "Time to measure the pi," in *Proceedings of the 2016 Internet Measurement Conference*, 2016, pp. 327–334.

[20] K. Claffy, Y. Hyun, K. Keys, M. Fomenkov, and D. Krioukov, "Internet mapping: from art to science," in *2009 Cybersecurity Applications & Technology Conference for Homeland Security*. IEEE, 2009, pp. 205–211.

[21] L. Abeni, A. Goel, C. Krasic, J. Snow, and J. Walpole, "A measurement-based analysis of the real-time performance of linux," in *Proceedings. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2002, pp. 133–142.

[22] V. Yodaiken and M. Barabanov, "A Real-Time Linux," New Mexico Institute of Technology, Tech. Rep., Jan. 1997.

[23] K. Yaghmour, *Building embedded Linux systems*. O'Reilly Japan, 2003.

[24] K. Koolwal, "Investigating latency effects of the linux real-time preemption patches (preempt rt) on amds geode lx platform," *RTLWS11*, 2009.

[25] K. Koolwal and R. ENGINEER, "Myths and realities of real-time linux software systems," in *Proc. Real-Time Linux Workshop (RTLWS 2009)*, 2009.

[26] W. Mauerer, "The Many Approaches to Real-Time and Safety-Critical Linux," http://events17.linuxfoundation.org/sites/events/files/slides/talk_10.pdf, Open Source Summit Japan 2017, 2017.

[27] G. Corder and D. Foreman, *Nonparametric Statistics: A Step-by-Step Approach*. Wiley, 2014.

[28] H. Guo and J. Heidemann, "Detecting ICMP rate limiting in the Internet," in *International Conference on Passive and Active Network Measurement*. Springer, 2018, pp. 3–17.