υTNT: Unikernels for Efficient and Flexible Internet Probing

Maxime Letemple[1], Gaulthier Gain[2], Sami Ben Mariem[2], Laurent Mathy[2], and Benoit Donnet[2]

[1]*Institut Polytechnique de Bordeaux, France*
[2]*Université de Liège, Montefiore Institute, Belgium*

Abstract—**Over the past two decades, network measurement infrastructures have witnessed significant development and widespread adoption. Internet measurement platforms have become common and have demonstrated their relevance in Internet understanding and security observation. However, despite their popularity, those platforms lack of flexibility and reactivity, as they are usually used for longitudinal measurements. Consequently, critical security and Internet-related events may evade detection. Concurrently, the evolution of operating systems towards virtual machines (VMs) has been notable, particularly with the emergence of unikernels—ultra-lightweight VMs tailored for specific applications by including only the essential components. This paper advocates for the integration of unikernels into measurement infrastructures to enhance their flexibility and efficiency. We introduce υTNT, a proof-of-concept unikernel-based implementation of TNT, a `traceroute` extension capable of discovering MPLS tunnels. This paper documents the full toolchain for porting TNT into a unikernel and evaluates υTNT's performance in comparison to conventional methodologies. Additionally, we explore a practical use case scenario demonstrating the utility of υTNT. The source code for υTNT is publicly available on Gitlab.**

I. Introduction

For more than twenty years now, numerous Internet measurement platforms [1]–[6] have emerged, primarily led by researchers and using dedicated hardware or distributed infrastructures [3], [4], [7]. These platforms collect data for research purposes and often face challenges like hardware compatibility and irregular data collection. Unfortunately, none of these approaches are flexible and easily deployable. As such, they are not built to quickly react to events, either related to security, network outages [8], [9], or even Internet topology dynamicity discovery [10], [11]. By quickly, we mean it should be instantiated on-demand (loading time must be as quick as possible), should require the lowest memory footprint, and shutdown when the measurement is over.

Simultaneously, advances in operating systems have led to the evolution of virtualization technologies. While virtual machines (VMs) were initially popular, their resource-heavy nature prompted the shift towards containerization like Docker [12], which shares the host OS kernel. However, containers have security concerns due to their large attack surface [13], [14]. To strike a balance between performance and isolation, a newer paradigm emerged: *unikernels* [15]. Unikernels are ultra-lightweight VMs tailored for specific applications, eliminating the need for a host OS. Including only essential OS components offers improved performance and a reduced attack surface, making them a promising alternative to VMs and containers [16]–[19]. Unikernels can further enhance their efficiency through memory deduplication [20]–[22]. This process, essential in virtualized settings, minimizes memory usage by identifying and consolidating identical memory pages across various instances. Technologies such as Kernel Samepage Merging (KSM) [23] within the Linux Kernel play a significant role in optimizing memory utilization within these environments.

Although unikernels require manual porting of applications, frameworks like Unikraft [24], [25] have eased the process, aiming for small image sizes and quick boot times. The Unikraft framework combines the modularity of micro-kernels with the design of monolithic kernels, enabling specialization through well-defined API boundaries. It relies on micro-libraries[1] and a user-friendly build system to facilitate the porting of applications to Unikraft.

This paper proposes unikernels as the foundation for efficient measurement infrastructures and introduces υTNT, a unikernel-based implementation of TNT, an extension able to reveal MPLS tunnels [26]–[28]. The paper presents υTNT as a flexible and efficient solution, demonstrating reduced CPU and memory consumption compared to traditional implementations. Additionally, it discusses potential use cases where the flexibility of unikernel-based probing tools could be advantageous and evaluates its performance in such scenarios. An extended version of this paper is available on Arxiv [29].

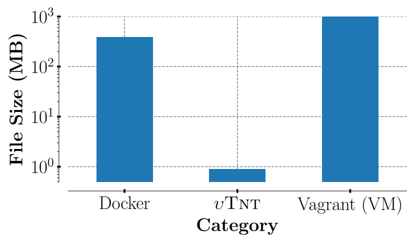
II. υTNT

This section introduces υTNT, the manual porting of TNT (a modified `traceroute` [30] driver for `scamper` [31]) into a unikernel using Unikraft. It provides an overview of `scamper`, a modern `traceroute` implementation, and describes the implementation of υTNT.

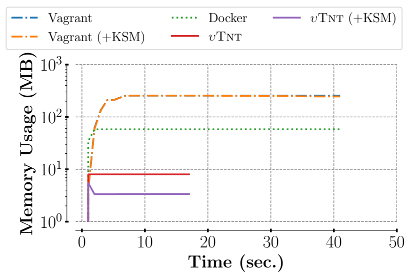
1) Overview: `scamper` extends traditional `traceroute` with additional tools like `ping` and DNS probing. TNT, built on `scamper`, detects MPLS tunnels [32] and identifies hardware vendors [33]. Initiating `scamper` involves running the main program and then executing the specific driver for desired measurements.

[1]Micro-libraries (micro-libs, for short) are software components which implement one of the core Unikraft APIs.

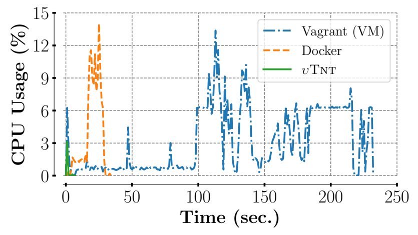
978-3-903176-64-5 ©2024 IFIP



(a) File size.



(b) Memory usage.



(c) CPU usage.

Figure 1. Filesize, memory and CPU usage of `TNT` ported and run in a Docker image, a Vagrant image, and an unikernel.

2) Implementation: To transform `TNT` into a unikernel, denoted as υ`TNT`, we use the Unikraft [18] and its toolsets [24] for the porting process. In Fig. 2, we outline the 3 general steps involved to port an application as unikernel: (*i*) Using a developer assisting tool [24] to perform the libraries matching and generating the required configuration files for Unikraft. (*ii*) Adapting the given application's codebase to support Unikraft paradigm (e.g., single address space and single process). (*iii*) Updating external libraries by modifying their codebase to be compliant with the ported application. These last two steps can either be skipped (no compilation or linking error) or be repeated several times until the application is successfully built using the Unikraft build system.

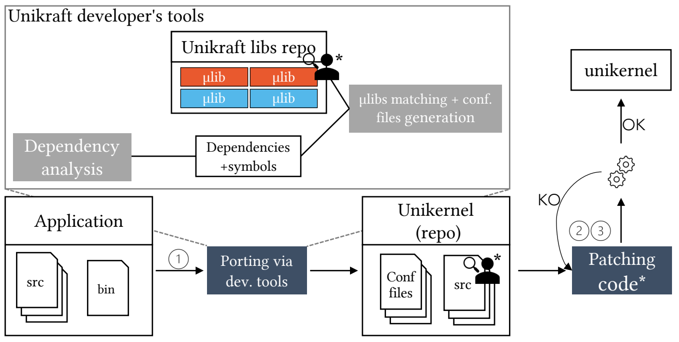


Figure 2. High-level overview of the different steps to port an application into a unikernel. The Unikraft developers tool is used to perform micro-libraries matching and to generate the required configuration files. After this step, the application's codebase has been patched to be compliant with Unikraft.

We made adjustments to `scamper` and `TNT` code, converted inter-process communication to multithreading, and ensured compatibility with Unikraft's libraries. Minor modifications were made to `lwip` [34] and `musl` [35] libraries for proper functionality. After completing these steps, υ`TNT` was successfully compiled and built. We give an overview of our changes in Table I. To count the lines of code (LoC), we used CLoc [36] and considered only the C/C++ files with their associated header files (only code without any comment). The last column gives the percentage of changes compared to the original codebase. All the patches containing our code/changes can be found with our setup in our GitLab repository [37].

Table I
OVERVIEW OF THE CHANGES MADE TO THE DIFFERENT LIBRARIES.

Micro Libraries	# File(s) updated	# LoC updated	% of changes
core (`ukboot`, etc)	/	/	/
`musl` (libc)	1	5	<0.01%
`lwip` (network)	3	12	0.01%
`scamper`	5	110	0.2%
`TNT`	1	70	1.8%
Total	11	197	/

III. Performance Evaluation

In this section, we showcase the experimental results obtained by comparing `TNT` across three architectures: a Debian virtual machine (VM) with Vagrant [38], Docker containerization, and υ`TNT` as a unikernel using Unikraft with Firecracker [39] hypervisor support. We aim to assess υ`TNT`'s performance in terms of memory usage, CPU utilization, file size, and total execution time. For a fair comparison, we standardized environments with identical `scamper` and `TNT` versions across all setups. Experiments were conducted on Debian GNU/Linux 11 with a Linux kernel 5.10.162. The host machine used for the experiments has 32 GB of RAM and an Intel (R) Xeon(R) CPU E5-2620v4 @2.10GHz with 16 cores. In addition, Unikraft Pandora 0.15.0 has been used for the following experiments.

Results indicate υ`TNT`'s significant advantages. As depicted in Fig. 1a, the υ`TNT` unikernel has a tiny file size, occupying less than 1 MB of disk space. This impressive size reduction can be attributed to Unikraft's capability to execute *Dead Code Elimination* (DCE – i.e., an optimization that removes code which does not affect the program results). In stark contrast, the Docker image is 430× larger than υ`TNT`. Similarly, the Vagrant image proved even more substantial, with an image 1200× larger than υ`TNT`.

As shown in Fig. 1b, memory consumption of υ`TNT` is also notably lower, around 7.2× less than Docker and 32× less than Vagrant. When KSM is utilized, υ`TNT`'s memory consumption drops further being 17× less than Docker and 74× less than the Vagrant configuration. CPU utilization shown in Fig. 1c is also lower with υ`TNT`, owing to its minimalist design and lightweight hypervisor.

In terms of deployment and execution time (creation, execution of `TNT` with specific parameters, instance destruction, and cleanup), υ`TNT` outperforms Docker and Vagrant as can be observed in Fig. 3. The execution time for υ`TNT` and Docker is relatively similar, as υ`TNT` involves initiating a unikernel from scratch, including to initialize essential components such as the memory, the scheduler, etc. However, the deployment time of υ`TNT` is the fastest due to its small image size unlike Docker and Vagrant which have larger images. All in all, deploying and executing υ`TNT` allows to have a gain of $2.4\times$ compared to Docker and $13\times$ compared to Vagrant.

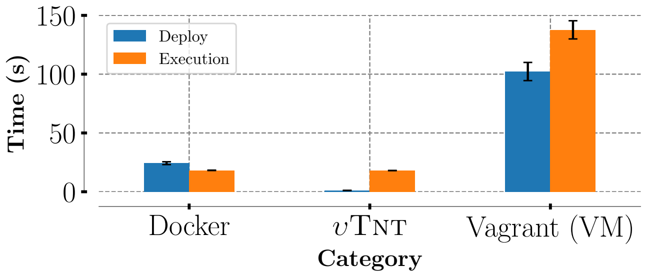


Figure 3. Total time (deployment + execution) to run `TNT`, compared to Docker and Vagrant. Using υ`TNT` gives the lower total time.

Moreover, υ`TNT` allows launching more instances within resource constraints: 1 GB of memory and a 25% CPU usage limit (4 cores). In this type of experiment, υ`TNT` outperforms Docker and Vagrant, supporting significantly more instances while maintaining efficient resource utilization as it can be observed in Fig 4.

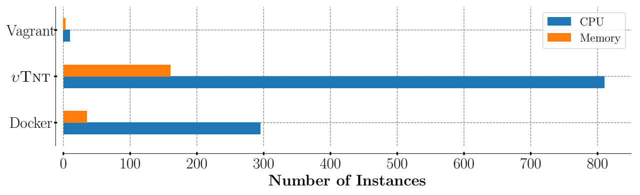


Figure 4. Maximum #instances running with a fixed memory and CPU budget.

Overall, υ`TNT` demonstrates superior performance across various dimensions compared to Docker and Vagrant, making it a compelling choice for network measurement applications.

IV. Flexible Deployment

In this section, we demonstrate the quick deployment of unikernels for network measurements anywhere on the Internet, orchestrated by a controller. For this experiment, we provisioned five remote servers across diverse locations using OVH cloud infrastructure [40]. A Python script acts as the controller, deploying instances on remote servers for measurement purposes. The controller triggers one deployment of a `TNT` instance per second for 60 seconds, records cumulative time for each node, and can adapt to high-request scenarios with two distinct behaviours: (i) either it waits for a node to become available, (ii) it ignores and discards the request if no node is available.

Results are shown in Table II. For the first behaviour, they indicate that υ`TNT` exhibits the fastest deployment and execution time, followed by Docker and Vagrant. Using υ`TNT` results in deployment speeds $5\times$ faster than Docker and $65\times$ faster than Vagrant.

Table II
Flexible deployment stats reported by the controller.

	Docker	**υTNT**	**Vagrant**
(i) Average time (sec)	17.3	3.5	216
(ii) Successful runs (/60)	18	55	5

In scenario (ii) where the controller ignores and discards the request, υ`TNT` achieves a 90% deployment success rate, compared to 28% for Docker and 8% for Vagrant. These results highlight υ`TNT`'s superior performance and scalability, making it an efficient choice for flexible deployment in network measurement applications.

V. Discussion

In this section, we explore two key aspects: porting to Unikraft and cloud deployment.

We opted for a manual approach for porting `TNT` to Unikraft due to the simplicity of the `scamper + TNT` codebase. However, Unikraft also offers binary compatibility, especially useful for closed-source or complex applications with a lot of shared libraries, eliminating manual porting but introducing potential overhead and compatibility issues. The choice between these two approaches ultimately lies with the developer and maintainer.

Regarding cloud deployment, our evaluations focused on the KVM platform and x86_64 architecture. However, Unikraft supports various platforms and architectures like Xen [41] and ARM, with expected similar performance results.

VI. Conclusion & Future Work

In this paper, we provided the complete toolchain when transforming `TNT` into υ`TNT`. We believe this toolchain could be the first step towards generalising unikernels in network infrastructures. Indeed, this paper has demonstrated the supremacy of υ`TNT` over more traditional approaches. Further, this paper also discussed a case study in which the flexibility and responsiveness of υ`TNT` could find a suitable usage, compared to more traditional approaches once again.

As future work, we are considering to study the implementation of unikernels in devices with few resources (embedded systems or IoT) in order to run monitoring/telemetry probing tools. In addition, we will further assess binary compatibility's overhead and compatibility.

Acknowledgments

This is supported by the CyberExcellence project funded by the Walloon Region, under number 2110186, and the Feder CyberGalaxia project.

References

[1] V. Bajpai and J. Schonwalder, “A survey on Internet performance measurement platforms and related standardization efforts,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1313–1341, April 2015.

[2] B. Huffaker, D. Plummer, D. Moore, and k. claffy, “Topology discovery by active probing,” in *Proc. Symposium on Applications and the Internet (SAINT)*, January 2002.

[3] k. claffy, Y. Hyun, K. Keys, M. Fomenkov, and D. Krioukov, “Internet mapping: from art to science,” in *Proc. IEEE Cybersecurity Application and Technologies Conference for Homeland Security (CATCH)*, March 2009.

[4] RIPE Network Coordination Center, “Atlas,” 2010, see https://atlas.ripe.net.

[5] P. Gill, C. Diot, L. Y. Ohlsen, M. Mathis, and S. Soltesz, “M-lab: User initiated internet data for the research community,” *ACM SIGCOMM Computer Communication Review*, vol. 1, no. 52, January 2022.

[6] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, “iPlane: An information plane for distributed services,” in *Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, November 2006.

[7] PlanetLab Consortium, “PlanetLab project,” 2002, see http://www.planet-lab.org.

[8] G. Aceto, A. Botta, P. Marchetta, V. Persico, and A. Pescapé, “A comprehensive survey on Internet outages,” *Journal of Network and Computer Applications*, vol. 113, pp. 36–63, July 2018.

[9] M. Safaei Pour, C. Nader, K. Friday, and E. Bou-Harb, “A comprehensive survey of recent internet measurement techniques for cyber security,” *Computers & Security*, vol. 128, May 2023.

[10] B. Donnet, “Incentvies for BGP guided IP-level topology discovery,” in *Proc. Traffic and Measurement Analysis Workshop (TMA)*, May 2009.

[11] B. Donnet and T. Friedman, “Internet topology discovery: a survey,” *IEEE Communications Surveys and Tutorials*, vol. 9, no. 4, December 2007.

[12] Hykes, S. et al, “Docker,” https://docker.com/, 2018, [Last Accessed: October 26th, 2023].

[13] E. Kovacs, “Docker fixes vulnerabilities, shares plans for making platform safer,” http://www.securityweek.com/docker-fixes-vulnerabilities-shares-plans-making-platform-safer, 2014, [Last Accessed: October 26th, 2023].

[14] A. Grattafiori, “Understanding and hardening linux containers,” https://research.nccgroup.com/2016/05/05/understanding-and-hardening-linux-containers/, 2016, [Last Accessed: October 26th, 2023].

[15] A. Madhavapeddy and D. J. Scott, “Unikernels: The rise of the virtual library operating systems,” *Communications of the ACM*, vol. 57, no. 1, pp. 61–69, January 2014.

[16] A. Kantee, “Flexible operating systems internals: The design and implementation of the anykernel and rump kernels,” Ph.D. dissertation, Aalto University, 2012.

[17] A. Kivity, D. Laor, G. Costa, P. Enberg, N. Har’El, D. Marti, and Z. V., “OSv–optimizing the operating system for virtual machines,” in *Proc. USENIX Annual Technical Conference*, June 2014.

[18] S. Kuenzer, V.-A. Bădoiu, H. Lefeuvre, S. Santhanam, A. Jung, G. Gain, C. Soldani, C. Lupu, c. Teodorescu, C. Răducanu, C. Banu, L. Mathy, R. Deaconescu, C. Raiciu, and F. Huici, “Unikraft: fast, specialized unikernels the easy way,” in *Proceedings of the Sixteenth European Conference on Computer Systems*, ser. EuroSys ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 376–394. [Online]. Available: https://doi.org/10.1145/3447786.3456248

[19] H.-C. Kuo, D. Williams, R. Koller, and S. Mohan, “A Linux in unikernel clothing,” in *Proc. European Conference on Computer Systems (EuroSys)*, April 2020.

[20] N. Xia, C. Tian, Y. Luo, H. Liu, and X. Wang, “UKSM: Swift memory deduplication via hierarchical and adaptive memory region distilling,” in *Proc. USENIX Conference on File and Storage Technologies (FAST)*, February 2018.

[21] K. Miller, F. Franz, T. Groeninger, M. Rittinghaus, M. Hillenbrand, and F. Bellosa, “KSM++: Using I/O-based hints to make memory-deduplication scanners more efficient,” in *Proc. ASPLOS Workshop on Runtime Environments, Systems, Layering and Virtualized Environments (RESoLVE)*, March 2012.

[22] G. Gain, C. Soldani, F. Huici, and L. Mathy, “Want more unikernels? inflate them!” in *Proc. Symposium on Cloud Computing (SoCC)*, November 2022.

[23] A. Arcangeli, I. Eidus, and C. Wright, “Increasing memory density by using KSM,” in *Proc. Linux Symposium*, January 2009.

[24] G. Gain, “Unikraft Tools,” 2019, [Last Accessed: October 24th, 2023]. [Online]. Available: https://github.com/gaulthiergain/tools

[25] H. Lefeuvre, G. Gain, V.-A. Bădoiu, D. Dinca, V.-R. Schiller, C. Raiciu, F. Huici, and P. Olivier, “Loupe: Driving the development of OS compatibility layers,” in *Proc. ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, May 2024.

[26] B. Donnet, M. Luckie, P. Mérindol, and J.-J. Pansiot, “Revealing MPLS tunnels obscured from traceroute,” *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 2, pp. 87–93, April 2012.

[27] Y. Vanaubel, J.-R. Luttringer, P. Mérindol, J.-J. Pansiot, and B. Donnet, “TNT, watch me explode: A light in the dark for revealing MPLS tunnels,” in *Proc. IFIP Network Traffic Measurement and Analysis Conference (TMA)*, June 2019.

[28] J.-R. Luttringer, Y. Vanaubel, P. Mérindol, J.-J. Pansiot, and B. Donnet, “Let there be light: Revealing hidden MPLS tunnels with TNT,” *IEEE Transactions on Network and Service Management (TNSM)*, vol. 17, no. 2, pp. 1239–1253, June 2020.

[29] M. Letemple, G. Gain, S. B. Mariem, L. Mathy, and B. Donnet, “utnt: Unikernels for efficient and flexible internet probing,” 2024.

[30] V. Jacobson et al., “traceroute,” UNIX, man page, 1989, see source code: ftp://ftp.ee.lbl.gov/traceroute.tar.gz.

[31] M. Luckie, “Scamper: a scalable and extensible packet prober for active measurement of the Internet,” in *Proc. ACM Internet Measurement Conference (IMC)*, November 2010.

[32] Y. Vanaubel, P. Mérindol, J.-J. Pansiot, and B. Donnet, “Through the wormhole: Tracking invisible MPLS tunnels,” in *Proc. ACM Internet Measurement Conference (IMC)*, November 2017.

[33] Y. Vanaubel, J.-J. Pansiot, P. Mérindol, and B. Donnet, “Network fingerprinting: TTL-based router signature,” in *Proc. ACM Internet Measurement Conference (IMC)*, October 2013.

[34] Dunkels, A. et al., “lwip: Lightweith IP,” [Last Accessed: October 24th, 2023]. [Online]. Available: https://savannah.nongnu.org/projects/lwip/

[35] Felker, R. et al., “musl,” [Last Accessed: October 24th, 2023]. [Online]. Available: https://musl.libc.org

[36] Al Danial, “cloc: Count lines of code,” [Last Accessed: October 30th, 2023]. [Online]. Available: https://github.com/AlDanial/cloc

[37] G. Gain, “utnt,” [Last Accessed: May 6th, 2024]. [Online]. Available: https://gitlab.uliege.be/Gaulthier.Gain/utnt

[38] Vagrant, “Vagrant cloud,” [Last Accessed: October 24th, 2023]. [Online]. Available: https://app.vagrantup.com/boxes/search

[39] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.-M. Popa, “Firecracker: Lightweight virtualization for serverless applications,” in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, February 2020.

[40] OVH, “Global cloud service provider | ovhcloud,” [Last Accessed: October 24th, 2023]. [Online]. Available: https://us.ovhcloud.com

[41] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” in *Proc. ACM symposium on Operating systems principles (SOSP)*, October 2003.