NetLoiter: A Tool for Automated Testing of Network Applications using Fault-injection

Michal Rozsíval
Faculty of Information Technology
Brno University of Technology
Email: michal.rozsival@vut.cz

Aleš Smrčka
Faculty of Information Technology
Brno University of Technology
Email: smrcka@fit.vut.cz

***Abstract*—The reliability of a network is a crucial requirement for applications and systems such as IoT (Internet-of-Things), cloud-based solutions, client-server, or peer-to-peer architectures. Unfortunately, real networks cannot be assumed to be fault-free, especially when considering various hardware problems, performance issues, or even malicious attacks. Testing network applications should include the evaluation of fault-tolerance of a system under various network conditions. The paper introduces a tool which helps developers and verification&validation practitioners easily analyse their network application's behaviour in unexpected network situations. The tool is based on man-in-the-middle and aims at network nodes communicating using a single network interface. The tool implements a fault-injection method; supported faults and attacks are inspired by the real world, including lossy channels, network jitter, data corruption, or disconnections.**

I. Introduction

Development and testing of reliable or fault-tolerant network applications regardless of their architecture must take into account various network conditions. The underlying network may experience faults or even failures caused by a number of reasons, including configuration errors, line or hardware failures, intrusions, or traffic spikes. Setup of erroneous testing environments is, in general, complex as it requires narrow modification of lower-level networking (either software/OS-kernel or hardware-based), mostly applied by experienced engineers. Moreover, the number of test scenarios rises rapidly with a number of different network conditions on which the network application must be evaluated.

Most consequences of network conditions are packets being lost, delayed, reordered, or maliciously corrupted. Packets may be lost in Ethernet or Wi-Fi networks for many reasons, including but not limited to physical factors (such as signal strength or noise) and overflow due to network congestion or excessive queue memory [1]. Loss of the packets or their damage may further cause their re-transmission, prolonging the transmission's delay. If a delay varies for a stream of packets, they may be received in a different order than the order in which they have been sent. If a network is accessible physically or remotely to an attacker, a tailored combination of faults must also be taken into account.

We have developed a tool that provides easily configurable ways to modify the parameters of an underlying network interface such that the network appears faulty. The tool supports several types of faults and attacks to be injected into the network interface. The user may specify under which circumstances the faults are enabled depending, e.g., on time, number of transmitted packets, or specified source and destination. Moreover, the tool provides an interface via its Rest API so that the fault-injection may be configured dynamically and thus automated.

The rest of the paper is organised as follows: The next section introduces the fault and attack models supported by the tool. Section III describes three different types of tool deployment. Section IV provides a tool demonstration and its usefulness in a real use case. The paper is concluded in Section V.

A. Related work in testing network applications

Currently, the most common method of simulating faults on a network communication is using software implemented fault-injection (SWIFI). The method is based on the idea that actual faults on a link layer are hard to be controlled, thus it focuses on the effects the faults cause [2].

Many tools support the injection of faults into network communication. These tools can be divided into two categories: simulation tools, which deal with simulating model situations of the network infrastructure, and tools that are deployed to real systems to evaluate the resulting implementation. An example of a popular simulation tool is OMNet++ [3], which allows the modelling of entire network infrastructures and defines the parameters of the different communication protocols used. An example of a popular tool used on real network interfaces is Netem [4], [5]; a network behaviour emulator that uses the traffic control feature of the Linux kernel, allowing it to add delay, packet loss, and other queuing disciplines to outgoing packets.

The NetLoiter tool has similar features to Netem but closes the gap between the easily used and tailored fault-injection and rich features to influence network communication.

II. Fault-injecting the communication channel

The main features of the NetLoiter include (i) easily specified faults and attack (*what* to be used), (ii) tailored targetted faults (*where* and *when* to be used), and (iii) easy reconfiguration even during run-time enabling a feedback loop depending on an actual net flow. NetLoiter is based

on a man-in-the-middle attack enhanced with fault-injection on a single network interface or a communication channel. Depending on its deployment, the tool uses Linux traffic control facilities, Linux netfilter [6] QoS, or TCP/UDP socket proxy infrastructure.

A. Fault and attack models

NetLoiter can model different types of faults and attacks[1], including packet loss, delayed packet delivery, jitter (unstable latencies between packets) [7], packet reordering, network speed, and packet content manipulation. The faults may be persistent or transient, applied on different IP flows (source and destination addresses and/or ports), protocols and other packet or frame attributes. The specification of the fault model to be used is done by configuring rules consisting of pairs of guards and actions, where guards specify under which condition the rules will be applied, and actions specify the type of fault to be injected. The Listing 1 straightforwardly shows an example of a simple configuration of the fault model, where if all conditions are met, the packet will be dropped (the only packet being dropped is every fifth IP packet incoming from the vutbr.cz subnet). Note that the configuration can be given during the NetLoiter setup or modified during run-time.

Listing 1
EXAMPLE OF A CONFIGURATION OF THE FAULT MODEL.

```
rules:
- $type: All
  guards:
  - $type: IP
    sip: vutbr.cz/16
  - $type: EveryN
    n: 5
  actions:
  - $type: Drop
```

NetLoiter supports the following faults (so-called actions):

- Delay(N)—wait N seconds with packet; N may be a fraction of a second,
- Drop—stop packet processing and do nothing,
- Finish—stop packet processing and pass it further,
- Reorder—change the order (either random or reverse) of N captured packets,
- Replicate(N)—create N copies of a packet,
- Restart—place the packet back into the packet queue and restart its processing,
- Throttle—reduce the maximal throughput (Bytes/s),
- BitNoise—introduce single or multiple bit-flips in a payload,
- SocketTCP(S)—send a packet to a remote socket S which can modify it and potentially return a decision (drop, finish). Note that this action enables a user to monitor (so-called packet sniffing), and modify the network flow during run-time, which can be used, among others, to simulate purpose-specific attacks.

[1]Further, we will only use the term fault as we see attacks as malicious types of fault

The tool supports the following conditions (so-called guards), *where* and *when* the faults will be injected. Each of the conditions expresses the predicate which is evaluated on the processed packet if its occurrence, type, or attribute holds (or does not hold, depending on parameters) a specific value.

- EveryN—holds for every N-th packet,
- ICMP—holds for specific ICMP type messages,
- IP—holds for IP packets and their src. or dst. addresses,
- Port—holds for TCP/UDP specific src. or dst. ports,
- Prob—true with probability P,
- Protocol—holds for specific IP protocol number,
- Time—holds after a specified time for a specified duration (or forever),
- Count—holds after a specified number of processed packets for a specified number of packets (or forever),
- TimePeriod/CountPeriod—alternate based on time period/packet count,
- Size—holds if a packet size is above or below the threshold.

Most of the guards and actions are parametrised and to increase the potential of automated testing or to approximate the real effects of injected faults, numerical parameters can be specified by function symbols instead of a constant:

- Uniform—generates a random value (a float or an integer) in a specified interval,
- Normal—generates a random value (a float or an integer) based on a normal distribution with mean and standard deviation,
- SeqCount—provides values in ascending/descending order within a specified interval and a step.

B. Extensibility of the Tool

NetLoiter is a modular tool which comes with a predefined set of guards and actions. The usage of the tool is not limited to these fault models, the tool can be extended by purpose-specific guards and actions. Due to the limited scope of this paper, only an example of the extension module is provided. NetLoiter is a tool with the main control loop written in Python language; all extensions should be specified as uniquely identified classes and inherit from GuardBase or ActionBase. An example in Listing 2 defines a new guard which holds for application layer packets (L4) with a payload size less than 32 (or other limit specified by a parameter in fault model configuration). Now assume an example of an encrypted UDP channel transferring audio stream (in large datagrams) interleaved with small control commands. A fault model combining *IP*, *Port*, *Tiny*, and *Prop* guards together with the *Drop* action can easily simulate an attack on control commands without the knowledge of decryption keys. Currently, the tool supports a number of attributes of frames (L2), packets (L3), or segments/datagrams (L4) which may be used in fault model extensions. These keys are listed and described in Table I.

Listing 2
EXAMPLE OF IMPLEMENTATION OF A NEW GUARD.

```
class Tiny(GuardBase):
  def __init__(self, threshold = 32):
    self.threshold = threshold

  def fulfilled(self, packet):
    key = ExtractableKey.L4_PAYLOAD_RAW
    if packet.has_value(key):
      if len(packet.get_value(key)) < \
          self.threshold:
        return (True, None)
    return (False, None)
```

TABLE I
SELECTED ATTRIBUTES (KEYS) OF PACKETS PROCESSED BY NETLOITER. THE KEYS ARE GROUPED DEPENDING ON THE NETWORK LAYER WHERE THE TRANSMITTED DATA ARE PROCESSED.

Key	Description
L2 frames key	
TYPE_ID	protocol type in the Ethernet frame
ETHERNET_RAW	Ethernet frame as bytes
SMAC, DMAC	source/destination MAC address
L3 packets keys	
PROTO	protocol type in the IP packet
IPv4, IPv6	IP packets version 4 resp. 6
SIP, DIP	source/destination IP address
L4 packets keys	
L4	unified structure of L4 data packets (TCP, UDP)
TCP, UDP	TCP/UDP segment/datagram
TCP_RAW	TCP segment as a sequence of bytes
UDP_RAW	UDP datagram as a sequence of bytes
SPORT, DPORT	source/destination port number
L4_PAYLOAD_RAW	L4 packet's payload as a sequence of bytes

III. TESTING ENVIRONMENTS

NetLoiter is designed in three variants, one hardware and two software. The hardware solution is an external device connected between the communicating stations, representing a bridge (Layer 2) or an IP network router (Layer 3). On the other hand, the software solutions are run directly on one of the communicating stations, either in a hidden form, in which it intercepts their communication without the knowledge of the communicating parties, or in a visible form, in which TCP or UDP streams must be redirected. NetLoiter intercepts communication between two or more nodes. All the traffic specified by the flow definition is sent to the NetLoiter where the traffic processing depends on the rules defined by the fault model. For a more thorough analysis, NetLoiter can report all the events and actions during run-time.

The three variants are described in the following subsections, where the meaning of the nodes and connections (A)–(E) are related to Figure 1.

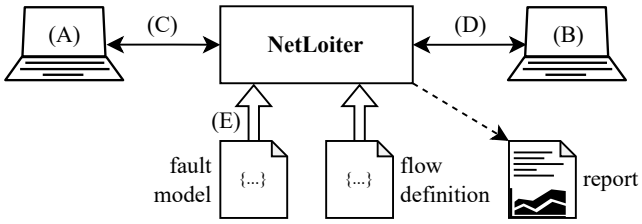

Fig. 1. Overall NetLoiter architecture.

A. Hardware-in-the-loop – Real Network Interfaces

The processing part of NetLoiter is installed on an external hardware device, e.g., a Linux-based minicomputer such as Raspberry Pi, with at least two network interfaces—(C) and (D). The device simulates a network bridge or IP router that receives the communication and forwards it transparently to its destination—(A) and (B). Fault model (E) can be set up statically by a configuration file or dynamically using Rest API via the reserved net flow from one of the communication nodes.

The main advantage of this approach comes from the black-box nature, where no intervention to the communicating nodes is required. Such an approach can be applied to testing in-the-field of a system using real network interfaces (such as cyber-physical devices or solutions accessing the external networks without the means of their control). This also comes with its disadvantage as it can hardly be used in remote testing solutions such as continuous integration/continuous deployment (CI/CD) pipelines, where in-the-lab testing relies mainly on virtual networks.

B. Hidden Software Solution – Virtual Network Interface

NetLoiter is installed on one of the communicating nodes, (A) or (B), where the desired traffic is rerouted to the tool via underlying technology. NetLoiter creates a virtual network or applies routing rules, (C) and (D), to catch and process all the data transferred within the tested system. The advantage of such a solution is that it can be used during testing in-the-lab, in CI/CD pipelines, or in a developer's computer. At the same time, the intercepting process is hidden to the tested network application—no need to apply dependency injection to mock network services. Another advantage is the easy way of dynamically changing the fault model via NetLoiter's Rest API interface (E). NetLoiter relies on traffic control or netfilter's NFTables, both of which are supported on Linux-based systems only—the disadvantage of this solution is the limitation that at least one of the nodes must run the Linux operating system. Another disadvantage is the requirement for advanced user permissions to access and modify virtual networks.

C. Visible Software Solution – TCP/UDP Proxy Application

Another possible solution for software testing of network applications without privileged access to the Linux-based system is to use NetLoiter as TCP/UDP proxy application.

Assuming one of the nodes is a client (A), and one is the server (B), such a solution requires the client to be modified or reconfigured internally to send outgoing traffic (C) to the NetLoiter mediator. NetLoiter will listen on specified TCP or UDP ports and forward the traffic to a specified destination, (B) and (D). NetLoiter, in such a case, has limited features as it can only inject faults in Layer 4—no access to underlying layers is available. For instance, messages (or data stream) already accepted on a TCP client side cannot be reordered on the server side as this has no rationale in real network faults since TCP/IP stacks ensure that received packets are reordered to restore the original data stream correctly. Static delays, reduced network throughput, and man-in-the-middle attacks aimed at application protocols are the main faults targeted by this solution.

IV. Testing Scenario

The tool has successfully been used for the validation of minimum requirements on the quality of a network link. NetLoiter has been used for searching the conditions under which the analysed system works correctly. NetLoiter has been applied as a hidden solution on a network interface shared by the communicating parties.

The scenario takes place in a teleoperation system [8] consisting of a computer station managing an actual vehicle remotely. The remote station and vehicle communication is managed by several TCP and UDP streams for connection management and for the transfer of driving commands, status data, video streams, and heartbeat messages.

This experiment aims to impact mentioned heartbeat messages, whose interruption may indicate a loss in the connection between the vehicle and the remote station. The connection failure between the remote station and the car can lead to loss of control, which can cause severe problems in real-world traffic. The goal of this experiment was to simulate the conditions of a real non-ideal network using delay, reordering, discarding, and replication of IP packets to find the limits of a reliable network connection.

The experiment was supported by 6 automated test cases incorporating the real remote station, the vehicle communication module, and the simulated scenario as a sequence of driving commands. NetLoiter was set to the hidden software solution with the HTTP server processor, allowing it to dynamically change NetLoiter's fault configuration using its Rest API configuration management. Capturing and processing the IP packets was set to influence the heartbeats between the remote station and the vehicle. NetLoiter's configuration has been systematically and automatically changing during repetitive execution of the test cases to quickly find the boundaries of fault-injection setup, which leads to failing tests (and thus to the detection of the unreliable link).

The findings of the experiment are shown in Table II. We aimed to influence the flow of IP packets (i) from the remote station only, (ii) to the remote station only, and (iii) both ways (duplex) between the remote station and the vehicle.

TABLE II
Evaluated network conditions impacting the stream with heartbeats in a remotely controlled car experiment. If any network conditions are met, the link between the remote station (RS) and the vehicle is assumed to be unreliable[2].

Injected fault	Duplex	From RS	To RS
Delay for x [s]	$x \geq 0.4$	$x \geq 0.7$	$x \geq 0.7$
Drop every N	$N \leq 4$	$N \leq 2$	$N \leq 1$
Drop with prob. x	$x \geq 0.45$	$x \geq 0.25$	$x \geq 0.95$
Reorder N	$N \geq 12$	$N \geq 2$	$N \geq 17$

V. Conclusion

The paper introduced the NetLoiter tool for testing the robustness of network applications and how they perform in different network conditions. The tool implements fault-injection on a network layer, simulating a number of faults and attacks which unreliable or even insecure networks may cause. NetLoiter is modular, dynamically reconfigurable, easy to use, and can be deployed for in-the-field and in-the-lab testing. This makes it a valuable tool for the functional testing of network applications.

Acknowledgment

This paper was supported by the VALU3S project which has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876852. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, and Turkey. The views expressed in this document are the sole responsibility of the authors and do not necessarily reflect the views or position of the European Commission. The authors, the VALU3S Consortium, and the ECSEL JU are not responsible for the use which might be made of the information contained here.

References

[1] C. A. G. D. Silva and C. M. Pedroso, "Mac-layer packet loss models for wi-fi networks: A survey," *IEEE Access*, vol. 7, pp. 180 512–180 531, 2019.

[2] R. Natella, D. Cotroneo, and H. S. Madeira, "Assessing dependability with software fault injection: A survey," *ACM Comput. Surv.*, vol. 48, no. 3, feb 2016. [Online]. Available: https://doi.org/10.1145/2841425

[3] OMNet++, "Discrete event simulator." [Online]. Available: https://omnetpp.org/

[4] H. Stephen, "Network emulation with netem," in *Proceedings of the 6th Australia's National Linux Conference*, 2005, pp. 1–8.

[5] Linux Manual Page, "NetEm - Network Emulator," 2011. [Online]. Available: https://www.man7.org/linux/man-pages/man8/tc-netem.8.html

[6] Netfilter contributors, "Netfilter.org Project," 2023. [Online]. Available: https://netfilter.org/

[7] V. Kartashevskiy and M. Buranova, "Analysis of packet jitter in multi-service network," in *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*, 2018, pp. 797–802.

[8] VALU3S Consortium, "VALU3S Automotive use cases," 2020. [Online]. Available: https://valu3s.eu/automotive-use-cases/

[2]Disclaimer: The results do not represent the requirements on the network conditions of the teleoperation in real-world situations; they only reflect the configuration settings for the simulated in-the-lab execution.