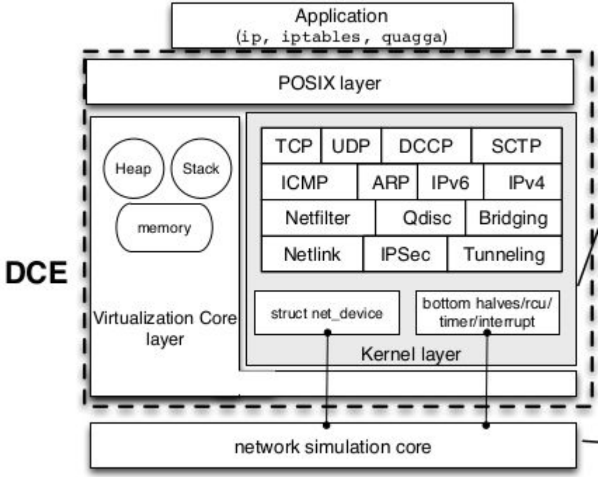
Revitalizing ns-3's Direct Code Execution

Parth Pratim Chatterjee, Thomas R. Henderson

WNS3 2022 Virtual Conference

What is DCE[1] - Direct Code Execution ?



Excerpt of Figure 1 from [1]

- ➔ Helps **run real C/C++ applications** in an ns-3 simulation context
- ➔ Supports both **Linux** and native ns-3 as **network stacks**.
- ➔ Complex protocols like BGP and OSPF supported
- ➔ **Two Modes :** (i) User Mode (ii) Kernel Mode

[1] *Hajime Tazaki, Frederic Urbani, Emilio Mancini, Mathieu Lacage, Daniel Camara, Thierry Turletti, and Walid Dabbous. Direct code execution: Revisiting library os architecture for reproducible network experiments. In Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT '13, pages 217–228, New York, NY, USA, 2013. Association for Computing Machinery.*

Why use DCE? Can’t I just use containers/namespaces?

- ➔ DCE executes simulations in a **single ns-3 process.**
    - ◆ Multi-process emulations can be complex to debug or to reproduce results
    - ◆ DCE simulations can run in simulation time-- not constrained to run in real-time
- ➔ However... to do this, DCE must support some sophisticated techniques:
    - ◆ Selected POSIX system calls need to be replaced with ns-3 equivalents
    - ◆ Executables (like ping) must be built in a special way such that they are shared libraries and not processes
    - ◆ Resources must be carefully tracked and managed (stack, heap, and global memory, file I/O, etc.)

GSoC ‘21 Project [1]

Problems Addressed

- ➔ DCE relied on some bypassing tricks of the standard library (glibc). Because these bypasses could also be exploited by attackers, the glibc developers blocked such bypasses.
- ➔ Rapid pace of Linux kernel code change makes upgrading the Linux DCE stack very difficult to maintain.

Results

- ➔ Extended DCE to build a customized glibc preserving previous bypasses.
- ➔ Evaluated Linux Kernel Library, and later upgraded kernel stack from 4.4 to 5.10.
- ➔ Improvised Docker based build for DCE, to make it user-friendly and reduce disk usage.

[1] : https://ns-3-dce-linux-upgrade.github.io

DCE Problems Addressed

1. LibIO VTable Redirection Limitation

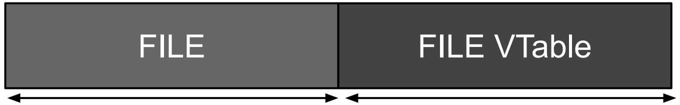
2. Position Independent Executable Loading & Usage

3. Outdated Linux Networking Stack

4. Not Platform Agnostic

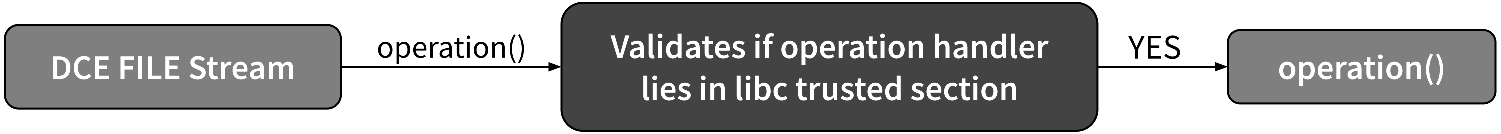
LibIO Vtable Hijacking & FILE VTable

- ➔ Multiple DCE applications writing to output streams, which might cause **cluttered & unordered logs**.
- ➔ DCE **hijacks** FILE stream operations, to redirect application output to **organized log folders**.
- ➔ This was possible by **overwriting** the FILE's VTable with DCE's handlers for some of the FILE operations.
- ➔ DCE took advantage of the **contiguous allocation** of the FILE vtable, when a FILE stream was created
- ➔ Till libc-2.25, this hack was possible, but in later libc releases, this security flaw was fixed.
- ➔ A contributor suggested using **fopencookie**, but it led to inconsistent logs for certain tests involving execve



Contiguous Memory Locations

LibIO Vtable Hijacking & FILE VTable (cont.)



- ➔ Every FILE stream operation is **executed only if** the operation handler lies in **libc's trusted section**
- ➔ We compiled a libc which avoids the process from aborting if the above validation fails.
- ➔ DCE waf wscript was modified to build DCE libraries with the new patched glibc as the system root.

PIE executables as dynamic libraries

- → **PIE** : Position Independent Executable, compiled with **-fPIC** and **-pie**
- → DCE requires PIEs to be loaded as dynamic libraries as DCE is **single threaded**.
- → libc developers spotted cases where loading a PIE as a library could cause relocation errors, and fixed it.
- → There were relocation issues when PIEs were loaded as dynamic libraries.

Solutions :

- → **Strip** the PIE identifier **flag** from the dynamic section when DCE copies the files to its local cache.
- → Alternatively, we can **patch the libc** to remove the additional check in dlopen

Linux Networking Stack - DCE

- ➔ DCE lets you use Linux as the networking stack for your simulation scripts.
- ➔ Two potential Linux-as-a-library frameworks :
  1. **LKL**
  2. **LibOS**
- ➔ Previous developers of DCE used LibOS.
- ➔ Hajime Tazaki founded LKL, which had broader applications and delivered to a wider range of use-cases.
- ➔ Previous DCE maintainer Matthieu suggested that LKL can be looked into, which is still a work in progress
- ➔ Due to the synchronization & preemption restrictions imposed by LKL, we moved on with LibOS.

Linux-As-A-Library Tools

	LKL	LibOS
Type	Full /arch level port	Partial Linking
Kernel Features	All features supported by base kernel	Selective feature support
Synchronization	Uniprocessor, CPU Lock based	Requires host API for yielding & synchronization
Maintenance	Easy upstream bug fix merging	Uncertainty to identify bug Irrelevant to Linux community
Preemption	Non-Preemptive	Preemptive (LWP)

Net-next-nuse-5.10

➔ Two development choices:

   i. Move from kernel 4.4.0 to 5.10, merging commits incrementally.

   ii. Start off with kernel–5.10 and port directly to net-next-nuse. ✔

➔ Linux-5.10 has features which can be interesting to the community. **Ex. BBR V2, TCP Prague**

➔ Can be a little difficult to move forward with next releases of Linux.

➔ Elf header manipulation libraries can be used to redirect **library symbols** to hijack internal kernel function calls, and map it to DCE’s glue code.

4. Not Platform Agnostic

Solution : Docker based build

Docker Compose Build for DCE

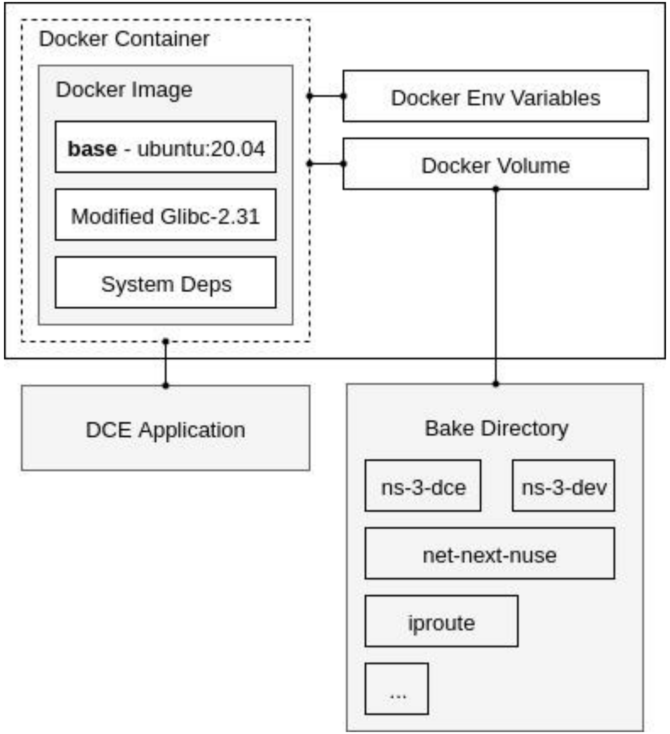
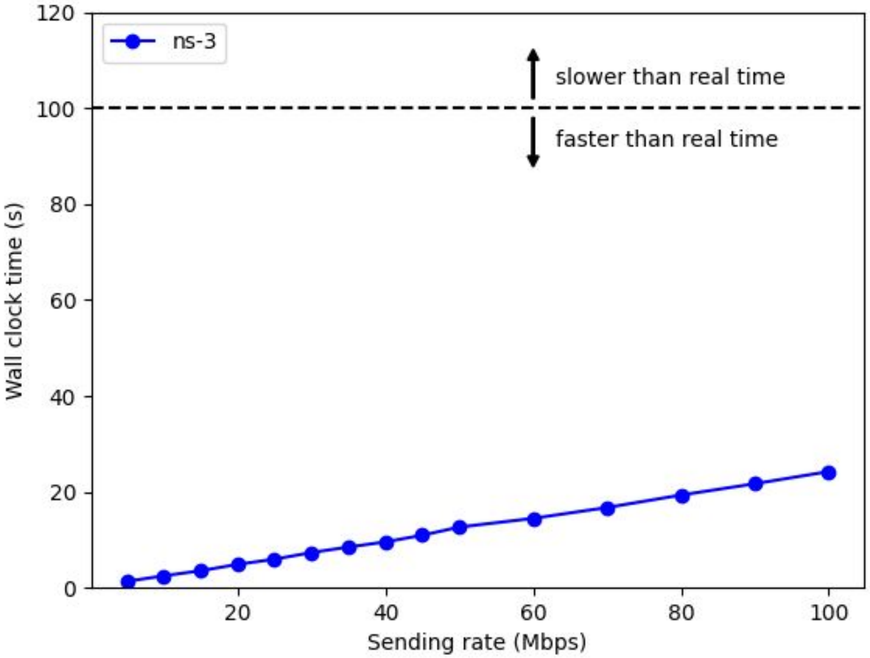


Fig 2. from Revitalizing ns-3's DCE Paper

- ➔ About ~**5GB** additional disk space was used in native build
- ➔ DCE Docker image of compressed size **256 MB** is downloaded, which decompresses to **~800MB**
- ➔ Docker reduces initial build time by ~**45%**
- ➔ Docker saves ~**4.2 GB** of disk usage
- ➔ Workspace space usage is same as older release i.e dce-1.11
- ➔ Docker environment has :
  - i. pre-installed patched libc(only needed libs & headers)
  - ii. all dependencies
  - iii. Shared host directory support
  - iv. Same build steps
  - v. Highly configurable (env variables & wscript)

Performance Evaluation



- ➔ Impact of different modes on execution time.
- ➔ Uses linear chain topology, with four hops.
- ➔ Expected linear relationship between sending rate and wall clock time, as number of hops per packet is kept constant

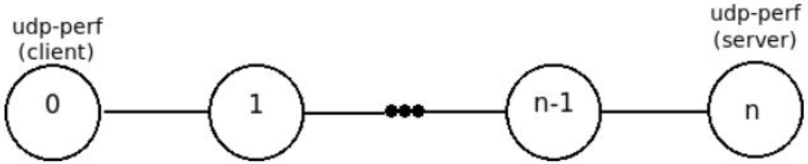
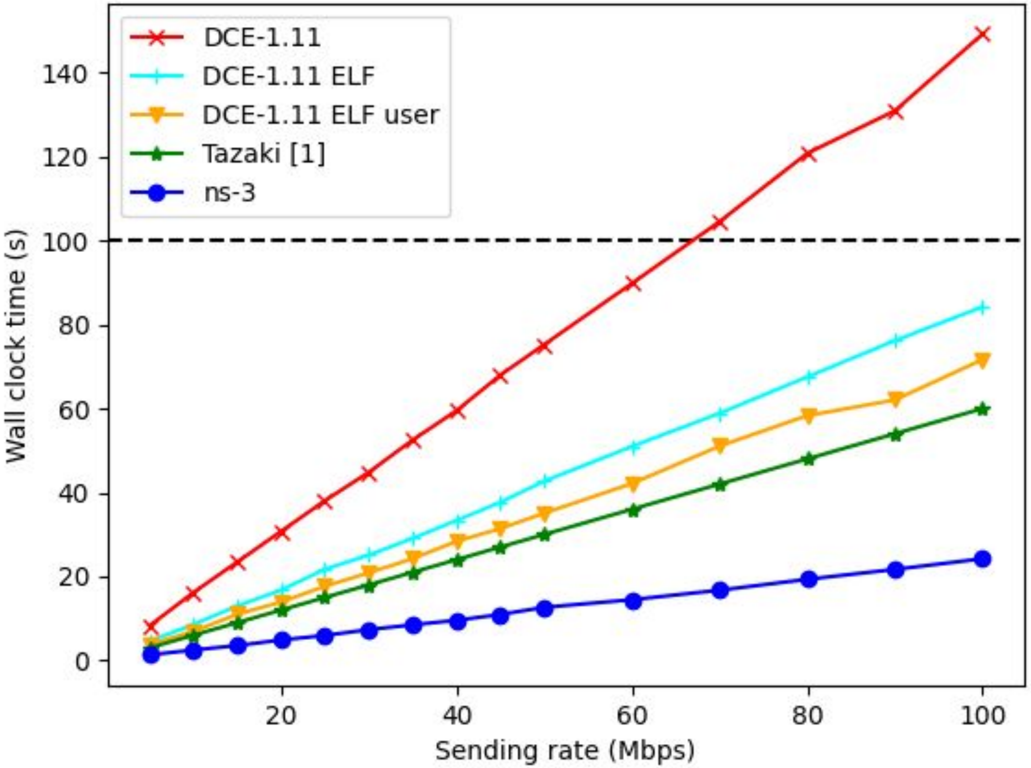


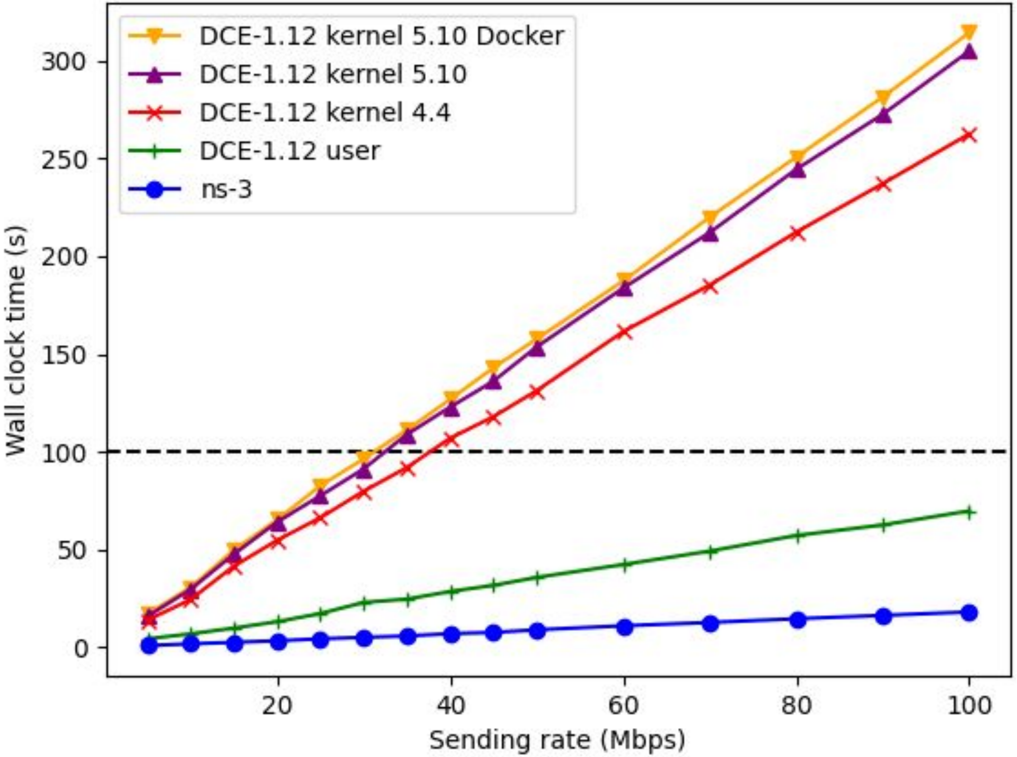
Figure 3: Simulation topology (redrawn from Figure 2 of [1])

Performance Evaluation



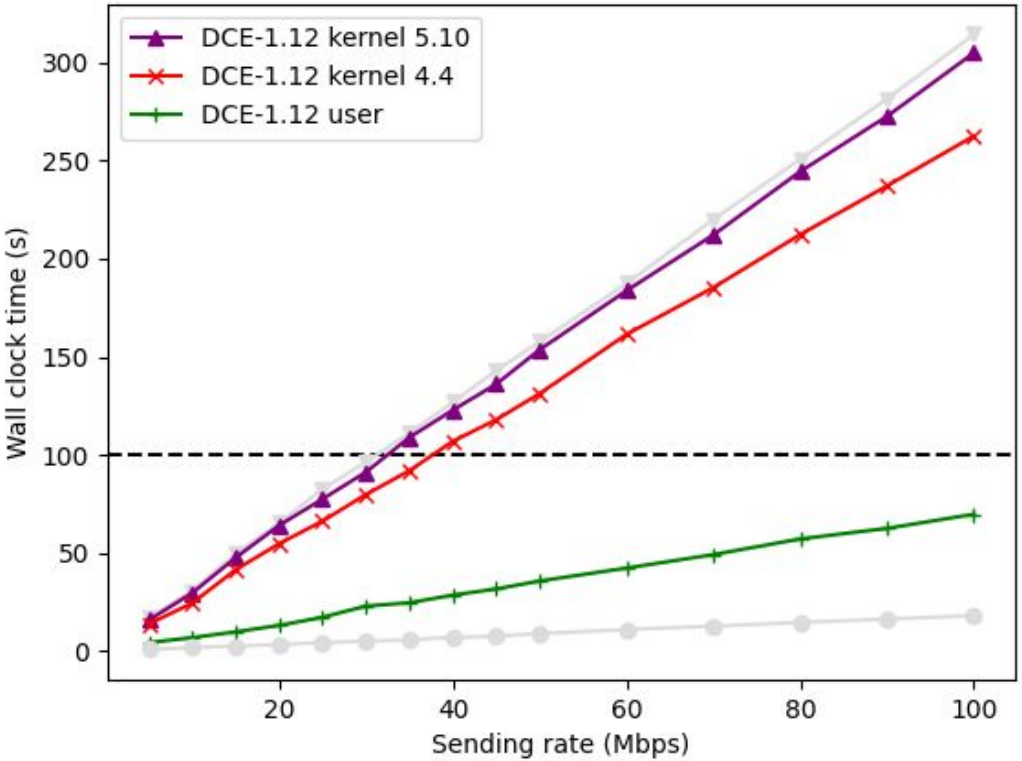
- ➔ We can notice the pure **ns-3** simulations to be the **fastest.**
- ➔ DCE has overheads
- ➔ Experiments from the CONext paper could not be perfectly reproduced due to lack of configuration information.
- ➔ DCE-1.11 (latest available DCE release, works on Ubuntu 16.04) runs have a lower runtime with **Elf-Loader**.

Performance Evaluation : Figure 6 from Paper



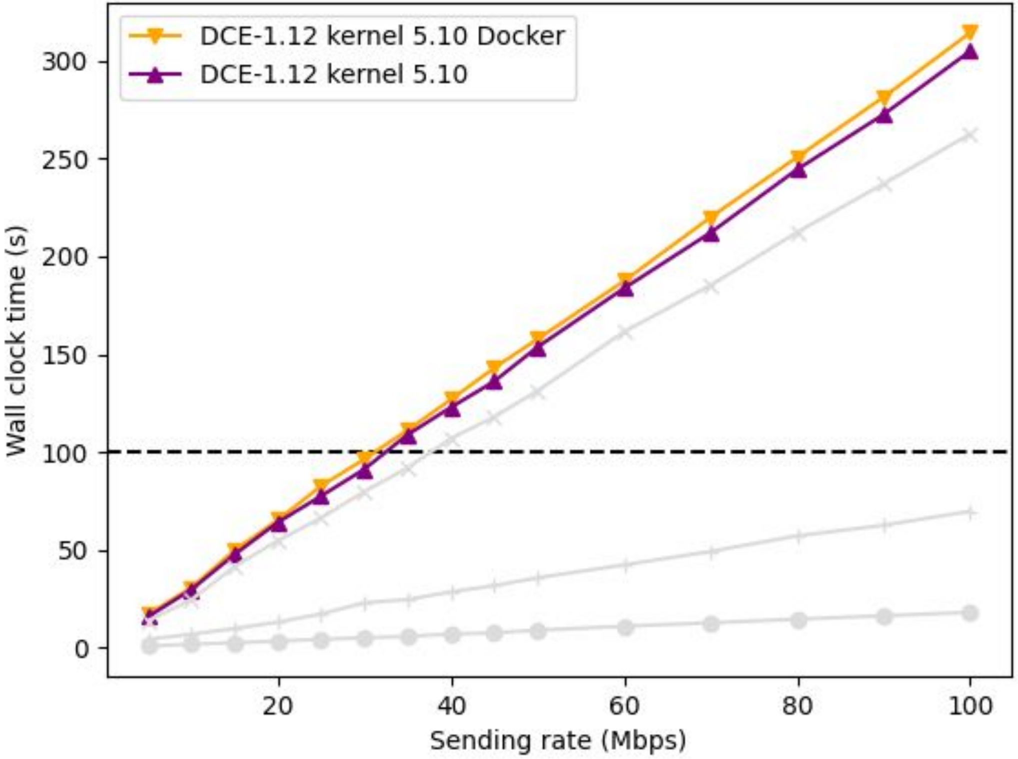
- ➔ Analysis of each curve has been made in next few slides
- ➔ Dce-1.12 is the next scheduled release, which is in review.
- ➔ Dce-1.12 uses the patched libc and is tested on Ubuntu-20.04

Performance Evaluation : Linux-5.10.0 Vs Linux-4.4.0



- ➔ Linux-5.10 runs on native build notice a **slight increase** in runtime when compared with Linux-4.4 native runs
- ➔ Lack of availability of **Elf-Loader** for libc-2.31 can be a reason for generally higher DCE 1.12 runtimes
- ➔ Requires further optimizations

Performance Evaluation : Docker Vs Native (Linux-5.10)



- ➔ Linux-5.10 docker runs **nearly overlap** with the native runs for Linux-5.10.
- ➔ Similar performance, with increased availability

Future Work

- ➔ IPV6 Support in net-nextt-nuse-5.10
- ➔ FILE VTable hijack native workaround
- ➔ Elf-Loader for Libc-2.31
- ➔ Support 3rd party libraries. Ex. Grpc, Tensorflow for C++

GSoC ‘21 Project [1]

Problems Addressed

- ➔ DCE relied on some bypassing tricks of the standard library (glibc). Because these bypasses could also be exploited by attackers, the glibc developers blocked such bypasses.
- ➔ Rapid pace of Linux kernel code change makes upgrading the Linux DCE stack very difficult to maintain.

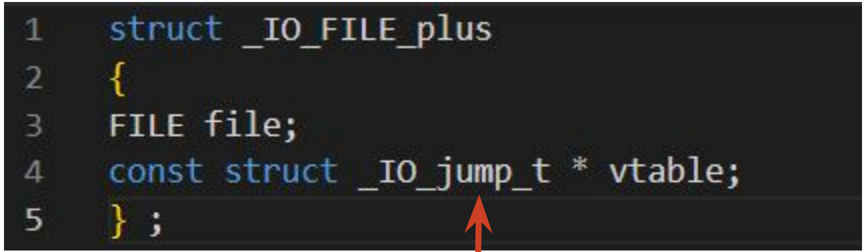
Results

- ➔ Extended DCE to build a customized glibc preserving previous bypasses.
- ➔ Evaluated Linux Kernel Library, and later upgraded kernel stack from 4.4 to 5.10.
- ➔ Improvised Docker based build for DCE, to make it user-friendly and reduce disk usage.

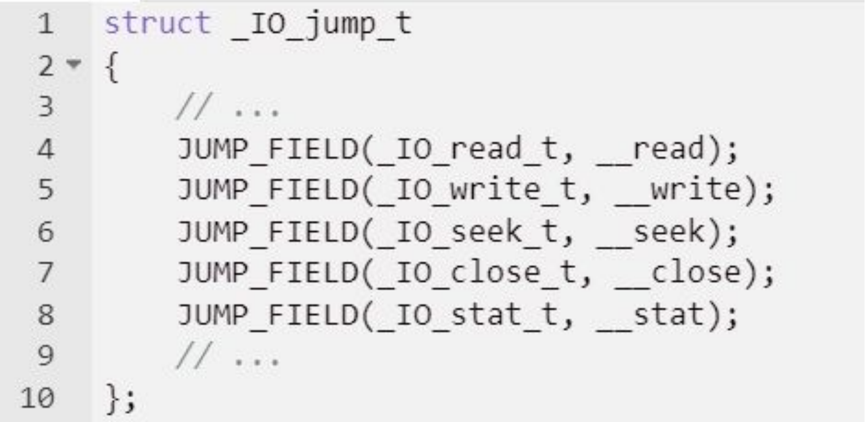
[1] : https://ns-3-dce-linux-upgrade.github.io

Backup

FILE VTable & _IO_FILE_plus



```
struct _IO_FILE_plus
{
FILE file;
const struct _IO_jump_t * vtable;
} ;
```



```
struct _IO_jump_t
{
    // ...
    JUMP_FIELD(_IO_read_t, __read);
    JUMP_FIELD(_IO_write_t, __write);
    JUMP_FIELD(_IO_seek_t, __seek);
    JUMP_FIELD(_IO_close_t, __close);
    JUMP_FIELD(_IO_stat_t, __stat);
    // ...
};
```

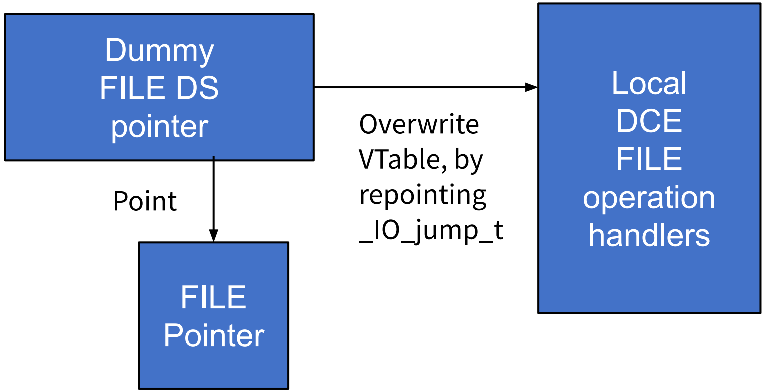
Stores callback references to handlers for operations like : read, close, seek, write & stat

1. LibIO VTable Redirection Limitation

Solution : Patched Glibc and new build SYSROOT

DCE Hijacking & LibIO VTable Protection

<= libc-2.28



Dummy FILE DS pointer

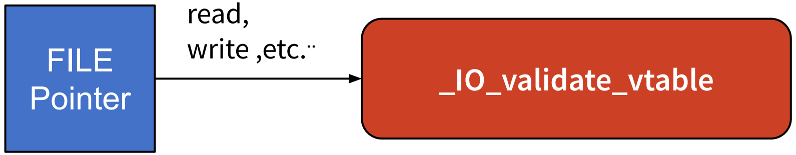
Overwrite VTable, by repointing _IO_jump_t

Local DCE FILE operation handlers

Point

FILE Pointer

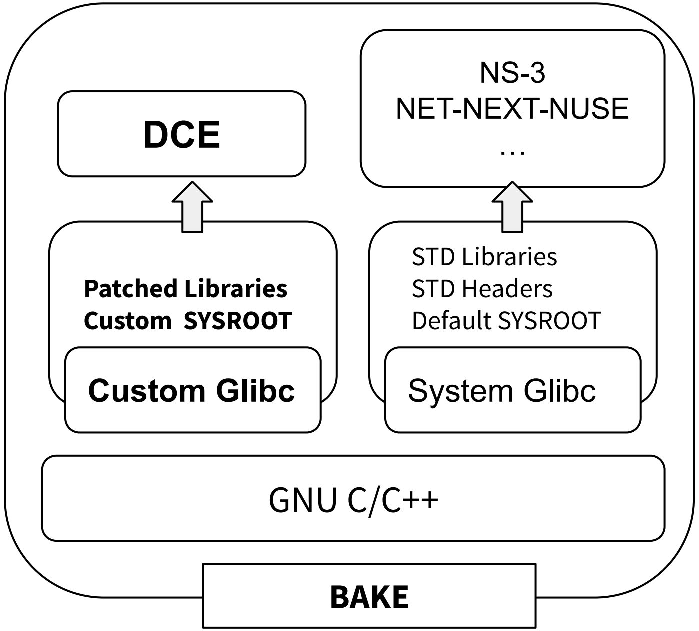
> libc-2.28



FILE Pointer

read, write ,etc.¨

_IO_validate_vtable



DCE

NS-3 NET-NEXT-NUSE …

Patched Libraries
Custom SYSROOT

Custom Glibc

STD Libraries
STD Headers
Default SYSROOT

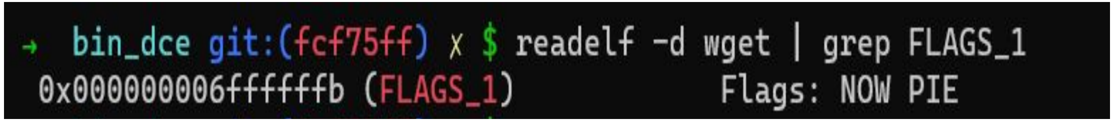
System Glibc

GNU C/C++

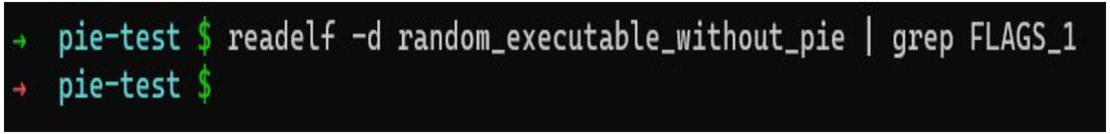
BAKE

PIE Loading in DCE

Compilation/Link Flags : -fPIC -pie

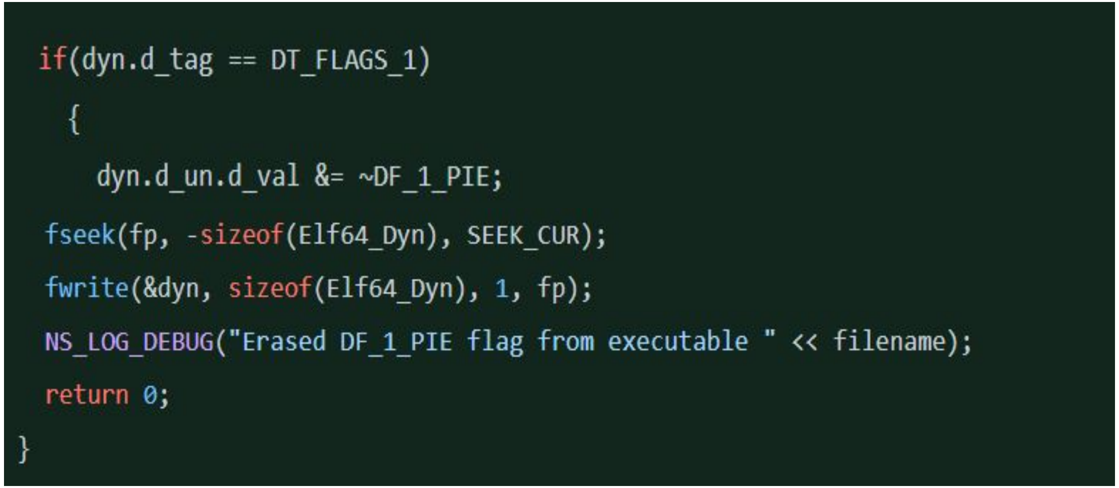


```
→  bin_dce git:(fcf75ff) ✗ $ readelf -d wget | grep FLAGS_1
 0x000000006ffffffb (FLAGS_1)            Flags: NOW PIE
```



```
→  pie-test $ readelf -d random_executable_without_pie | grep FLAGS_1
→  pie-test $
```

DF_1_PIE removal from ELF Header



```
  if(dyn.d_tag == DT_FLAGS_1)
    {
      dyn.d_un.d_val &= ~DF_1_PIE;
   fseek(fp, -sizeof(Elf64_Dyn), SEEK_CUR);
   fwrite(&dyn, sizeof(Elf64_Dyn), 1, fp);
   NS_LOG_DEBUG("Erased DF_1_PIE flag from executable " << filename);
   return 0;
}
```

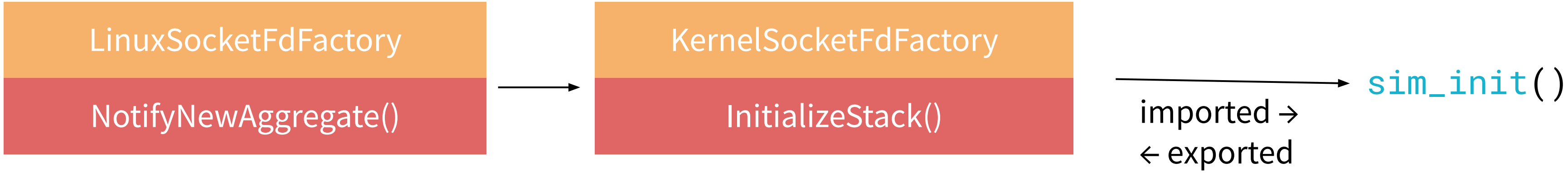
Dropped DF_1_PIE FLAG from Dynamic Table

<u>Alternative</u> : Patch glibc to avoid dynamic section flag validation

Linux Networking Stack - DCE

```
DceManagerHelper dceManager;
dceManager.SetNetworkStack ("ns3::LinuxSocketFdFactory",
                            "Library", StringValue ("liblinux.so"));
```

Usage in DCE Scripts



LinuxSocketFdFactory

NotifyNewAggregate()

KernelSocketFdFactory

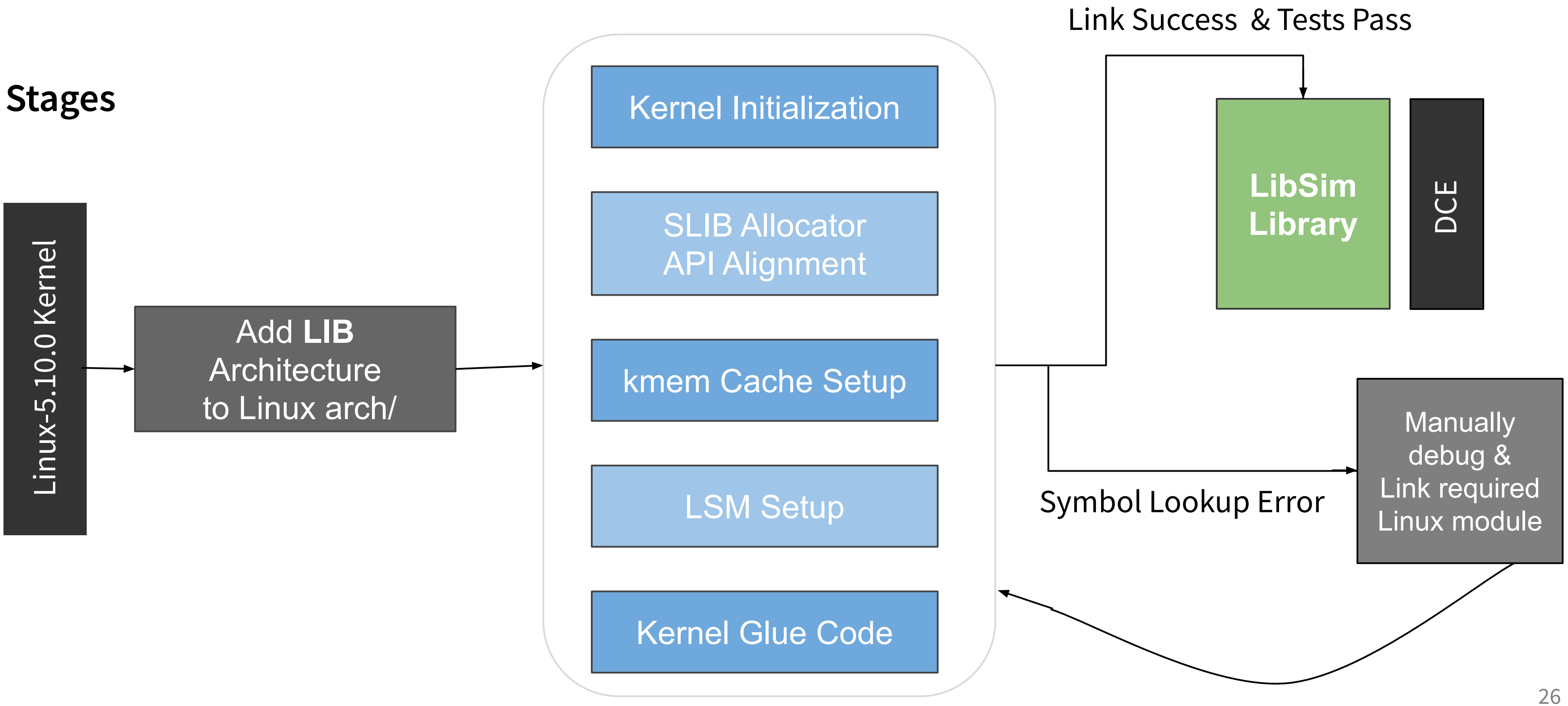
InitializeStack()

imported →

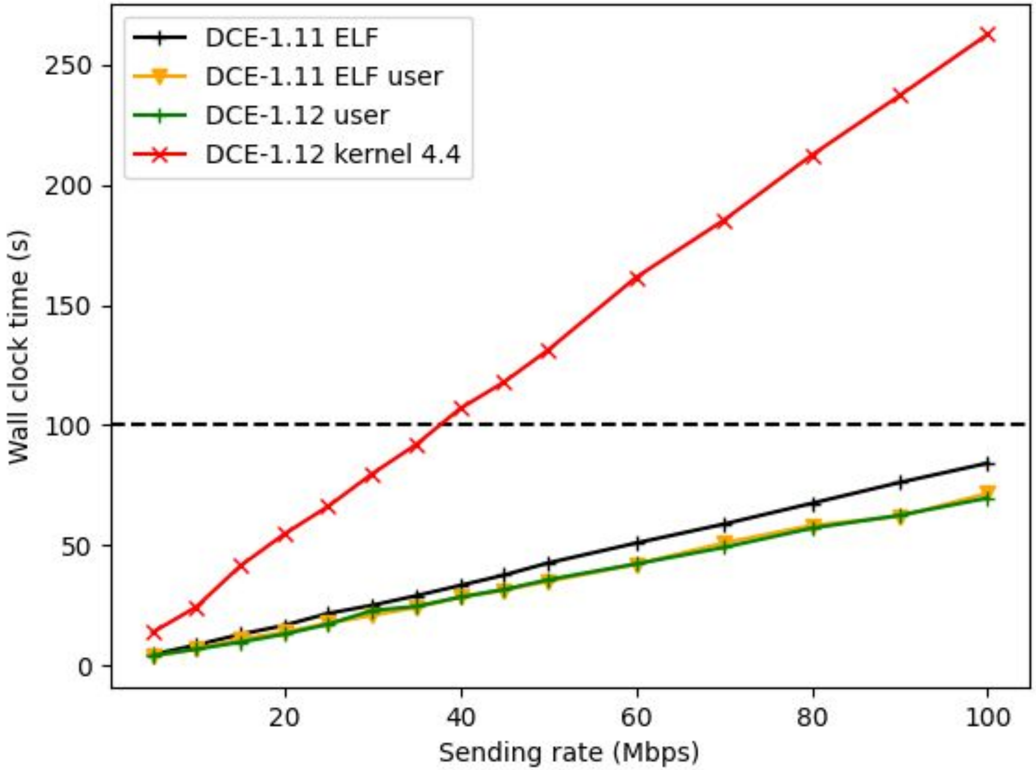
← exported

`sim_init()`

Net-Next-Nuse-5.10.0


Stages
Linux-5.10.0 Kernel
Add **LIB** Architecture to Linux arch/
Kernel Initialization
SLIB Allocator API Alignment
kmem Cache Setup
LSM Setup
Kernel Glue Code
Link Success & Tests Pass
LibSim Library
DCE
Symbol Lookup Error
Manually debug & Link required Linux module

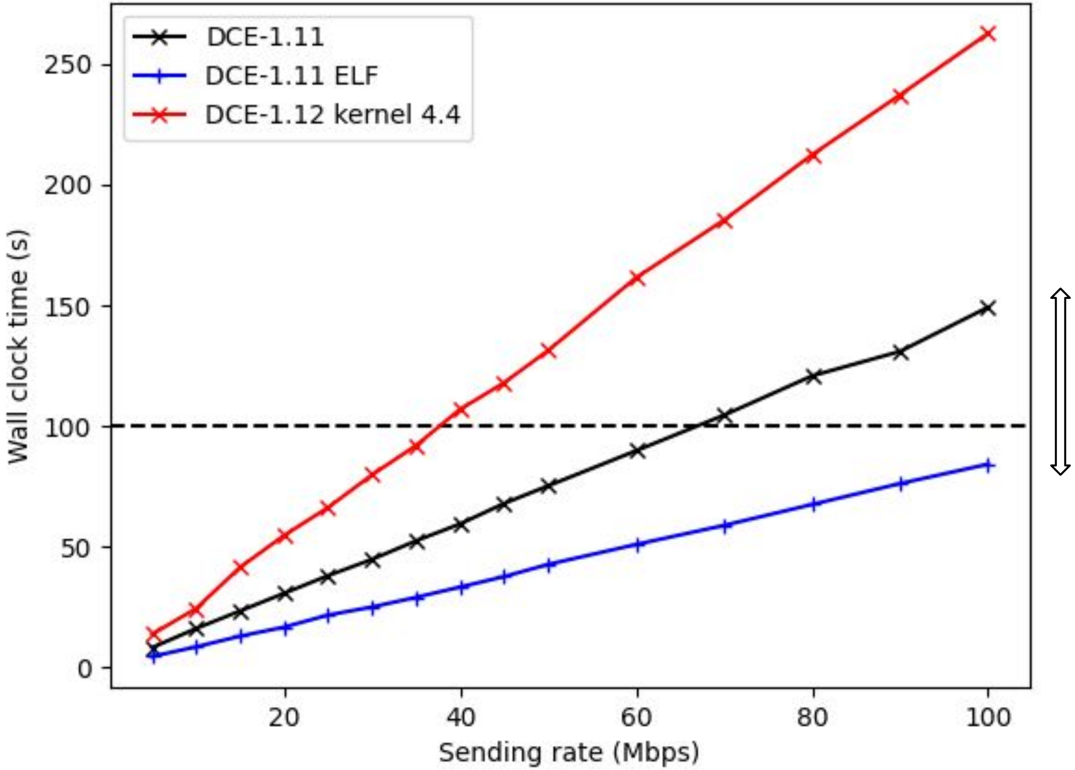
Performance Evaluation : Patched Glibc Vs Hijacked libc



- ➔ Curves for patched glibc (dce-1.12 user) and native libc vtable hijacking (dce-1.11 ELF user), **almost coincide** for ns-3’s network stack.

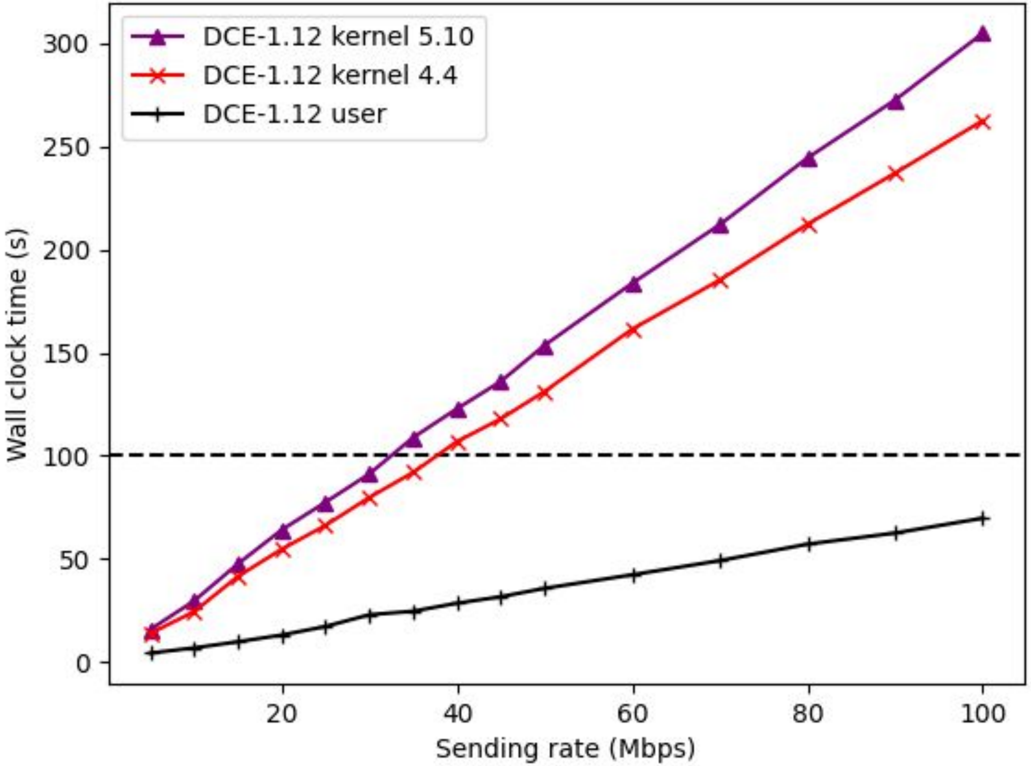
- ➔ Patched glibc runs for Linux network stack notice **high runtimes** when Elf-Loader is not used (dce-1.12 kernel 4.4).

Performance Evaluation : Elf-Loader Impact



Without Elf-Loader, even DCE-1.11 faced performance degradation

Performance Evaluation : Linux-5.10.0 Vs Linux-4.4.0

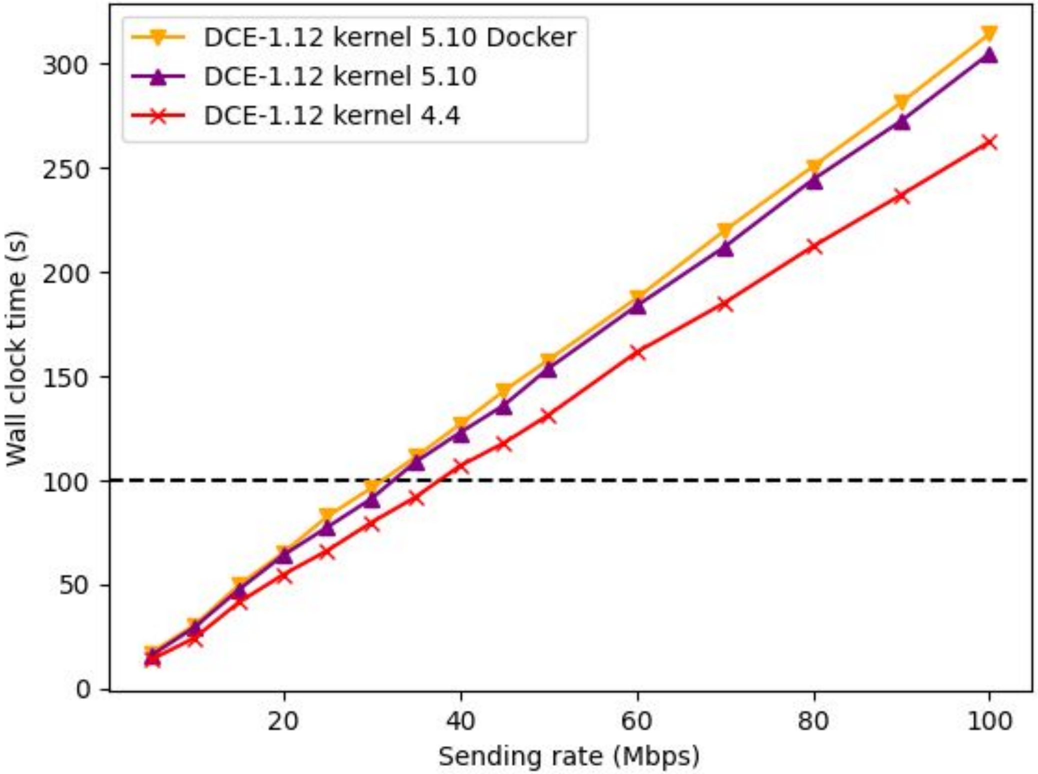


➔ net-next-nuse-5.10.0 plots were seen to have slightly higher runtime, but was **very close.**

➔ Increased number of **context switch** needs for later kernel releases **can** be a reason

➔ Demands further optimization

Performance Evaluation : Linux-5.10.0 on Docker



- ➔ net-next-nuse-5.10.0 plots on native and Docker almost **coincide.**
- ➔ Similar behaviour was noticed for ns-3 network stack runs.
- ➔ Similar performance, with ease of use