Investigating TCP/MPTCP Support for Drop Computing in User Space Network Stacks

Cosmin Stoica, Radu-Ioan Ciobanu, **Ciprian Dobre**

ciprian.dobre@upb.ro

Problem

- During the pandemic, activities such as remote working, school from home, home cinema, sports at home, etc., have become normal for many people
- However, the Internet was under a massive strain
- The cloud model needed to be kept alive and usable under acceptable latency, throughput, costs, etc.
- Key video streaming services like Netflix or YouTube were forced to reduce the quality of their video streaming

Potential solution

- Fully engage the local resources of smart devices using a paradigm such as **Drop Computing**
- Move towards a decentralized computing model with applicability in collaborative computation and data storage services
- Use crowd-based edge/mobile clouds composed of mobile/wearable devices
- Communication is possible by using a TCP/IP stack customised for the Drop Computing devices

Potential solution (2)

- Drop Computing devices do not have all the standard TCP/IP stack functionalities implemented in the kernel
- Their operating systems allow the export of some kernel functionalities into user space as libraries
- Examples: Linux Kernel Library (LKL), Library Operating System (LibOS), UserModeLinux (UML), etc.
- Drop Computing works best with multipath TCP

Potential solution (2)

- Drop Computing devices do not have all the standard TCP/IP stack functionalities implemented in the kernel
- Their operating systems allow the export of some kernel functionalities into user space as libraries
- Examples: Linux Kernel Library (LKL), Library Operating System (LibOS), UserModeLinux (UML), etc.
- Drop Computing works best with multipath TCP

Goals

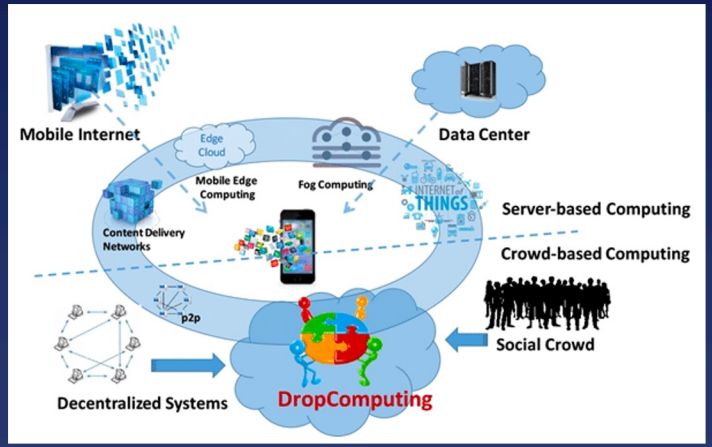
- Evaluate the native Linux network stack and its user space Linux network stacks in the context of Drop Computing applications that use services over TCP
- Analyze the overall behaviour and benefits of using MPTCP in conjunction with the Drop Computing paradigm

LKL and UML

- Linux Kernel Library (LKL)
  - an export of the Linux kernel code as a library
  - provides a simple and maintainable way for applications that run in environments other than Linux to reuse the Linux kernel code
  - the LKL network communication with the host and with the external network is possible because *virtio-net* devices are used
- User Mode Linux (UML)
  - a user space virtual machine that simulates the hardware based on the services exported by the host kernel
  - is able to run almost all host applications and services without modifications, because the user space run is the same as the one run by the native kernel
  - the user mode kernel is a full Linux kernel without the hardware-specific drivers

The Drop Computing vision

- Drop Computing proposes creating dynamic ad hoc micro-clouds of small devices that can collaborate with each other locally, before going towards the edge/fog nodes or to the cloud
- These low-level devices can communicate with each other using mainly close-range protocols (Bluetooth, Wi-Fi Direct) and help each other by delivering data (“I have something that you need so I can give it to you”) or performing offloaded computations (“I can help you process this”)

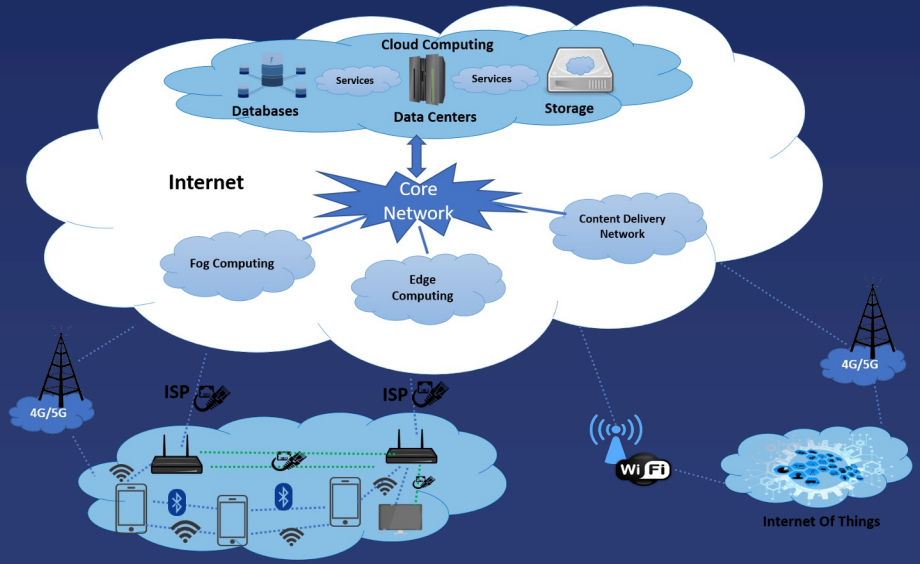


Adding MPTCP to Drop Computing

- MPTCP is a key technology in Drop Computing, because of all the heterogeneous nodes with different computation resources and multiple networking interfaces
- In a Drop Computing ad-hoc network, a hybrid solution has to be in place, which:
  - uses the native kernel network stack for the nodes located at the network border (gateways, smart set-top boxes, etc.)
  - uses user space networking stacks (like LKL/UML) or virtualisation technologies for the other nodes

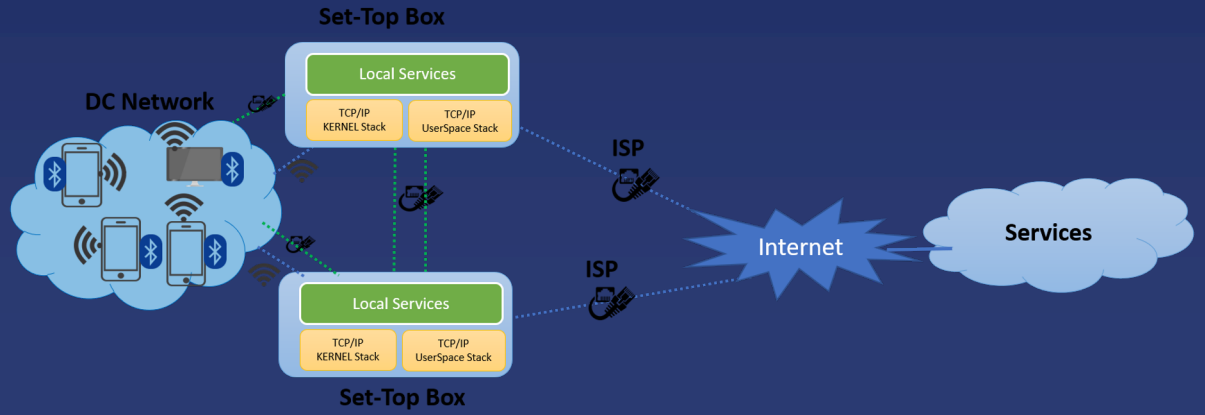
Adding MPTCP to Drop Computing (2)

- The figure shows a networking system model which can be engaged in Drop Computing taking into consideration current networking nodes capabilities



Adding MPTCP to Drop Computing (3)

- The figure shows how two set-top boxes running Linux can concurrently use the wired LAN connections between them, the wired connection with the Internet Provider to connect to the services provided by the servers from the cloud, and Wi-Fi for the connection with smart devices like TVs, smartphones, smart robot vacuums, etc



Experimental setup

- Hardware:
  - computing nodes configured as set-top boxes
  - based on PC Engines APU2D34 motherboards
  - 1 GHz 64bit Dual Core AMD Geode processor
  - 16 GB SSD
  - 4 GB RAM
  - three mini-PCIe slots used for Wi-Fi card/3G/4G modem, etc.
  - 3 1GB NICs (one interface connected to the Internet)

Experimental setup (2)

- Software:
  - Debian distribution with a Linux Kernel 5.4 with MPTCP enabled
  - enabled MPTCP with all path managers and congestion control algorithms activated
  - ported the LKL library to the 5.4 MPTCP Linux kernel
  - activated and configured UML (already supported by the Linux kernel utilised)
  - used *iperf* for data traffic generation and *curl* for file download

Experimental setup (3)

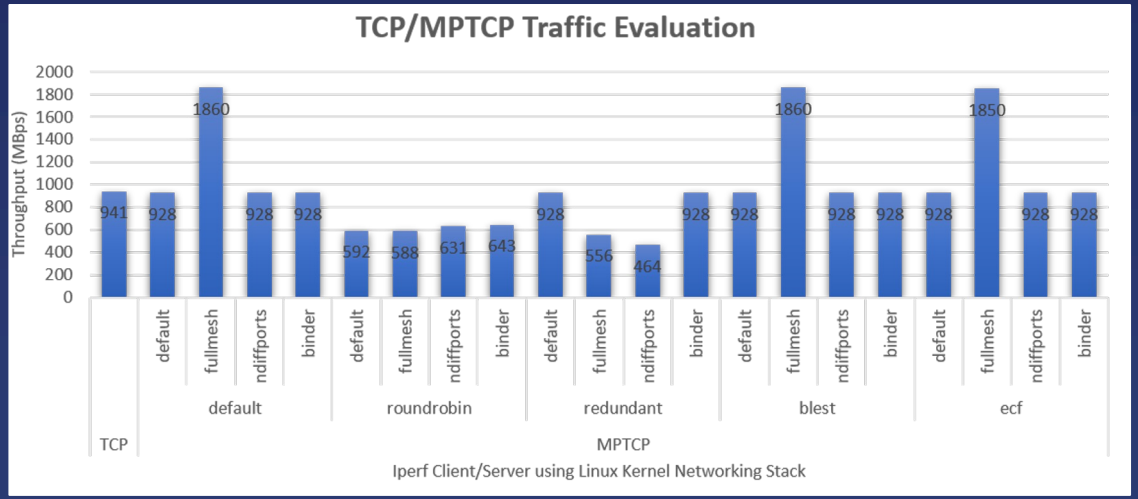
- Scenario 1
  - analyses the network traffic between two gateways over the two 1GBps Ethernet Interfaces
  - acting as border routers, the two devices have a dual role which allows local nodes to connect to the Internet or to interconnect between themselves if the local nodes can not connect directly
  - TCP traffic generated using *iperf*
  - check network throughput using TCP and MPTCP over the 2 Ethernet interfaces running *iperf* directly in Linux with the default Linux network stack
  - then, check how the network throughput is sustained by the user space network stacks from LKL and UML for TCP and MPTCP

Experimental setup (3)

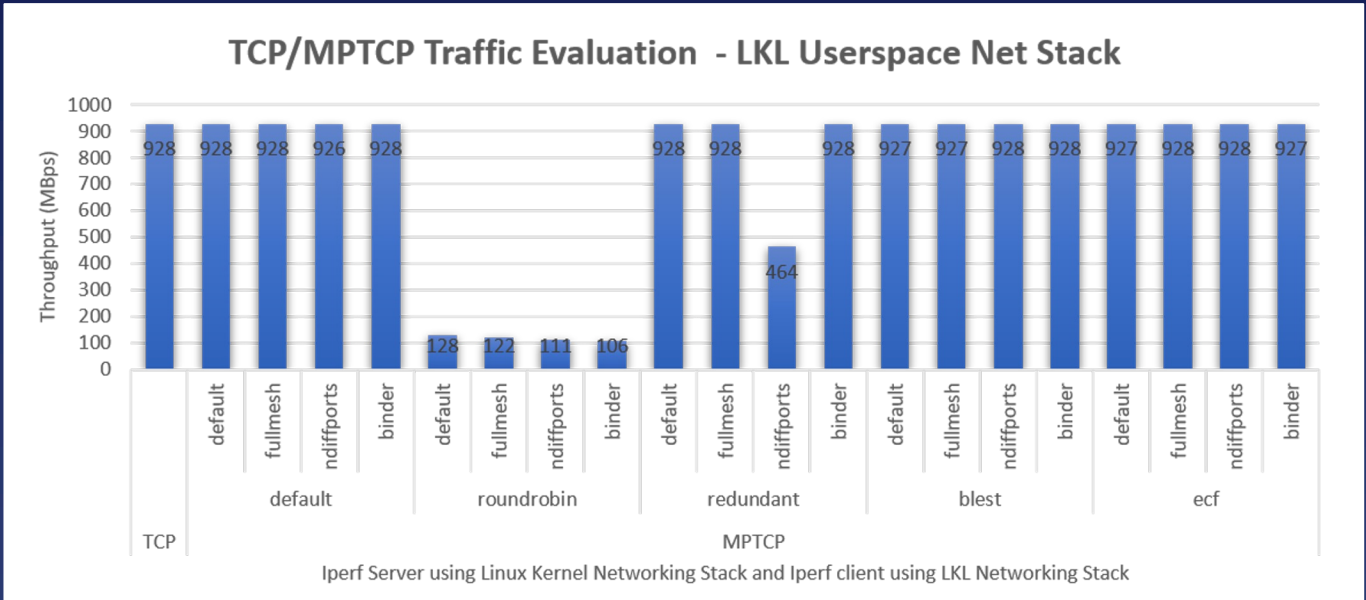
- Scenario 2
  - tests the behaviour of MPTCP when one gateway node plays the role of a local cache for different services like file storage, video streaming, audio streaming, etc., providing the requested data if it is available locally
  - used the network configuration from the first scenario and configured one gateway node to act as a web server using the Python SimpleHTTP module, and the other device as a client which can download files using *curl*

Results

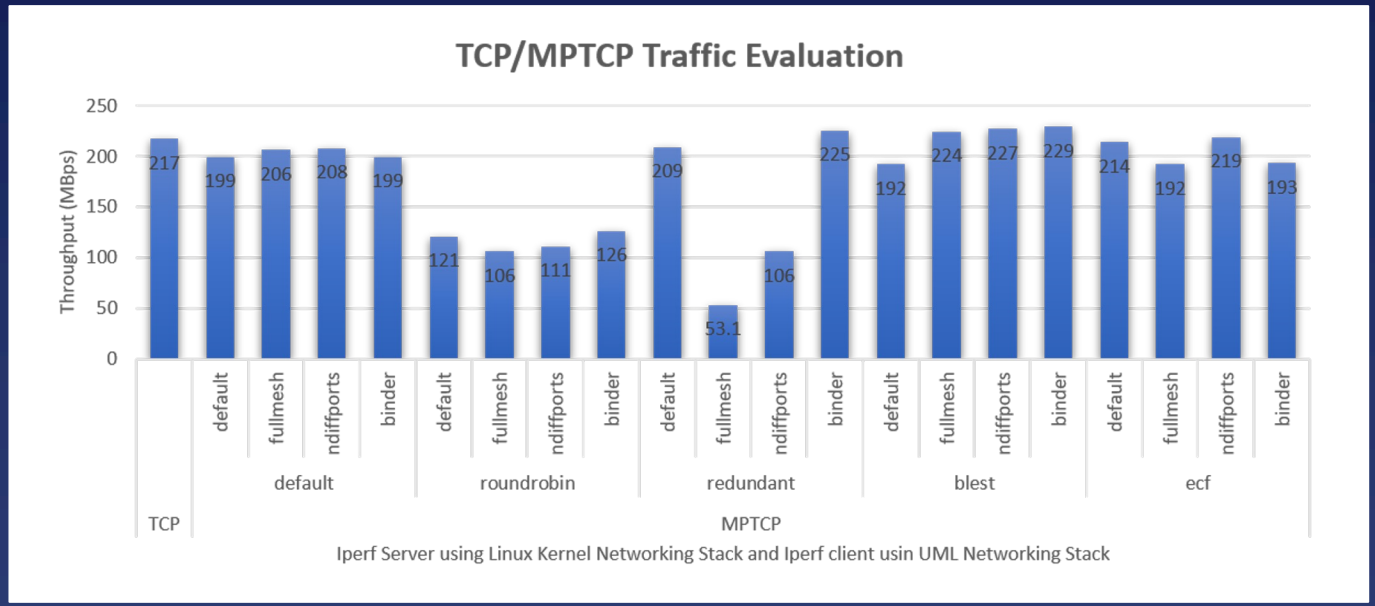
- The experiments performed in the first scenario focus on how the throughput is influenced by the MPTCP scheduler and path manager
- Additionally, we are analysing the impact of the network stack used (Linux Native, LKL, UML) on TCP/MPTCP traffic



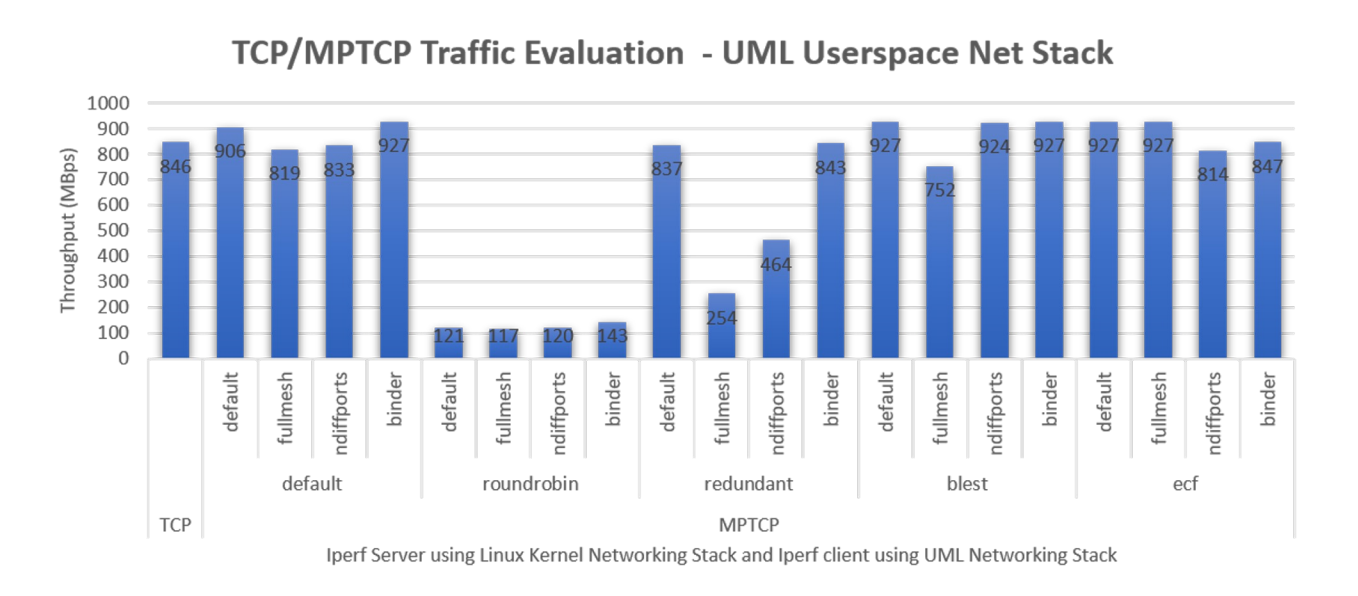
Results (2)


TCP/MPTCP Traffic Evaluation - LKL Userspace Net Stack
Throughput (MBps)
1000
900
800
700
600
500
400
300
200
100
0
928
928
928
926
928
128
122
111
106
928
928
464
928
927
927
928
928
927
928
928
927
default
fullmesh
ndiffports
binder
default
fullmesh
ndiffports
binder
default
fullmesh
ndiffports
binder
default
fullmesh
ndiffports
binder
default
fullmesh
ndiffports
binder
default
roundrobin
redundant
blest
ecf
TCP
MPTCP
Iperf Server using Linux Kernel Networking Stack and Iperf client using LKL Networking Stack

Results (3)

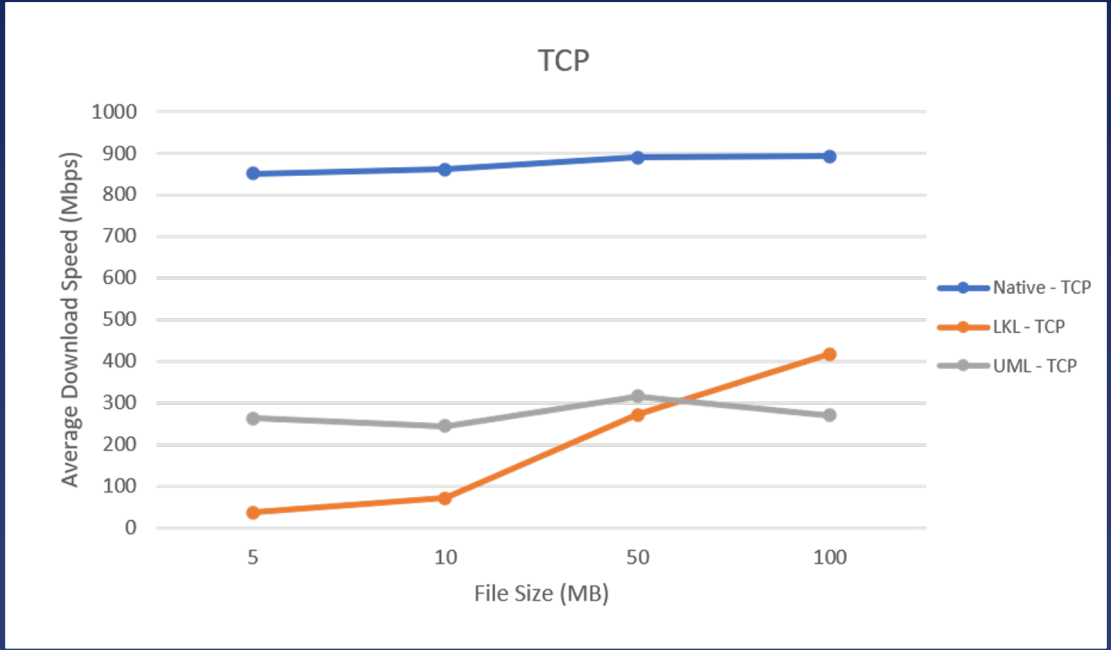

TCP/MPTCP Traffic Evaluation
Throughput (MBps)
250
200
150
100
50
0
217
199
206
208
199
121
106
111
126
209
53.1
106
225
192
224
227
229
214
192
219
193
default
fullmesh
ndiffports
binder
default
roundrobin
redundant
blest
ecf
TCP
MPTCP
Iperf Server using Linux Kernel Networking Stack and Iperf client usin UML Networking Stack

Results (4)

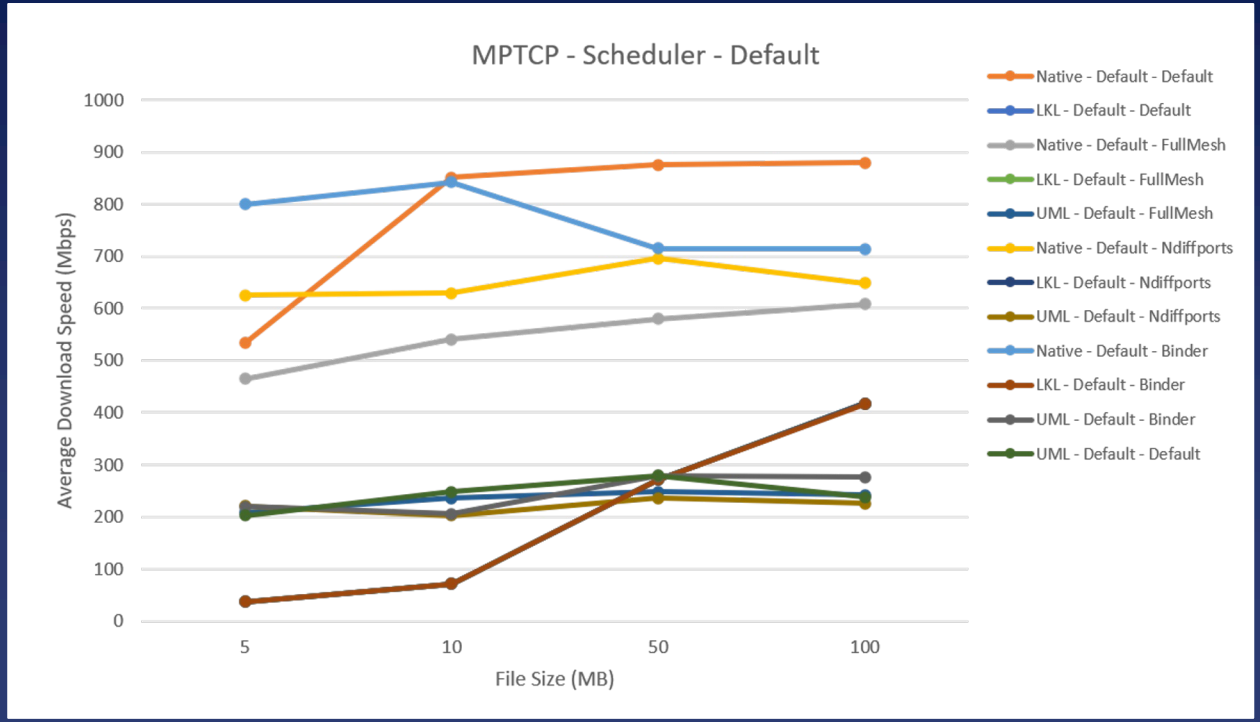

TCP/MPTCP Traffic Evaluation - UML Userspace Net Stack
Throughput (MBps)
1000
900
800
700
600
500
400
300
200
100
0
846
906
819
833
927
121
117
120
143
837
254
464
843
927
752
924
927
927
927
814
847
default
fullmesh
ndiffports
binder
default
fullmesh
ndiffports
binder
default
fullmesh
ndiffports
binder
default
fullmesh
ndiffports
binder
default
fullmesh
ndiffports
binder
default
roundrobin
redundant
blest
ecf
TCP
MPTCP
Iperf Server using Linux Kernel Networking Stack and Iperf client using UML Networking Stack

Results (5)

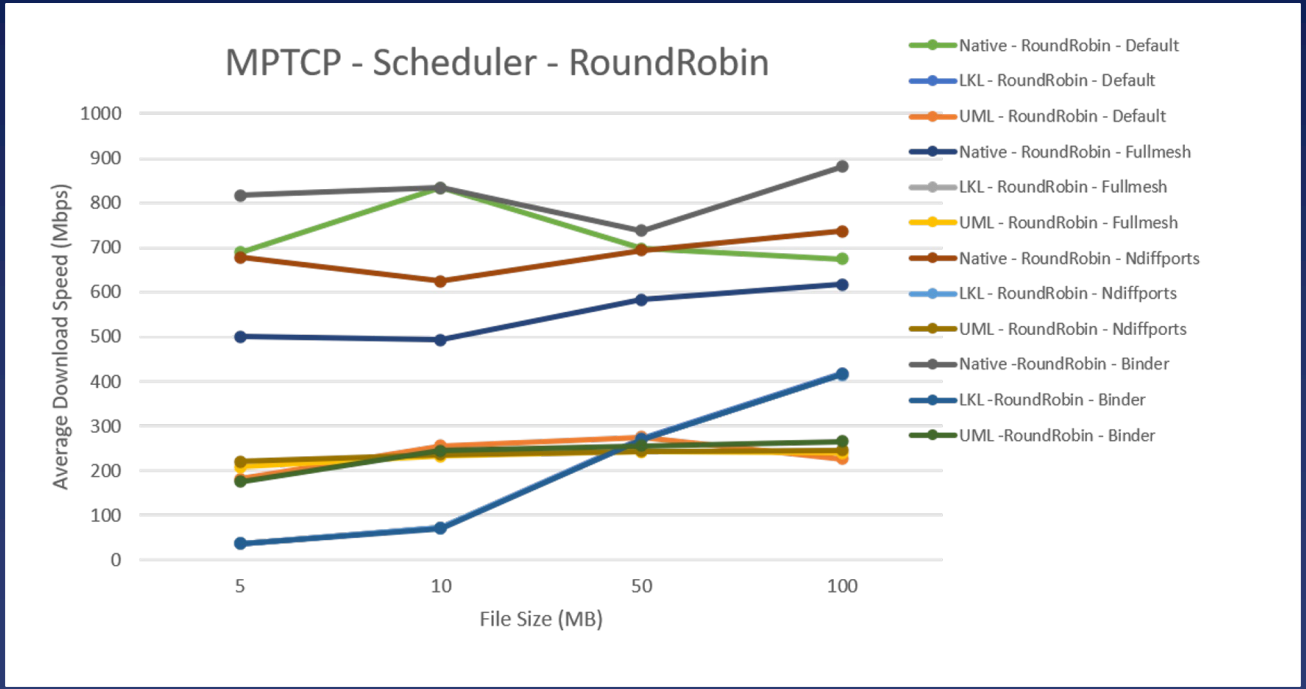
- The second scenario shows that MPTCP can be successfully used in a web client-server architecture, a solution which is widely valued in cloud services



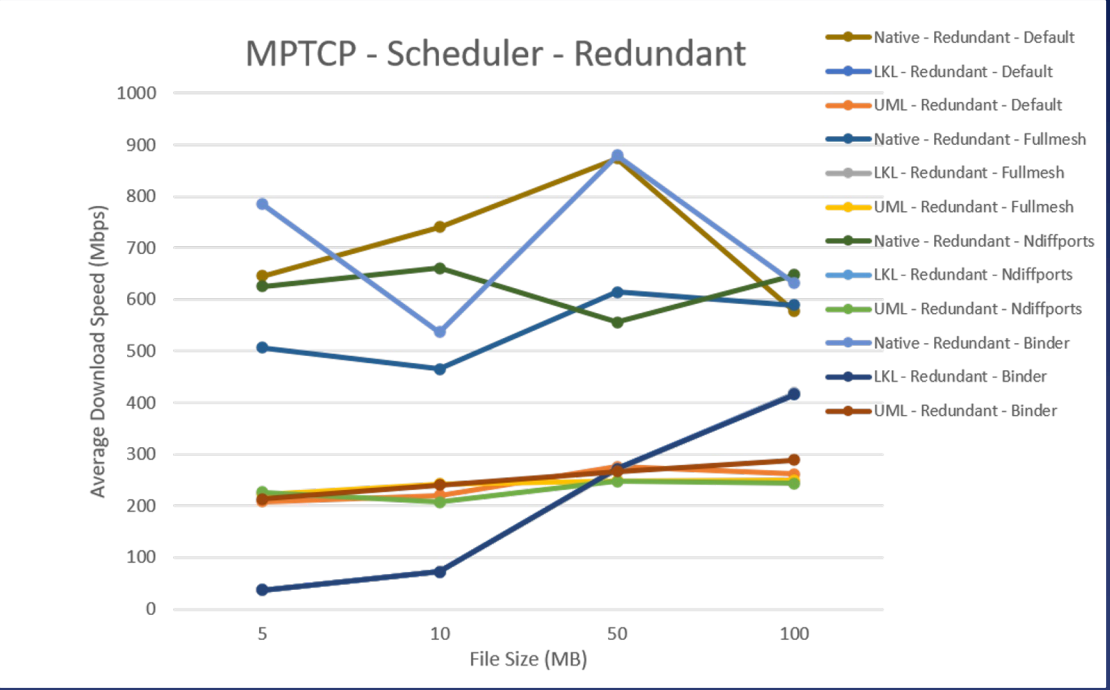
Results (6)


MPTCP - Scheduler - Default
Average Download Speed (Mbps)
1000
900
800
700
600
500
400
300
200
100
0
5
10
50
100
File Size (MB)
Native - Default - Default
LKL - Default - Default
Native - Default - FullMesh
LKL - Default - FullMesh
UML - Default - FullMesh
Native - Default - Ndiffports
LKL - Default - Ndiffports
UML - Default - Ndiffports
Native - Default - Binder
LKL - Default - Binder
UML - Default - Binder
UML - Default - Default

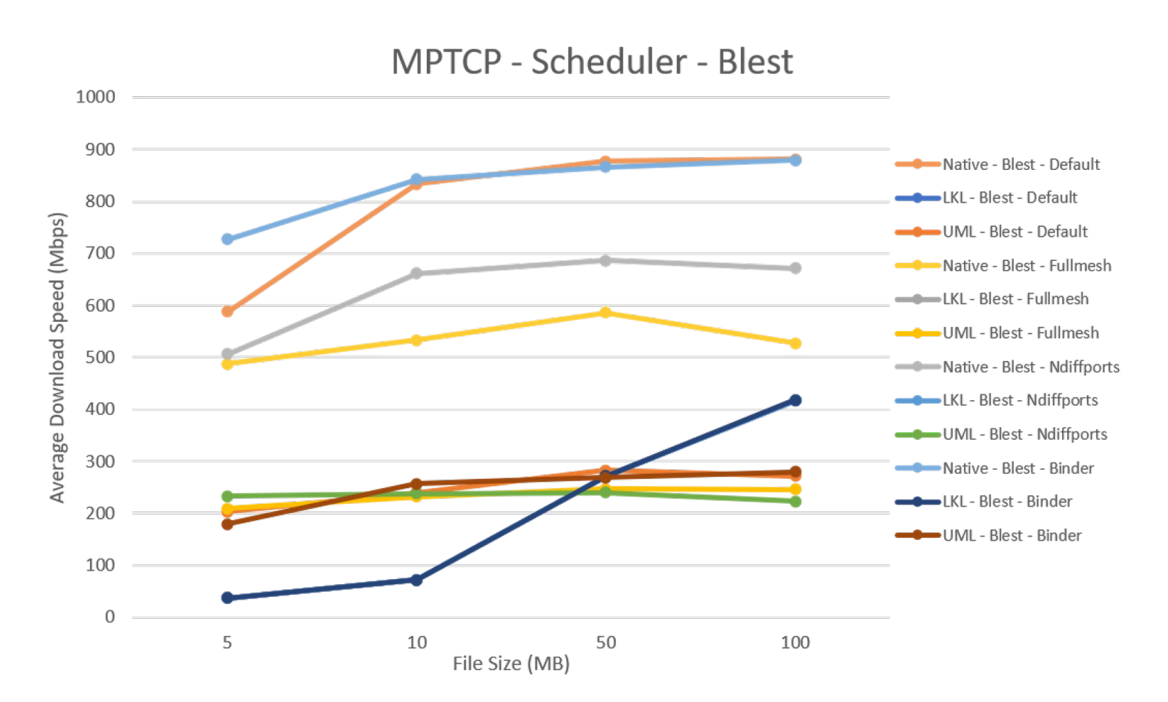
Results (7)


MPTCP - Scheduler - RoundRobin
Average Download Speed (Mbps)
1000
900
800
700
600
500
400
300
200
100
0
5
10
50
100
File Size (MB)
Native - RoundRobin - Default
LKL - RoundRobin - Default
UML - RoundRobin - Default
Native - RoundRobin - Fullmesh
LKL - RoundRobin - Fullmesh
UML - RoundRobin - Fullmesh
Native - RoundRobin - Ndiffports
LKL - RoundRobin - Ndiffports
UML - RoundRobin - Ndiffports
Native -RoundRobin - Binder
LKL -RoundRobin - Binder
UML -RoundRobin - Binder

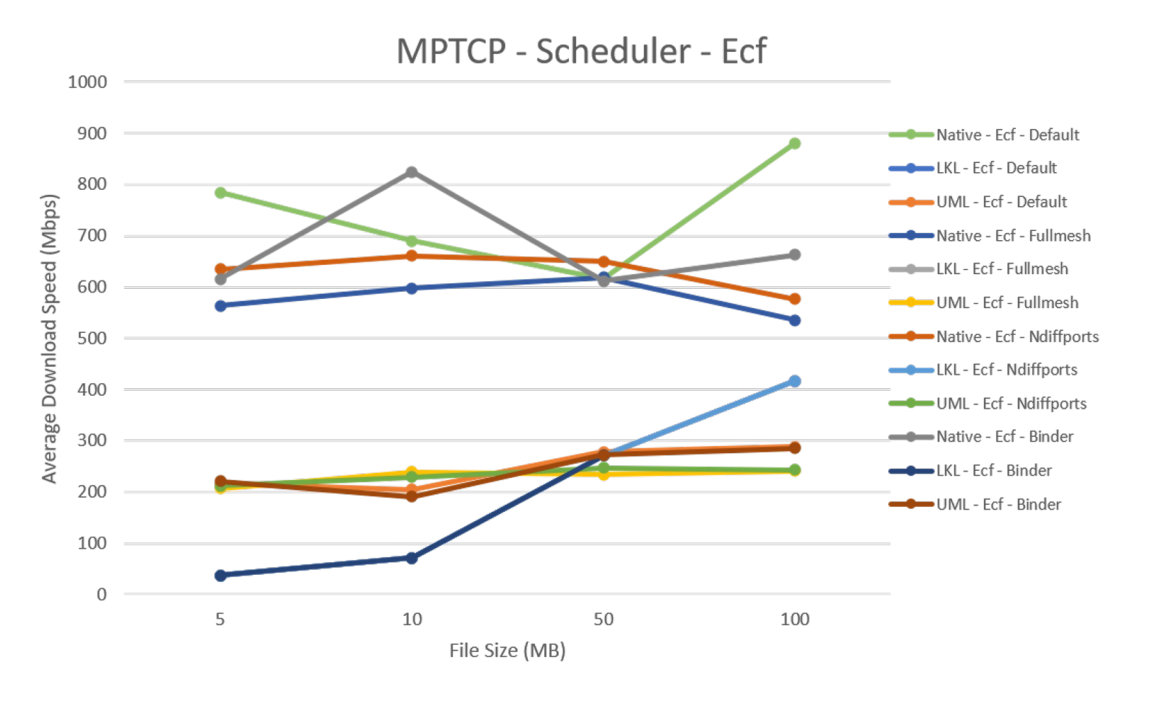
Results (8)


MPTCP - Scheduler - Redundant
Average Download Speed (Mbps)
1000
900
800
700
600
500
400
300
200
100
0
5
10
50
100
File Size (MB)
Native - Redundant - Default
LKL - Redundant - Default
UML - Redundant - Default
Native - Redundant - Fullmesh
LKL - Redundant - Fullmesh
UML - Redundant - Fullmesh
Native - Redundant - Ndiffports
LKL - Redundant - Ndiffports
UML - Redundant - Ndiffports
Native - Redundant - Binder
LKL - Redundant - Binder
UML - Redundant - Binder

Results (9)


MPTCP - Scheduler - Blest
Average Download Speed (Mbps)
1000
900
800
700
600
500
400
300
200
100
0
5
10
50
100
File Size (MB)
Native - Blest - Default
LKL - Blest - Default
UML - Blest - Default
Native - Blest - Fullmesh
LKL - Blest - Fullmesh
UML - Blest - Fullmesh
Native - Blest - Ndiffports
LKL - Blest - Ndiffports
UML - Blest - Ndiffports
Native - Blest - Binder
LKL - Blest - Binder
UML - Blest - Binder

Results (10)


MPTCP - Scheduler - Ecf
Average Download Speed (Mbps)
1000
900
800
700
600
500
400
300
200
100
0
5
10
50
100
File Size (MB)
Native - Ecf - Default
LKL - Ecf - Default
UML - Ecf - Default
Native - Ecf - Fullmesh
LKL - Ecf - Fullmesh
UML - Ecf - Fullmesh
Native - Ecf - Ndiffports
LKL - Ecf - Ndiffports
UML - Ecf - Ndiffports
Native - Ecf - Binder
LKL - Ecf - Binder
UML - Ecf - Binder

Conclusions and future work

- Conclusions
  - the evaluation of MPTCP behaviour over the Linux native network stack, LKL and UML network stacks performed in this paper proves that MPTCP can be used successfully over any of them
  - the most suitable one should be selected based on the hardware profile of the device used and the software application of the implemented use case
  - we also showed that LKL and UML can be enabled on low-end devices in order to allow them to use all their network interfaces and have a better failure handover solution
- Future work
  - evaluate the new version of the Unified Linux Kernel Library
  - start developing MPTCP path planner and scheduler algorithms in order to take into consideration the resources capabilities of Drop Computing network nodes

Thank you!

Questions?