codilime

Network Hardware Performance

Exploring Automated Testing Strategies Using Open-Source Solutions

- Designing a network topology for RFC2544 tests
- Deployment in physical vs. virtualized environments
- Choosing the right technologies for test automation



Executive summary

In this ebook, we aim to present our approach to performance measurement of network cards, including innovative solutions such as SmartNICs. This approach is developed by integrating the best practices from the fields of software development and network engineering.

Our goal is not only to showcase tools and techniques but also to demonstrate their practical application in real business situations. **Remember that this ebook is not intended to delve into technical details but to build awareness of the approach we can take for performance testing of a product and to frame it within a suitable use case.**

Exploring client product functionality

Starting our journey through the testing process, the first step is to understand how our client's product works and what functions it is intended to fulfill in a real operational environment. Focusing on network cards, including advanced solutions such as SmartNICs (Smart network interface cards), our interest lies in products designed to meet the increasing demands for throughput and processing in today's complex networks. The key goal of our research is not only to assess the basic performance of the device but above all to understand how offloading network functions and other advanced technologies impact overall system efficiency.

Understanding the testing process

We'll guide you through each stage of our testing process, starting with building the approach and testing strategy. Then, we'll take a closer look at the initial manual tests which form the foundation of our efforts. However, this is just the beginning, as we aim to implement automation to make our actions more efficient and scalable.

Selection of network methodologies and standards

Choosing the right methodologies and network standards is crucial for accurately assessing device performance. In our research, we use standards such as RFC 2544, Y.1564, and RFC6349. RFC 2544 focuses on maximum throughput and device stability, while Y.1564 and RFC6349 allow for a comprehensive evaluation of network services, including QoS and TCP throughput. It is also important to select the appropriate tools and devices that will enable standards verification, such as considering the hardware or software that allows for traffic generation compliant with the tested standard.

Additionally, we'll delve into the measurement of latency, which is essential for assessing network performance. This can be done using specialized standards such as RFC 5481, which provides guidelines for measuring one-way delay in IP networks. **Throughout this ebook we will mainly rely on tests from the RFC2544 standard.**

Required hardware and software

Conducting tests requires suitable software and hardware. Various solutions are available, both commercial and open source. Among the commercial solutions are tools such as IXIA, Spirent TestCenter, EXFO, and Keysight, offering advanced network performance testing capabilities. In the case of open-

source solutions, it is worth mentioning tools such as Iperf/Jperf, Ostinato, TRex, and Netperf, which are available at no additional cost and can be beneficial for projects with limited budgets. This ebook aims to focus exclusively on open-source traffic generation tools. We will largely concentrate on the TRex tool from Cisco.

Generating reports and interpreting results

We'll also consider generating reports and interpreting results – a crucial step in making data-driven business decisions. Finally, we'll leave some questions unanswered to encourage reflection and further exploration in the field of performance measurements.

Conclusion

Our approach to testing the performance of hardware acceleration products is based on a deep understanding of their functions and capabilities, strategic selection of testing methodologies, and the use of advanced hardware and software. This enables us to provide our clients with detailed information about the performance of their devices, allowing them to optimize their network systems and identify any potential issues.

By exploring various hardware acceleration solutions, including SmartNICs and FPGA-based solutions, we ensure a comprehensive evaluation of network performance regardless of the technology used in product development. Our testing approach remains versatile, adaptable to different business needs, and focused on delivering actionable insights for optimizing network systems in today's digital ecosystem.

Table of contents

Test preparation 5
Test purpose 5
Manual test scenarios 7
Automated test scenarios 7

Designing a network topology for RFC2544 tests 8
Multi-server configuration 8
Single-server configuration 9
Summary and common setup 10
Choosing additional network tools 11

Run manual tests 12
Traffic generator verification 12
Network interfaces 12
Deployment in physical vs. virtualized environments 13
Connect traffic generator to DUT 13
Preparing the testing topology 14
Traffic profile 14
Traffic forwarding at the device under test 15
Running the test 15

From manual test to automation 16
Designing a CI/CD (continuous integration/continuous delivery) pipeline 17
Choosing the right technologies for test automation 18
The importance of CI/CD in test automation 19
The importance frequency of conducting performance tests 19
Choosing a testing framework 20
Pass/fail criteria for performance tests 21

Reporting and results 21
Reports for technical teams 24
Reports for business and marketing teams 24
Common elements for all reports and summary 25

About the author 26

About CodiLime 26

Test preparation

Before embarking on our journey through the maze of performance testing, it's important to establish the direction of our testing efforts. This process will start with selecting or designing a test scenario tailored to the specifications and requirements of the product being evaluated. This step will determine how we measure the performance of our product.

The next stage involves designing or selecting the appropriate test topology. It's worth noting that depending on the needs, such tests can take place in various configurations – for example, tests compliant with RFC 2544 may require symmetric data flow in both directions, while delay measurements may be based on linear flow in one direction.

In the context of performance, an important metric may be RTT (round-trip time), but **does it truly provide a complete picture of the time needed to process a packet by our product? This question opens the door to discussion**, and for those interested in delving deeper into the topic of delays, we encourage you to visit our blog where we focus on a traffic generator for measuring network performance.

Next, our attention will focus on configuring the chosen test topology, with a particular emphasis on environments based on CentOS 8 or RedHat systems. Optimizing the test environment, considering aspects such as memory configuration for Intel Xeon DIMMs, NUMA node settings, and BIOS settings will be crucial, especially when the product provided by the client exists only as a simulation without dedicated hardware. **It's worth considering whether performance tests make sense in such a scenario, which is another open question on our discussion list**.

One of the most significant aspects of our testing approach will be selecting a tool for generating network traffic. There are many options available, both paid and open source. In our guide, we will focus on free solutions such as Cisco's TRex, without neglecting auxiliary tools like Scapy. Depending on the specifics of the product being tested, there may be a need to use additional tools, such as virtual or physical routers like Bird or FRR, etc.

Test purpose

The purpose of a test, whether conducted manually or automatically, is always to determine the performance and reliability of the tested product or system. In that context, the first choice would be to base tests on RFC2544 standards, which include parameters such as latency. But depending on circumstances or requirements we may have to focus more on how the system performs in interface switching scenarios such as LACP (Link Aggregation Control Protocol) or LAG (link aggregation group) failover.

It's advisable to precisely define which aspects of the product will be verified for performance and what the specific product requirements are. Variables such as the product development stage or the possibility of conducting tests on simulated products should also be taken into account. For products designed for hardware acceleration of MPLS or VxLAN frame encapsulation/decapsulation, tests should focus on encapsulation/decapsulation time and the number of processed packets.

In RFC2544 testing, it is crucial to establish acceptable values for the packet threshold. According to RFC2544, the packet loss threshold is set at zero, meaning that the tested device should ideally experience no packet loss during throughput, delay, jitter, and packet loss tests. However, this standard assumes that the device is in its final form. In scenarios where the system under test (SUT) is still under development, predefined modifications to this criterion may be necessary. These adjustments should be clearly defined and documented prior to the commencement of testing, ensuring all stakeholders have aligned expectations regarding performance benchmarks. For example, an agreed upon packet loss of up to 1% might be acceptable for

products that are not yet fully optimized. Additionally, should test results indicate lower than expected device throughput, it is imperative to investigate and identify potential causes for this discrepancy.

However, if tests reveal lower than expected device throughput, it becomes necessary to identify potential causes, which may be:

1. **Insufficient throughput of the test infrastructure.**
   If the infrastructure used for testing, including switches, routers, and transmission media, does not support a throughput of 25 Gbps, this may lead to unexpected packet loss or lower throughput results.

2. **Device configuration errors.**
   Incorrect device configuration, such as improper queue settings, throughput limits, or quality of service (QoS) policies, may lead to packet loss even if the theoretical device throughput is not exceeded.

3. **Hardware limitations.**
   Network devices, including the tested device, may have hardware limitations that are not directly related to throughput but may affect throughput under high load, such as packet header processing, interrupt handling, or buffering strategies.

4. **Protocol stack implementation errors.**
   Issues with the implementation of network protocol stacks may lead to improper packet handling under high load, resulting in errors such as packet loss, even if the physical layer is capable of handling high throughput.

5. **Tester performance limitations.**
   The hardware and software used for generating and analyzing network traffic must also be capable of handling speeds of 25 Gbps without introducing their own limitations that could skew test results.When employing a software-based network traffic generator, such as TRex, it is crucial to select hardware that is compatible with the chosen software. Detailed guidelines and hardware recommendations for TRex can be accessed here. This principle holds true for any software traffic generator: the hardware used must be supported by, and thoroughly tested with, the software to ensure reliable and accurate performance measurements.

6. **Time synchronization issues.**
   In delay and jitter tests, precise time synchronization between the testing device and the tested device is crucial, and its absence can lead to incorrect measurements.

7. **Inadequate test conditions.**
   RFC2544 tests should be conducted in a controlled environment that replicates real-world usage conditions. Inadequate test conditions, such as unforeseen network congestion during tests, can affect results.

A thorough understanding and definition of the test objective, along with proper preparation for identifying and resolving potential issues, are crucial to ensuring the credibility of test results and effective demonstration of the performance and reliability of the tested product. During the process of conducting tests, whether manually or automatically, it is essential to utilize appropriate tools (such as Cisco's TRex) and hardware recommended by the tool manufacturer. This ensures the highest level of accuracy and reliability in the results obtained. Additionally, it is imperative that these tools are thoroughly tested and verified prior to their use, rather than during the testing of the device or system under test.

Manual test scenarios

1. **The test design process:** This begins with identifying the objectives of the tests, which should be based on the specific requirements of the client and the intended use of the product. For example, the client may be interested in evaluating the performance of jumbo frames and their impact on overall system performance. During this phase, we combine predefined scenarios/procedures, such as those defined by RFC standards, with the individual needs of the client and the product characteristics. This ensures that our testing procedures are tailored to the specific requirements of the client and are relevant to the product being tested. Test scenarios may also directly result from the production topology provided by the client. This approach allows us to replicate real-world conditions and ensures that our tests accurately reflect the environment in which the product will be used.

2. **Test environment configuration:** Manually preparing the environment involves configuring the hardware and software according to the test scenario requirements. This may include setting up the TRex traffic generator to simulate specific network traffic conditions in line with the designed test.

3. **Test execution:** Testers physically conduct tests according to prepared scenarios. During this phase, testers run and supervise network traffic generated by the TRex server, monitor system behavior, and record any abnormalities or errors by the traffic generator stats.

4. **Results analysis:** This involves evaluating the collected data, identifying potential errors, performance issues, or other unexpected system behaviors. This phase may also involve comparing results with expectations and industry standards.

5. **Reporting:** Documenting test results is essential for further analysis and system optimization. Reports include detailed descriptions of the performed steps, encountered issues, and recommended corrective actions. Additionally, our process includes an iterative approach, allowing for the implementation of improvements in subsequent test series.

Automated test scenarios

1. **Testbed verification:** This involves conducting simple smoke tests to ensure that all components of the test environment are properly configured and ready for full testing. It verifies if the SW traffic generator (TRex) is capable of generating traffic at the expected level and if the environment is stable.

2. **Test scenario configuration:** The test scenario configuration process begins with the automation of test scenarios, ensuring efficiency and reliability. This includes defining and configuring tests using available TRex test scripts, which can be customized to specific testing needs. Additionally, our testing process benefits from the availability of ready-to-use test scripts provided by TRex, which is one of its key advantages. Configuration also involves setting test parameters, such as the type and intensity of generated network traffic.

3. **Test environment deployment:** This involves automatically preparing the test environment, including deploying necessary software and configuring devices. This may involve automated scripts for configuring the TRex generator and other infrastructure elements.

4. **Tests execution:** During this phase, a series of tests is automatically initiated according to the test plan. The full capabilities of the traffic generator (TRex) are utilized to simulate network traffic and monitor the performance of the device under test. It's important to note that test parameters should be specified, with smoke tests typically conducted using reduced parameter values. Subsequent tests are then performed with appropriate parameter settings to fully saturate the device under test.

5. **Data collection and analysis:** The automatic collection of test data, followed by analysis

and comparison with expected results. Special attention is paid to identifying deviations from expected performance and any abnormalities.

6. **Reporting and interpretation of the results:** During this phase, detailed reports regarding the conducted tests, analysis findings, and recommendations for optimization or repair of identified issues are prepared. These reports are crucial for further development and improvement of the tested system. It is worth noting that certain aspects of the reporting process can be easily automated. For instance, report generation based on traffic generator statistics can be automated to various file formats such as CSV. Additionally, issue detection in areas where the expected RFC2544 threshold was not achieved can be automated. This ensures that reports are easily comprehensible, highlight anomalies, and are stored in a systematic format.

By combining both manual and automated approaches, a comprehensive assessment of the system is achieved, not only providing information about quality and performance of the SUT but also enabling swift identification and confirmation of discovered issues.

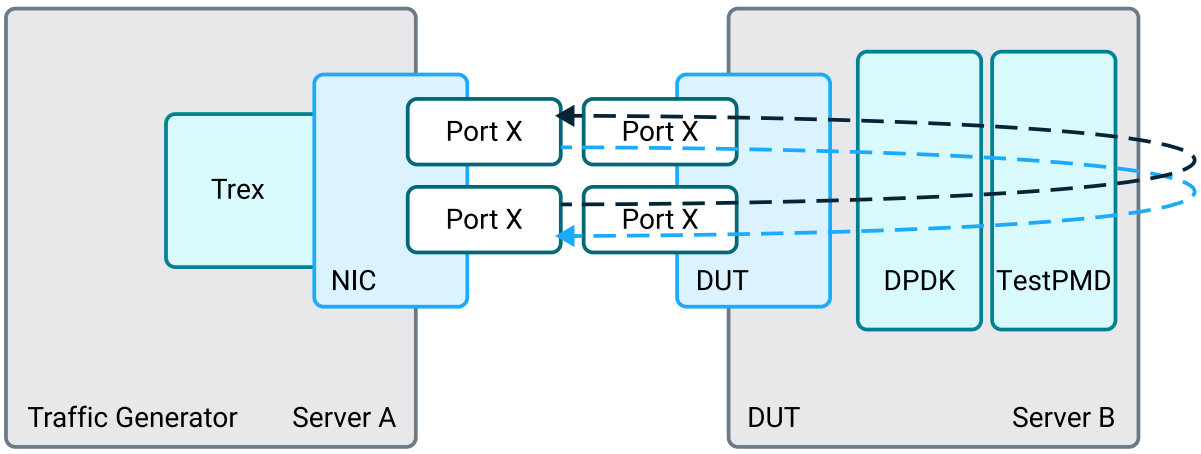
Designing a network topology for RFC2544 tests

Designing a network topology for RFC2544 tests using TRex as the network traffic generator requires careful planning and configuration to ensure the reliability and accuracy of results. Below is a detailed description of such a topology, taking into account the key aspects.

Multi-server configuration

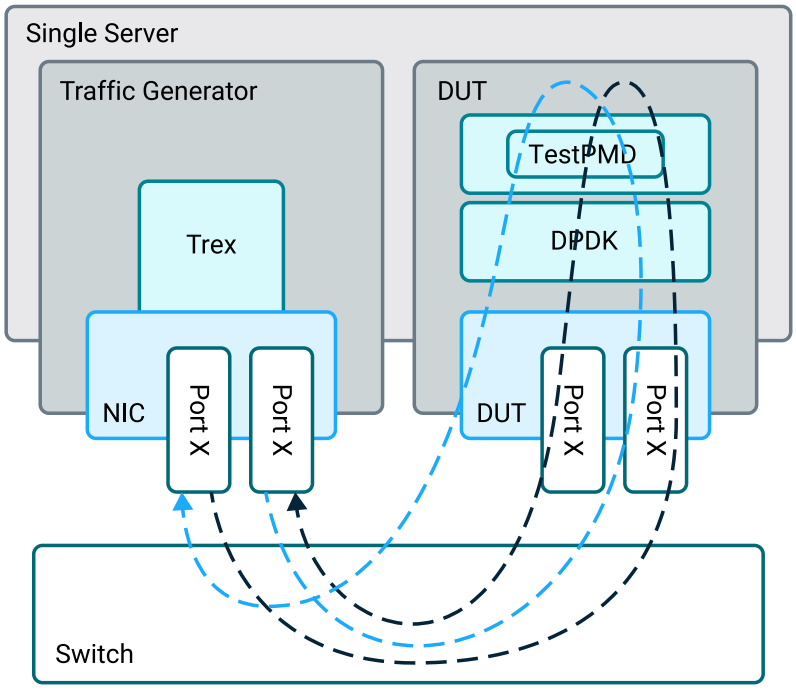
Traffic generator server (TRex)
TRex is an advanced network traffic generator that can be used to simulate various types of traffic such as HTTP, VoIP/Calls, ACLs, and more. TRex documentation is available on the official project website and provides detailed information on hardware requirements and configuration. Running TRex requires a relatively powerful server, preferably with high-end processors, ample RAM, and at least two network interfaces ranging from 1/10/25/40/50/100 GbE, depending on the desired throughput to be tested. The configuration should also consider NUMA architecture settings to optimize performance and minimize memory access latency.

Server for the tested product
Requirements for the server hosting the tested product depend on the product specification and client requirements. Ideally, it should be a server with technical parameters that fully utilize the networking capabilities offered by the tested network hardware.



Single-server configuration

Consolidating the test environment on a single server. To reduce costs, it's possible to consolidate the entire test environment, including TRex and the tested product, on a single server. However, such a configuration requires careful analysis and planning:



Summary and common setup

High hardware requirements: The server must be exceptionally powerful to simultaneously generate high network traffic and handle the tested product. Fast processors, ample RAM, and an adequate number of fast network interfaces are crucial here.

NUMA configuration for TRex and TestPMD (DUT): In such a setup, optimal CPU core assignment and memory allocation in the context of NUMA nodes will be crucial to minimize latency and maximize performance. This requires manual adjustment and can be complex to configure but is essential for achieving reliable test results.

Isolation of network traffic: Ensuring isolation of the network traffic generated by TRex from the traffic handled by the tested product may require the use of networking features such as VLANs, network namespaces, or even physical separation of network interfaces.

Software and virtualization for TRex and TestPMD (DUT): Consideration can be given to using containers or virtual machines to separate the test environment from the product, which further increases flexibility and isolation but also introduces an additional layer of complexity and potential performance overhead.

Isolation of CPU resources for TRex and TestPMD (DUT): Ensuring that CPU resources are properly isolated for the test environment can prevent interference between different components running on the server. This isolation can be achieved through techniques such as CPU pinning, where specific CPU cores are dedicated to different tasks, optimizing performance and minimizing contention.

Implementation of huge pages for TRex and TestPMD (DUT): Enabling huge pages can significantly improve memory management efficiency, especially in environments with high memory usage. By using larger page sizes, the system can reduce the overhead associated with memory translation and improve overall performance. Proper configuration of huge pages is essential for maximizing the benefits while avoiding potential drawbacks.

Queue configuration for TestPMD (DUT): TestPMD and DPDK-l3fwd rely on queue management for packet processing. Properly configuring queues involves setting up queue parameters such as queue size, buffer size, and queue mapping. This optimization ensures efficient packet handling, reduces latency, and maximizes throughput.
Queue mapping, so assigning queues to specific CPU cores or threads is critical for load balancing and parallel processing. By mapping queues to dedicated CPU resources, TestPMD or DPDK-l3fwd can efficiently distribute packet processing tasks across available

cores, optimizing performance. By carefully configuring queues and mapping them to CPU cores, TestPMD or DPDK-l3fwd can achieve optimal packet processing performance, ensuring reliable and efficient testing of network performance.

In summary, designing a network topology for RFC2544 tests using TRex as the traffic generator is a complex task that requires meticulous planning and configuration. Depending on specific requirements and budget constraints, various approaches can be considered, from dedicated hardware to integrated test environments on a single server, each with its own set of challenges and compromises.

Choosing additional network tools

Additional network tools are crucial for accurately conducting performance tests, especially when the tested product requires special features or supports specific protocols, or when simulating real-use case scenarios. To give you a taste, here are a couple of examples of issues that we have encountered requiring an additional tools approach, and how we handled them:

Scenario and example.
Let's assume our product has built-in hardware acceleration for MPLS over UDP encapsulation and decapsulation. If we choose TRex as our network traffic generator, we encounter the first problem. TRex supports L2/L3 traffic (here you'll find more details), but for special cases like MPLS over UDP, additional support is needed.

Statistics collection.
To obtain accurate performance results, we need detailed statistics, such as the number of packets sent and received within a specified time.
However, when additional elements are present in the topology, such as network switches, they generate additional traffic, such as ARP packets. TRex offers special parsers that can collect only the statistics defined in our streams for the traffic generator.

Parsing issues and latency.
However, TRex parsers do not support the MPLS over UDP protocol and may have limitations in collecting timing information, such as latency calculation. In such a situation, we may consider using an external device, such as a router or switch, as an additional translation layer between the SUT and network traffic generator. This additional module would be responsible for encapsulating or decapsulating traffic before reaching our device under test (DUT) in the network topology.

Bird and FRR integration for LACP/LAG.
When testing for performance verification scenarios involving Link Aggregation Control Protocol or link aggregation group where measuring switch-over time during failures is essential, additional services like virtual routers Bird or FRR (Free Range Routing) can be valuable.
These routing daemons support dynamic routing protocols and can be configured to simulate complex network topologies, enabling accurate testing of LACP/LAG behavior and failover mechanisms.

Traffic reflectors in testing: an overview.
In scenarios where the topology demands it, traffic reflectors such as TestPMD or DPDK-l3fwd can be employed to handle specific types of traffic, ensuring comprehensive testing coverage.
Traffic reflectors, also known as traffic generators or packet reflectors, are essential tools in performance testing. They simulate network traffic patterns to evaluate the performance and reliability of networking devices, such as routers, switches, and firewalls. Reflectors play a crucial role in stress testing and benchmarking network infrastructure to identify bottlenecks, measure throughput, and assess latency and packet loss.

Lessons learned.
When designing a test approach, it's essential to consider the need for additional components/modules, especially when the tested product requires special features or supports specific protocols. It's worth starting discussions about the necessary additional tools at an early stage of the project because it may affect the cost of the entire endeavor.
Clarifying requirements regarding tools and components early on will help avoid problems and delays in later stages of performance testing.

Run manual tests

In the context of this ebook, issues related to the environment of the traffic generator and the device under test are addressed in a simplified manner, omitting detailed technical aspects. As mentioned in the introduction, the decision regarding the installation and configuration method of TRex, whether through compiling from provided source code or by downloading a ready-made installation package, remains at the discretion of the user. A similar situation applies to the DUT - the method of its preparation and configuration also depends on individual preferences and project requirements. We emphasize that both the choice of TRex installation method and the preparation of DUT should primarily be guided by expectations regarding performance, scalability, and the specifics of the tested scenarios. Our goal is to provide general guidelines and best practices that can be adapted to specific project conditions and requirements.

Traffic generator verification

Before embarking on the first performance test of network solutions using data acceleration by SmartNIC card, it is crucial to carry out several preliminary steps to ensure the reliability and efficiency of testing. Initially, it is necessary to verify the configuration of the traffic generator. This process involves setting it up in a loopback topology and adopting assumptions for the test scenario based on the RFC 2544 standard, which is a key reference point for network performance tests.

Initially, we assume that the device under test has a maximum duplex throughput of 25 Gbps. Therefore, it is necessary to use a traffic generator characterized by higher throughput than the tested DUT in order to fully load the device and measure its performance at maximum possible throughput values (supported NICs for the traffic generator are described in the TRex documentation.

Next, the loopback topology must be configured, which can be achieved in several ways. One option is to use special cabling (for technical details, see the documentation). Alternatively, the traffic generator can be connected to an external switch, which then "loops" the traffic back to the generator, thereby enabling an effective loopback.

Utilizing such a configured topology allows for various tests, including the basic UDP traffic test. The aim of this test is to verify the generator's ability to generate maximum traffic, which will be handled by the NIC cards in collaboration with TRex. For users looking to automate the testing process, TRex offers a special NDR script which provides information on the maximum throughput available for the traffic generator.

Network interfaces

After selecting the appropriate network interface cards based on the TRex documentation, the next step is to configure the interfaces available to the TRex server, which plays a crucial role in generating network traffic. TRex offers two main modes of interface operation: the first, with lower performance, involves integrating interfaces directly with the Linux system, while the second, designed for high-performance applications, relies on using interfaces integrated with DPDK (Data Plane Development Kit).

With the higher performance option, we choose to configure and bind interfaces with DPDK. This process is carried out using dedicated scripts provided by TRex, enabling quick and efficient configuration. As a result, the TRex server can fully leverage the potential of the NICs, generating network traffic in a significantly

more efficient and effective manner. TRex also supports Kernel interfaces; however, their performance significantly falls short of meeting the requirements of RFC2544. This inadequacy is largely due to the interrupts that occur within the kernel (here, you can find details).

Implementing such a solution not only significantly enhances the testing capabilities of the system but also paves the way for conducting more complex and demanding performance tests, which is particularly crucial in scenarios where maximum throughput and minimal delays are key factors in evaluating the quality of network solutions.

Deployment in physical vs. virtualized environments

When launching traffic generator like TRex we also have to make a choice of whether to run it directly on a bare-metal machine or in a virtualized environment, it's important to bear in mind that the best performance is achieved when it operates at the level of a physical host, without additional abstraction layers such as virtualization (e.g. using QEMU or virtual machines) or containerization (as in the case of Docker). Introducing such layers can generate unwanted disturbances that directly affect the accuracy of measurements – for instance, in calculations concerning latency and other parameters that may disrupt the obtained results.

However, it's essential to remember that the choice of the appropriate runtime platform for a traffic generator like TRex also depends on specific project requirements and goals set for the tested system. Therefore, it's not possible to unequivocally state that one solution will always be better than the other in all contexts.

Based on our experience, though, TRex operates most efficiently and provides the most reliable results when run directly on the host hardware. Such a configuration minimizes the risk of introducing uncontrolled variables that could adversely affect test precision. Nevertheless, we encourage you to consider all factors.

Connect traffic generator to DUT

After verifying the performance of our network traffic generator using a loopback topology and ensuring that it meets the expectations regarding project requirements, without detecting disturbances stemming from faulty hardware, we can proceed to integrate the traffic generator with the device under test. At this stage, depending on the project specifications, we have the option of directly connecting the generator to the DUT or via physical devices such as switches.

The use of a switch may prove advantageous in the long term, especially if we plan to scale our testing environment. One can imagine a scenario where one traffic generator serves multiple testing environments by connecting to various DUTs through a switch, which could significantly reduce project costs. Additionally, the selection of such a switch, along with other network devices and cables, is crucial to ensure they do not become bottlenecks in the system. This is just one of many potential utilization scenarios.

For simplicity, in our specific case, we may opt for a direct connection between the traffic generator and DUT using appropriate cables. The choice between using a single server for both the traffic generator and DUT versus separating these functions into two separate servers will impact the configuration of TRex in each of these cases. This will be particularly significant when considering NUMA node configurations and CPU isolation between the traffic generator and DUT.

A key aspect here is the proper planning of the testing architecture, considering both current and future project requirements, to ensure flexibility, scalability, and cost effectiveness.

Preparing the testing topology

Preparing the testing technology requires considering the specificity of the product being tested and the optimal utilization of DUT resources. In the case of our usage scenario, where the product is a SmartNIC card with dedicated firmware, driver modules, and user applications, it's crucial to first test the DUT in the most basic configuration.

Let's imagine that the SmartNIC, according to our assumptions, consists of three layers: firmware, driver module, and user applications. To optimize testing, we should aim to remove the last two layers, leaving only the firmware. In practice, this means we want the SmartNIC to function as a simple traffic reflector, without additional layers of abstraction. This allows us to evaluate basic hardware functions such as traffic load and distribution to queues, or memory operation correctness.

Then, after verifying this basic topology using the traffic generator, we can proceed to add the driver module and user applications, and introduce the final version of the firmware. Such a division of tests allows for quickly detecting any issues at lower levels of configuration, which can significantly reduce defect fix costs. In cases where the client provides the firmware, providing feedback promptly is possible, impacting the efficiency of the project process.

Regarding firmware modifications, particularly those affecting the operation of components such as SmartNIC cards, it's crucial to define the scope of changes clearly to ensure they meet project expectations and requirements. At this stage, collaboration with designers and developers is essential to enable the testing of individual layers and the configurability of input parameters. This approach ensures that modifications are both effective and verifiable.

Traffic profile

The traffic profile is a significant element of testing the performance of the device under test, and its proper definition can have a significant impact on test results. In our case, we will focus on several key aspects of the traffic profile related to the communication protocol, the number of flows, and the packet distribution algorithm.

Firstly, the choice of communication protocol depends on the specifics of the product being tested and the testing objectives. Typically, performance tests based on the RFC 2544 standard utilize a stateless protocol such as UDP. However, it's worth noting that the protocol choice can be tailored to project needs and requirements. In our usage scenario, we adhere to the RFC 2544 standard and utilize the UDP protocol for our streams.

Another vital aspect is the number of flows, which is often determined by product requirements. For network devices, the packet distribution algorithm, such as RSS (receive side scaling), is utilized to evenly distribute network traffic across different processor queues. Utilizing the TRex API interface, we can introduce random or incremental changes for each successive packet, thus affecting the values of the 5-tuple (source IP, destination IP, source port, destination port). By adjusting the algorithm during initial tests, we can verify whether our DUT queues effectively distribute traffic to different queues. This is crucial for efficient network load management and ensures optimal DUT performance.

Additionally, besides standard performance testing aspects, it's essential to consider traffic profiles that may be relevant to the tested user applications. These can include characteristics of application traffic, such as request types, packet size, or transmission frequency, which may have a significant impact on system behavior under real usage conditions.

Traffic forwarding at the device under test

The tTraffic forwarding at the device under test level is a crucial element in performance measurements that can significantly impact the final test results. In our scenario, assuming that the DUT has been deployed on a separate bare-metal server, the traffic forwarding process occurs through firmware, which can handle various networking functions such as routing, traffic filtering, or packet modifications.

We assume a scenario where incoming traffic at the DUT port is modified at the input by adding MPLS labels and then passed to a user space application. This application, equipped with two output network interfaces, is connected to DPDK drivers, and for testing, we utilize the TestPMD DPDK application responsible for packet forwarding between user space application ports.

In this context, several significant issues arise for consideration. Firstly, the TestPMD application utilizes the resources of the bare-metal server, which may introduce additional processing time, affecting the final measurement results, especially concerning latency measurements. Additionally, TestPMD may act as a bottleneck in terms of packet processing performance, potentially leading to packet drops.

In such a scenario, the TestPMD application provides various statistics allowing performance monitoring, such as querying DPDK drivers at a lower level. This enables verification of whether traffic is correctly distributed to queues and monitoring the number of incoming and outgoing packets, facilitating the identification of potential performance-related issues.

It's also worth noting that previous tests were conducted solely at the firmware level, providing us with some insights into hardware performance. By introducing the TestPMD element, we can identify potential bottlenecks and adjust resources available to the application, enabling a better assessment of system performance and latency measurement. Of course, the information presented here is simplified and represents a step-by-step approach to our traffic testing, which passes through various elements of our topology."

Running the test

After establishing and verifying the network topology, we proceed to execute a test. Although there are various testing scenarios available, as previously discussed, we will focus on conducting the first RFC2544 test as an illustrative example. To perform this test, several options are available: writing your own script in Python using the TRex API with a simple binary search algorithm, or employing ready-made tools like the TRex NDR script. This example serves to demonstrate the process but can be adapted to other testing requirements as needed.

If the goal is to quickly obtain initial results and assess the performance of the product, I suggest utilizing the ready-made solutions provided by the TRex traffic generator. These solutions have already been verified by the community working on the open-source traffic generator, allowing for a swift performance assessment. As the project develops, you may decide to design your own test suites for RFC2544. This will give you control over flow learning process duration, iteration time during binary search for RFC2544, and other parameters that may influence the final test results.

It's vital to initiate the verification of test scenarios with simpler, less complex cases. Starting simple is strategically sound for minimizing project costs and accelerating the timeline for delivering the final product to the client. By beginning with the most straightforward scenarios, we facilitate the early detection of any potential issues, allowing for timely adjustments in response to these findings. Moreover, parameters such as packet sizes may be modified during the testing process to further investigate and pinpoint the sources of anomalies. Importantly, using TRex enhances our flexibility in adjusting these parameters through its API or by using scripts written in Python, providing a dynamic and adaptable testing environment.

From manual test to automation

In today's dynamic world, where the pace of technological development is exceptionally fast, test automation has become an essential element of the software development process. Transitioning from manual testing to automation brings many benefits, such as speeding up the testing process, increasing accuracy, improving product quality, and saving time and resources. However, is test automation always the optimal solution? The ROI (return on investment) factor did not arise by chance. The decision to automate individual steps should be carefully analyzed from both business and technical perspectives. It's also worth considering the possibility of semi-automated testing, where environment setup is automated, but tests are executed manually.

During the analysis of each case, the repeatability of the same steps is a significant factor, often evident during the construction of the testing environment. If we focus solely on the tests where the end user is only responsible for running a script from the command line interface (CLI), the costs associated with writing a framework to manage tests and the necessity of testing it should be considered. In some cases, these costs may far exceed the benefits of automation.

There may be a situation where the project involves building a platform for benchmarking various products and verifying their performance. Although this may seem like a very ambitious project, aspects of automating the construction of test profiles, running tests, and configuring the environment can serve as a precursor to automation. In such cases, everything can be shifted to the highest level of abstraction with CI/CD and standardized processes.

Automation in network testing transcends the basic use of tools like TRex, which offer preliminary automated functions. In our discussion, comprehensive test automation refers to a fully automated system where operational tasks, such as inserting a network interface card (NIC) and initiating tests, are streamlined to the point where minimal human intervention is required. The results are automatically generated and delivered in report format. Moreover, integrating such automated processes into continuous integration/continuous deployment (CI/CD) pipelines for software development projects facilitates real-time decision-making regarding software changes, enhancing efficiency and accuracy.

Stages of automation in the testing process:

1. **Test environment setup**
   Automation at this stage involves configuring the test environment's topology, which includes setting up bare-metal servers (BMs), virtual machines (VMs), network configurations, and software installations. This foundational step ensures that the environment matches the test requirements precisely, reducing the potential for configuration errors and increasing repeatability.

2. **Test execution**
   Automation here entails executing tests with varied parameters across the established environments. This stage is crucial for assessing the performance and stability of the system under test, ensuring that all scenarios are thoroughly evaluated without manual oversight.

3. **Report generation**
   Post-test, automation aids in the generation of detailed reports in specified formats. This not only speeds up the reporting process but also standardizes the presentation of results, making it easier to interpret and analyze data.

4. **Report analysis**
   The final stage involves the automated analysis of test outcomes using predefined heuristics or historical data comparisons. This advanced automation capability allows for immediate insights into performance trends and anomaly detection, which are crucial for timely decision-making.

Each phase of automation requires careful evaluation to determine its viability and effectiveness. Factors such as the complexity of the test scenarios, the frequency of product updates, and the overall scope of testing need thorough consideration. By methodically analyzing these factors, organizations can identify the most beneficial areas for automation, ensuring optimal use of resources and maximized testing efficacy. Through this structured approach, we aim to clarify the significant advantages and necessary considerations of implementing automation in network testing environments, guiding readers to make informed decisions tailored to their specific project needs.

It's essential to remember that everything depends on the first and crucial issue: the project's goal. Therefore, this is where our discussion begins in this ebook.

Designing a CI/CD (continuous integration/ continuous delivery) pipeline

Designing a CI/CD pipeline is an element in the test automation process, ensuring a continuous workflow from code integration, through building and testing, to deploying the DUT. To effectively design such a pipeline, it's beneficial to utilize various tools that facilitate managing the CI/CD process.

One of the tools for managing the CI/CD process is Jenkins. Jenkins enables the automatic execution of building, testing, and deploying DUT processes, significantly speeding up the DUT's life cycle along with the software. Additionally, Docker can be used to containerize applications like TestPMD, allowing easy isolation of test environments and ensuring consistency across different stages of the pipeline. Meanwhile, Ansible can be utilized for automating deployment processes, accelerating and simplifying infrastructure management.

It's important to note that configuring the network topology for the traffic generator and DUT should occur at the highest level of abstraction and be passed down in the form of configuration files, such as YAML or JSON, to the lower testing layer. This makes maintaining all variables at the automation level much easier. The software interface, defining the test environment configuration, serves as a crucial link between the deployment abstraction and our testing framework. This interface, which can be utilized for test execution, is established using parameters or configuration files. For instance, in the case of Pytest, a defined software interface allows for seamless integration and execution of tests.

Describing the topology in a YAML file format is clear for the client and facilitates both setup and configuration of IP addressing for the test environment. Moreover, it clearly illustrates the topology, providing information about the number of servers, virtual machines, and the location of the traffic generator and DUT, making it easier for all stakeholders to understand the network configuration.

Choosing the right technologies for test automation

Choosing the right technologies for test automation is necessary to create an effective and efficient testing environment. There are many tools and technologies that can be used for this purpose, depending on specific project needs.

Docker is one of the fundamental tools that enables easy creation and management of containers, significantly facilitating test isolation. With Docker, we gain the ability to virtualize network services, allowing for the creation of additional services such as virtual routers like Bird or FRR. These services are isolated from the rest of the test environment, providing more flexible test management. For instance, Docker facilitates performance tests for protocols like LACP or LAG, and enables verification of switch-over time between interfaces.

Ansible is another tool worth considering for test automation. It allows for the automation of

configuration and management of the test environment. Additionally, Ansible can also serve as a form of documentation, enabling easy reproduction of the test environment. At the Ansible level, we can also easily verify the correctness of network interface configurations or bind them to DPDK drivers, allowing for quick identification of errors in the network topology.

Jenkins is a popular tool for building and deploying applications as well as running automated tests. It is also a great tool for managing test environments, allowing for the assignment of test environments to specific users and enabling division into different jobs, such as test environment deployment. When running tests, we can use additional abstractions, such as test levels or types of network topologies. We can have access to different topologies, each characterized by a different configuration. Depending on project needs, we can appropriately adjust and select technologies that best meet our requirements, ultimately contributing to cost reduction and the rapid delivery of quality information about our tested product.

The importance of CI/CD in test automation

Continuous integration/continuous deployment (CI/CD) has revolutionized software development by automating the process of building, testing, and deploying applications. CI/CD pipelines offer a streamlined approach to software delivery, allowing teams to release high-quality software quickly and efficiently.

One of the primary benefits of CI/CD is the automation of the entire software development life cycle. By automating tasks such as code compilation, testing, and deployment, CI/CD pipelines reduce the risk of human error and ensure that software is thoroughly tested before being released to production.

Automated testing is a crucial component of the CI/CD process. By automating tests, teams can quickly identify and fix bugs, ensuring that only high-quality code is deployed. Additionally, automated testing allows for the continuous monitoring of system performance, helping teams identify and address performance issues before they impact end-users.

Quality gateways are another important aspect of the CI/CD process. Quality gateways are automated checks that evaluate code quality, security, and performance before allowing it to be deployed. By automating these checks, teams can ensure that only code that meets predefined standards is deployed to production, reducing the risk of introducing bugs or security vulnerabilities.

Overall, CI/CD and the automation of testing and quality gateways offer numerous benefits, including faster time-to-market, improved code quality, and increased confidence in the software being released. By implementing CI/CD pipelines and automated testing, teams can deliver high-quality software more efficiently, ultimately providing a better experience for end users.

The importance frequency of conducting performance tests

The frequency of conducting performance tests is crucial for maintaining continuous control over system performance during SW development. Ideally, they should be performed daily or after every major code change. However, in practice, due to limited resources and the time required to conduct full performance tests, it may be preferable to perform them once a week or at other predetermined intervals, tailored to the project's needs.

RFC2544 compliance tests, due to their time-consuming nature, require special attention. We assume that a test scope covering packets of various sizes (e.g. from 64 to 1500 bytes) and applying eight iterations for the binary method, each lasting 60 seconds, the execution time for a single test can be approximately 56 minutes. Of course, there is the possibility of introducing different assumptions and exceptions that can shorten this process, such as setting a packet loss threshold after which the test

will be interrupted or other conditions affecting the acceleration of the process.

Additional client requirements should also be considered, such as different versions of IP addressing or IPv6, which can further increase the time complexity of the testing process. Another aspect worth considering is deciding whether tests should be automatically run at fixed times or triggered by changes in pull requests. If the latter approach is chosen, it may be necessary to consider alternative approaches to conducting RFC2544 tests and reducing benchmark analysis time. One effective strategy is to categorize tests based on their importance and resource requirements. For instance, tests involving different packet sizes could be run daily, while others are scheduled to run weekly.

By reducing the frequency of certain tests, teams can optimize their testing process, saving time and resources while still ensuring that critical functionality is thoroughly tested. However, it's essential to strike a balance between test frequency and coverage. Running tests less frequently may reduce testing overhead, but it also means that performance issues may not be detected as quickly. Careful planning and optimization are required to achieve the most efficient performance testing process, ensuring that changes in performance are detected promptly while minimizing testing overhead.

Choosing a testing framework

The choice of testing framework is important for effective test automation. A testing framework provides a structured environment for organizing and executing tests, making it easier to manage and maintain test suites. Two popular frameworks that we recommend are Pytest and PeopleSoft Test Framework or PTF. These frameworks are widely used in the industry and offer a range of features to support test automation.

A testing framework typically consists of several key components, including:

- Test discovery: allows tests to be automatically discovered and executed.
- Fixture management: provides a way to set up preconditions and clean up after tests.
- Assertions: enables the verification of expected outcomes.
- Reporting: generates detailed reports on test results.

When choosing a testing framework, it's essential to consider factors such as ease of use, scalability, and community support. Pytest and PTF excel in these areas, making them excellent choices for test automation projects.

Pytest is one of the most popular frameworks for automated testing in Python. It is characterized by its simplicity, making it a preferred choice for many programmers. Its flexibility allows for easy test creation, management, and execution, making it an attractive tool for teams involved in test automation. Choosing Pytest as a wrapper for the TRex traffic generator can be advantageous due to its clear logs, ease of maintenance, and rich documentation. However, integrating with the TRex API may require some practice, especially initially when lacking experience with this Python interface. Nonetheless, it is worth taking the time to understand the TRex API documentation; it is a valuable resource for leveraging the benefits of using Pytest in conjunction with TRex. One of the significant advantages is that both the TRex API and Pytest are written in Python. This commonality allows for the seamless creation and execution of test scenarios in Python, streamlining the automation process and making it easier to write and maintain test scripts.

PTF largely relies on the Python unit testing library. It is a tool that enables clear and efficient creation of tests and configuration of the testing environment. By using a similar structure to unit tests, developers can feel comfortable working with PTF, which accelerates the learning and adaptation process of this tool. One of its main advantages is its flexibility in creating

network packets, similar to Scapy, combined with high performance. Additionally, PTF offers good documentation and community support, facilitating problem-solving and test development.

In summary, the choice between Pytest and PTF depends on individual needs, preferences, and project context. Both tools have their advantages and can be effective solutions for test automation, especially when combined with the TRex traffic generator.

Pass/fail criteria for performance tests

Determining pass/fail criteria for performance tests can be challenging because it depends on individual requirements and project context. However, we can go beyond simply collecting measurements during test scenario execution and establish detailed criteria. In performance testing, a test passes when all measurements are collected accurately and meet the predefined criteria. There are certain indicators that can serve as basic assessment criteria, such as response time, transaction performance, system load, or resource consumption.

When verifying the RFC2544 standard, which serves as our initial approach to test automation, we adopt a simple criterion: achieving a result for eight iterations in each packet. If there is no packet loss for eight iterations, meeting our expected condition (packet loss = 0), the test is considered passed. However, it's essential to note that the absence of packet loss in the first eight iterations doesn't guarantee that the solution is issue-free. Issues may arise after a longer period or a greater number of system operation iterations, or under specific load conditions and packet sizes. In addition, passing this test does not necessarily mean that the DUT is free of bugs in processing packets of a given size.

These issues may become apparent under different parameters of the test scenario, such as a greater number of iterations. Moreover, the measured value may not always be the maximum achievable. For example, allowing a small amount of packet loss may lead to higher data transmission speeds, which, from a product perspective, could be acceptable, at least at a certain stage of development.

Therefore, as the project develops, more advanced evaluation criteria can be considered, such as minimum acceptable values for specific parameters, e.g. packets per second. In this case, we can dispense with eight iterations (the number of iterations for RFC2544, which we refer to in this ebook, has been developed based on our past projects and their requirements) for each packet and focus on the minimum expected results.

However, such advanced criteria may be considered in later stages of the project, once more concrete goals and requirements have been established. It is also helpful to monitor (with Grafana or other dynamic analysis tools) and analyze test results to identify potential issues and causes of performance degradation, allowing for ongoing adjustment of assessment criteria to project needs.

Reporting and results

We have already received the initial results for RFC2544 or latency tests. The form of the raw data we received depends on the testing scenario implemented. It might be that we used pre-defined scenarios from TRex, such as NDR, or we might have crafted our test scenarios independently, generating measurement data in the form of a CSV file.

For our example, let's assume we received a table that defines statistics from the TRex API, such as the number of packets sent and received, along with other detailed information available in the TRex documentation. The rows of the table represent different packet sizes, while the columns provide information about the average latency for each packet, the amount of data sent and received, and other details.

It is important to remember that statistics in the TRex tool can be divided into two categories: global and per stream. Global statistics, which include data about all packets visible on the ports of our traffic generator, might include packets such as ARP from switches or other servers dispersed by the kernel, which could distort the test results. These global statistics can be useful during the initial setup of the test environment, for example, when we want to check our traffic generator for potential hardware or software defects.

Per stream statistics, which are available in TRex, provide detailed data concerning the packets sent and received on an individual stream basis. This approach is facilitated by the TRex parser and the addition of special tags to each stream during packet transmission and data collection as they are received by TRex. Such detailed analysis allows us to precisely define the test scenario for RFC2544 tests and observe specific results for the zero packet scenario.

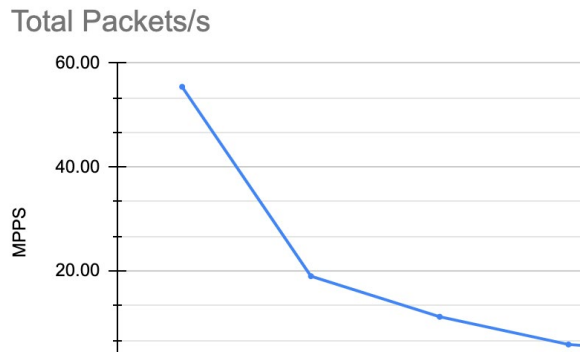
Below is an example of a table for the data received regarding the number of packets sent and received per second. In the table, it can be seen that traffic was symmetrically sent from two ports: p0 and p1. This traffic was looped at the level of the reflector added to the device under test:

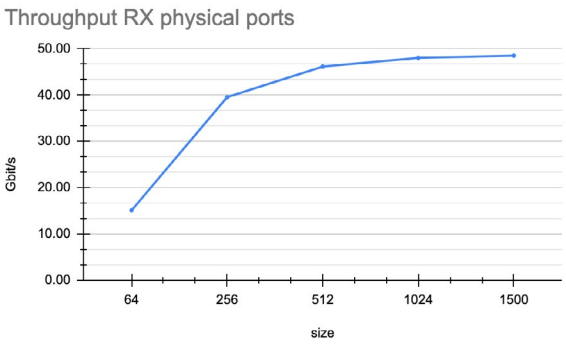
Pkt. size	p0/p1 loss	p0 TX PPS	p0 RX PPS	p1 TX PPS	p1 RX PPS	Total
64	0.00000	13,847,889	13,847,889	13,869,911	13,869,911	55,435,599.80
256	0.00000	9,490,135	9,490,135	9,517,789	9,517,789	18,980,270.80
512	0.00000	5,581,686	5,581,686	5,595,406	5,595,406	11,163,372.60
1024	0.00000	2,916,201	2,916,201	2,923,478	2,923,478	5,832,402.40
1500	0.00000	2,012,899	2,012,899	2,017,725	2,017,725	4,025,798.20

Throughput, or bandwidth, is a measure of the amount of data transmitted through a network over a specified time, usually given in bits per second (bps) or Gb per second. In the context of RFC 2544 tests, throughput is one of the key indicators of network efficiency, showing the maximum speed at which data can be transmitted through the network without packet loss.

Pkt. size	p0/p1 loss	p0 RX bytes	p1 RX bytes	p0 RX Gbit/s	p1 RX Gbit/s	Total
64	0.00000	941,656,472	943,153,921	7.53	7.55	15.08
256	0.00000	2,467,435,204	2,474,625,192	19.74	19.80	39.54
512	0.00000	2,880,150,131	2,887,229,702	23.04	23.10	46.14
1024	0.00000	2,997,854,834	3,005,335,692	23.98	24.04	48.03
1500	0.00000	3,027,400,246	3,034,657,798	24.22	24.28	48.50

For better understanding and analysis of throughput and packets per second (PPS) test results, it is recommended to visualize them using line graphs. This type of visualization allows for quick assessment of result variability, identification of trends, and potential network issues. Displaying data on graphs also facilitates easier comparison of results between different ports or test conditions.





Thanks to having detailed and separate statistics for each stream, we can more precisely analyze and interpret the results of performance tests, which is crucial for ensuring the highest quality and reliability of the tested product.

For a more detailed report and analysis of our tested device's performance, it's also worth considering information about the maximum theoretical throughput or PPS . To calculate the, for example, the theoretical maximum PPS for an RFC2544 report, taking into account a device under test with a throughput of up to 50 Gbps, you need to consider several key factors such as DUT throughput, packet size, and the hardware characteristics of the network devices. Here's how you can do it:

1. Determine the DUT throughput
   This is the maximum theoretical throughput of your network device, usually expressed in gigabits per second (Gbps). In this case, we're dealing with a DUT with a throughput of up to 50 Gbps, which means 50,000,000,000 bits per second.

2. Choose the packet size
   Packet size is crucial as it affects the number of packets that can be transmitted over the network per second. Typical packet sizes include 64 bytes, 128 bytes, 512 bytes, 1024 bytes, etc.

3. Calculate throughput to PPS
   To convert the DUT throughput to the maximum number of packets per second (pps), use the following formula:

$$\text{PPS} = \frac{\textit{DUT throughput (in bits per second)}}{\textit{packet size (in bits)} + \textit{overhead bits}}$$

   Overhead bits include additional bits required to transmit the packet, including e.g. Ethernet headers, IP headers, tunneling, VLANs, etc. For example, assuming a 50 Gbps link and a packet size of 64 bytes (512 bits), and assuming overhead is 18 bytes (144 bits), the calculations would look like this:

$$\text{PPS} = \frac{50{,}000{,}000{,}000}{512 + 144} = \sim 71{,}225{,}071\ \textit{pps}$$

4. Consider other factors
   When calculating the theoretical maximum pps, it's also worth considering hardware limitations such as processor speed, memory capabilities, and other specific features of network devices that may affect actual performance. These calculations are simplifications, and under real conditions, a detailed analysis may be necessary.

When reporting on performance tests such as RFC2544 (throughput, latency, frame loss) and latency measurements, it's crucial to customize the reports to meet the needs of different audiences within an organization. Technical teams and business stakeholders each have unique expectations regarding the content and format of these reports. Here are some recommendations for crafting reports that resonate with each group.

Reports for technical teams

Detailed technical data:

- Complete results from each test, including throughput, latency, frame loss.
- Comprehensive details on test configurations, software versions, hardware, and other technical specifics that influenced the test results.

Analysis and conclusions:

- Detailed examination of use cases and test scenarios, pinpointing potential causes of issues identified during the tests.
- Recommendations for further actions such as configuration changes, software updates, or the need for additional testing.

Visualizations:

- Charts and graphs that display test results over time, helping to understand trends and anomalies.
- Diagrams of the testing architecture and the network topologies used.

Reports for business and marketing teams

Performance summary:

- A general summary of the test results, emphasizing how they meet business objectives such as reliability, performance, and scalability of the system.
- Highlights of key achievements and potential risks to the project or product.

Business impact:

- Analysis of how the test results impact service quality for customers, the ability to launch new products or services, and potential financial benefits.
- Examples of how improvements can enhance customer satisfaction and market penetration.

Visualizations and infographics:

- Easy-to-understand charts and infographics that summarize key data and results, suitable for inclusion in annual reports or promotional materials.
- Charts showing performance outcomes in relation to business goals, such as achieving performance levels compared to competitors.

Common elements for all reports and summary

This ebook provides an overview of key findings and conclusions, ensuring quick comprehension for all stakeholders.

It begins with an explanation of the testing methodology, detailing the procedures used, including the test setup, tools, and standards employed. The report then highlights essential points that recipients should take into account after reviewing the findings.

Customizing reports to meet the needs of different audience segments is crucial for effective communication of results and subsequent decision-making. Tailoring reports to specific stakeholders ensures that the information presented is both relevant and actionable. For technical teams, this approach allows for an in-depth analysis of performance metrics, identification of potential bottlenecks, and optimization of system parameters for peak performance. Conversely, business units can concentrate on high-level performance indicators to understand how system performance aligns with overall business objectives.

Automating the report generation process improves the efficiency of report writing. Automated reports come with preformatted tabular data, charts, and environment configurations, ensuring consistency and ease of comparison. Additionally, automation saves time and effort, enabling teams to focus on analyzing results rather than formatting them. This streamlined process accelerates decision-making and facilitates a faster response to performance issues.

By standardizing report formats and utilizing tools like Pandas for analysis, teams can derive deeper insights from performance test results. Unified reports simplify the comparison of results over time and across different test scenarios, aiding in trend analysis and identifying performance patterns.

In conclusion, customizing and automating performance test reports not only improves communication between technical and business teams but also streamlines the analysis process, leading to more informed decision-making and better overall system performance.

About the author



Jarosław Trepkowski
Senior QA Engineer

Jarosław is a highly skilled Senior Software Engineer with over a decade of experience, specializing in software-defined networking and network function virtualization. He is an expert in cutting-edge networking technologies and has profound expertise in quality assurance automation, designing and implementing robust test processes and tools to ensure the delivery of top-notch software solutions. Additionally, Jarosław is a sought-after consultant in the field of QA, where he provides strategic guidance and innovative solutions to enhance testing efficiency and effectiveness.

About CodiLime

Since 2011, CodiLime has been the engineering partner of choice for semiconductor companies, networking vendors, telecom services, and software solution providers.

We're home to 300 top-notch software developers, network engineers, DevOps experts, and solution architects. We appreciate long-term collaborations above all, as well as our partners. Below are some names that have already trusted us:

CodiLime aims to link network engineering talent with business domain expertise – we focus on five N.E.E.D.S. - Networks, Equipment, Environment, Data and Security.

contact@codilime.com