Network stack personality in Android phone

Cristina Opriceana

Polytechnic University of Bucharest

Hajime Tazaki

IIJ Research Laboratory

1. Motivation

The network stack personality with an alternate network stack is a way to introduce protocol extensions without interventions of host systems. Various studies [1][9][8] have attempted to understand how it is useful with detailed performance analysis. This paper explores further with another target environment, a smartphone.

Smartphone market has been grown and now the most dominant operating system in the world is Android, which is based on Linux kernel. While it keeps providing important and critical features for mobile users, the system faces a difficult situation in case of software updates. It takes a long time to deliver an update of the various part of software components, even the update includes a critical security issue.

Why is it so difficult? Because a software update involves various players such as Android core developers, silicon manufactures, device makers, and mobile carriers, which makes immediate delivery of a software update hard. Google announced a new framework, called project Treble [2], to alleviate this issue, but introducing new network stack extension to an Android device is still hard.

There are several options which we can consider for alternatives: virtual guest operating system on top of a hypervisor, user-mode linux (UML [3]), or container-based guest OS. But none of them meets various requirements of an application under Android platform. Others [4] also tried to deploy a middlebox which is capable of a network extension, but this deployment manner heavily relies on an infrastructure-side upgrade, which is sometime much harder than an endpoint upgrade.

In the rest of sections, we describe how we reached to the network stack personality on Android phone in order to introduce a protocol extension to a particular application. Especially we used Multipath-TCP (mptcp) [5] as an example of protocol extension, which is still an out-of-tree extension to the Linux mainline kernel.

2. Extention to LKL

Our contribution is to extend Linux Kernel Library (LKL) [6] to run an application with it on Android platform. There are several other people who also contributed to this development[1], especially most of ARM Android cross-build toolchain support are from them. Our development focuses on an integration to LKL with additional features which require to use network devices on Android devices.

IPv4 encapsulation over PF_PACKET socket: Our additional implementation is to support proper pf_packet backend over a celllar interface of Android device. A cellular interface encapsulates pakcets with IP (v4/v6) so that point-to-point communication with the gateway does not require additional Layer-2 level negotiation (e.g., ARP). On the other hand, the original implementation of pf_packet backend in LKL encapsulate packets as if the underlying network interface expects Ethernet encapsulation, which does not meet the condition with this case. Thus, we extend this pf_packet backend with an option to encapsulate with IP header. Note that the IPv6 encapsulation, which nowadays is provided by various service providers, is not implemented yet at the time of writing.

Permission issues: At the beginning of design, we would like to have our extension in a transparent way: if we could install our module over the application store, all of users benefit without a pain of system configuration. But unfortunately our extension requires root privilege, more specifically a `CAP_NET_RAW` permission, in order to open a packet socket inside an application.

With the above issue resolved, an application on Andriod device has ability to run with alternate network stack which will be dynamically loaded at runtime. As usual LKL applications, we use `LD_PRELOAD` to load this library, replace necessary function call symbols in order to redirect such calls into the library (we call this library as a hijack library).

Out-of-tree protocols with LKL: Network stack personality is always powerful with any kind of extensions to a network stack, but an out-of-tree protocol extension, which is still under review, or is far to reach the goal, is most suitable use case since it does not require to replace a host kernel which is often a hard-to-achieve task in some system.

Multipath-TCP (mptcp) is categorized in such extension: it was proposed a couple of years ago and has always difficulties to widely deploy its implementation. Backporting its im-

[1] The detail is described in a pull request on github `https://github.com/lkl/linux/issues/59`, especially huge contributions from @mxi1 and @stfairy.

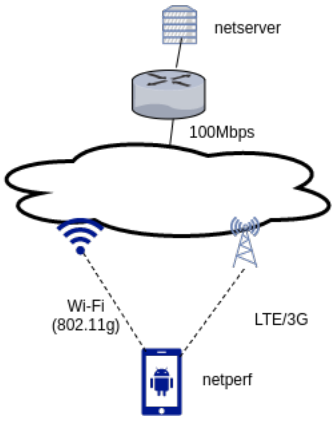

Figure 1: Network configuration of experiment.

plementation to Android kernel has been conducted several times[2], but they are required to be specific devices, which makes this backport hard to apply any Android devices and update with the latest modification to both mptcp and Android kernel.

Backporting LKL to mptcp kernel[3] benefits such a painfulness of maintenance and availability of various devices, and then we can use the out-of-tree network protocol on Android phones.

3. Benchmarks

One might raise natural questions to our implementation: 1) how does the network stack personality show the performance compared to the native kernel ? 2) how does the battery consumption look like ? Our evaluation tries to answer these questions by conduction benchmarks with popular `netperf` tool on a Android phone.

The benchmark was conducted with a single android device, Nexus5, with the Android 6.0.1 stock image installed or with a custom image of mptcp extension included. We used a Linux server, Ubuntu 16.04 (amd64) over QEMU/KVM, with mptcp kernel extension installed (mptcp-4.4.70, v0.92). As illustrated in figure 1, there are two paths between the Android phone and Linux server, which are diverged between a cellular and WiFi network attached to the phone. We collected achieved goodput of ten-second TCP_STREAM in bits/second from five repetitions of measurement as well as CPU utilization reported by `netperf` command, which is based on the values reported at `/proc/stat`.

Single-stream TCP_STREAM At first, we measured the goodput and processor usages with a single path TCP stream in order to understand the base performance. We simply executed netperf sessions with variable size of payload (from 64 to 65507 bytes), and only used a WiFi interface over stock version of Android kernel. Figure 2 is the result of this experiment. While the WiFi condition is quite unstable as usual,

[2] `https://multipath-tcp.org/pmwiki.php/Users/Android`

[3] `https://github.com/multipath-tcp/mptcp`

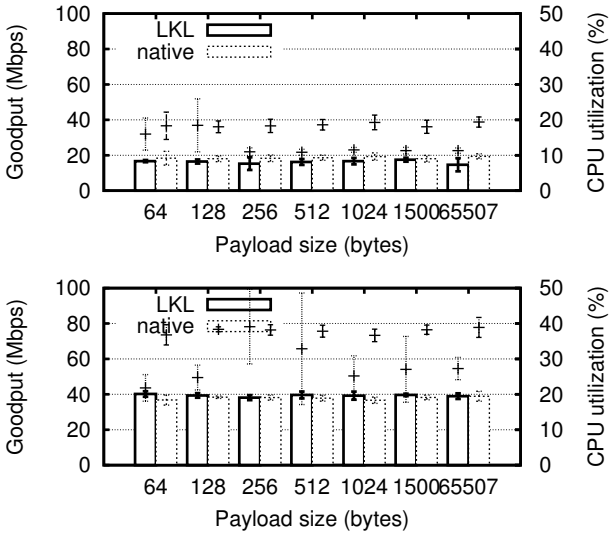

Figure 2: Goodput (bars) and CPU utilization (points) with a single path (WiFi) TCP_STREAM: Tx (top) and Rx (bottom).

the achieved goodput between LKL-ed netperf and host netperf are almost comparable in both directions. On the other hand, the CPU utilization of LKL over the executions is almost less than the one with host kernel, resulting the power consumption is more effective with LKL. We have no experience with the detail profiling with those executions; we plan to do in future investigation.

Dual-stream TCP_STREAM Next measurement uses the same configurations as the previous one but with multiple streams by mptcp. We used a private custom Android image instead of the stock one for the mptcp-enabled kernel, and the LKL as the previous measurement. We also collected the same information based on netperf command outputs. Figure 3 plots the result of this experiment. Unlike the previous experiment with a single stream, the achieved goodputs of Tx path (from netperf to netserver) are different: the goodput of LKL always outperforms the one of native kernel, while the goodput of Rx path is opposite (LKL $\leq$ native). One of the reason of this difference for the Tx path is the used congestion control algorithms: LKL uses cubic while native kernel uses LIA [7]. Further investigations are required but it will be future work.

We also observed that the CPU utilization with LKL is larger than LKL unlike the previous experiment, even if the achieved goodput of LKL is less than the native kernel. Although we also need to profile in detail, a possible cause of this higher CPU usage with LKL is likely attributed to the raw socket implementation with IPv4 encapsulation. With this mode over the cellular interface, every incoming packet goes to both the host kernel and the LKL network stack because those two stacks use the same IP address[4]. We needed to configure `iptables` entry to drop the specific TCP reset packets from the host kernel so that the LKL stack can

[4] Otherwise, the gateway at cellular network does forward any packets.

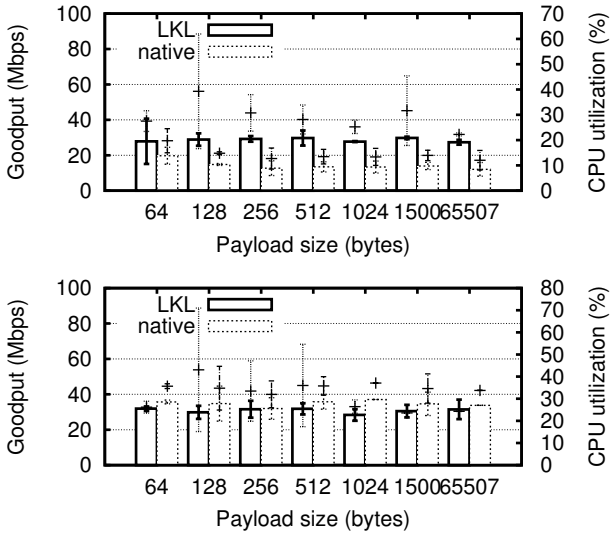

Figure 3: Goodput (bars) and CPU utilization (points) with multiple paths (WiFi+LTE) TCP_STREAM: Tx (top) and Rx (bottom).

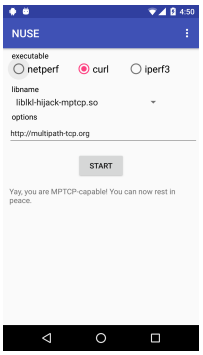

Figure 4: A crafted application for Android to use alternate network stacks.

handle TCP sessions, but this limitation consumes more processing resource while the native kernel does not have such issue. To alleviate this issue, we may want to use other network backend for LKL, such as `macvtap` devices, so that this limitation is to be eliminated.

4. Further Use Cases

Although current implementation only allows us to use an alternate network stack at the condition of available permission, the deployment obstacle which requires to re-install a custom image (with a kernel replacement) is eliminated by installing our library (i.e., LKL). Figure 4 is a screenshot of a crafted Android application which executes a couple of networking utilties (netperf, iperf, curl) with different network stack, specified by a library (libname in the application).

Another example is to expand the idea into a GUI-based application. Typically the most of traffic is based on http or https nowadays, supporting this feature into a web browser

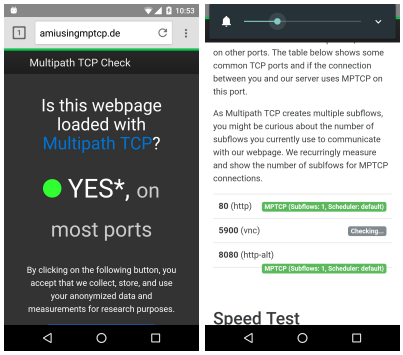

Figure 5: Multipath-TCP enabled web browser as an Android application.

might be of interest. Unlike console-based applications, a GUI based application has a different way to do the `LD_PRELOAD`-like jobs[5]. Figure 5 is another screenshot to demonstrate mptcp capability with the client application, enhanced with mptcp support in LKL, without any host kernel upgrade to introduce the extension.

5. Summary and Further Directions

Network stack personality is useful if you have no control on a system which you have to live with, but we also have to carefully consider that this has still restrictions on transparent usages as we have on our commodity operating system. In this paper, we have investigated if the personality is possible on Android devices and how much this restriction we would have.

Further investigations of observed measurement results are required and we will look for a profiling technique on commodity Android phone in order to see LKL equipped applications have meaningful insight to alleviate the update fragmentation issue of Android platform.

References

1. HK Jerry Chu and Yuan Liu, *User Space TCP-Getting LKL Ready for the Prime Time*, Linux Netdev 1.2 (October 2016).
2. The Android Source Code, *Project Treble*. (Accessed Oct 11th 2017).
3. J. Dike, *User Mode Linux*, Proceedings of the 5th anual linux showcase and conference, 2001, pp. 3–14.
4. SungHoon Seo et al., *KT's MPTCP Proxy Experiences - Deployment and testing considerations*. (Accessed Oct 11th 2017).
5. Doru-Cristian Gucea and Octavian Purdila, *Shaping the Linux kernel MPTCP implementation towards upstream acceptance*, 2015.
6. Octavian Purdila, Lucian Adrian Grijincu, and Nicolae Roedunet International Conference RoEduNet 2010 9th Tapus, *LKL: The Linux kernel library*, Roedunet international conference (roedunet), 2010 9th, 2010, pp. 328–333.

[5] `setprop wrap.$(package_name) LD_PRELOAD=liblkl-hijack-mptcp.so` is the exact command line to specify the library name.

7. C. Raiciu, M. Handley, and D. Wischik, *Coupled Congestion Control for Multipath Transport Protocols*, Request for Comments, IETF, Internet Engineering Task Force, 2011, http://www.ietf.org/rfc/rfc6356.txt.
8. Hajime Tazaki, *Playing BBR with a userspace network stack*, 2017.
9. Hajime Tazaki, Ryo Nakamura, and Yuji Sekiya, *Library operating system with mainline Linux kernel*, 2015.