User-Space Network Tunneling Under a Mobile Platform: A Case Study for Android Environments

Dario Bruneo[1], Salvatore Distefano[1,2], Kostya Esmukov[2], Francesco Longo[1], Giovanni Merlino[1(✉)], and Antonio Puliafito[1]

[1] Dipartimento di Ingegneria, Università degli Studi di Messina, Messina, Italy
{dbruneo,sdistefano,flongo,gmerlino,apuliafito}@unime.it
[2] Social and Urban Computing Group, Kazan Federal University, Kazan, Russia
s_distefano@it.kfu.ru, kostya@esmukov.ru

Abstract. The IoT ecosystem is taking the whole ICT world by storm and, in particular for currently hot topics such as Smart Cities, it is becoming one of the key enablers for innovative applications and services. When talking about end users, or even citizens, mobiles enter the picture as the ultimate personal gadget, as well as relevant outlets for most of the duties (sensing, networking, edge computing) IoT devices are typically envisioned in the first place. Smartphones, tablets and similar accessories are even more powerful in terms of hardware capabilities (and function diversity) than typical embedded systems for IoT, but it is typically the software platform (e.g., the OS and SDK) which limits choices for the sake of security and control on the user experience. Even a relatively open environment, such as Android, exhibits these limits, in stark contrast to the otherwise very powerful and feature-complete functionalities the underlying system (i.e., Linux) natively supports. In this work the authors describe a fully user-friendly and platform-compliant approach to let users break free from some of these limitations, in particular with regard to network virtualisation, for the purpose of extending an IoT-ready Smart City use case to mobiles.

Keywords: Stack4Things · OpenStack · Fog computing · IoT · Cloud · Network virtualization · VPN · Reverse tunneling

1 Introduction

Information and communication technologies (ICT) and solutions are progressing very quickly, radically changing the landscape. Recent trends, on the one hand, pushed towards more and more powerful computing infrastructure such as the Cloud ones, providing customizable computational resources as services through the Internet, elastically, on demand. This allowed to think about new offloading patterns where business logic processing is outsourced, ubiquitously offloaded to remote server while lightweight thin client are running on local

A. Puliafito et al. (Eds.): ADHOC-NOW 2017, LNCS 10517, pp. 135–143, 2017.
DOI: 10.1007/978-3-319-67910-5_11

nodes. This, on the other hand, favoured the widespread deployment of devices, initially mainly conceived for pervasively probing real-word/physical phenomena, gradually becoming more and more powerful and 'smart'. Network, Internet and mobile computing solutions then allowed to consider this ensemble as an ecosystem of smart objects or things, giving rise to the Internet of Things (IoT). Tens of billions of things already populate the IoT ecosystem, allowing to think about novel application domains such as Smart Cities, Industry 4.0, intelligent transportation systems, e-government, to name a few, while posing challenges related to heterogeneity, networking, scalability, security, high level management, among others.

In particular, after its first inception, mobiles and personal devices triggered a new wave for IoT. Indeed, rapid advances in embedded systems (especially mobile devices), wireless sensors, and mobile communications have led to the development of more and more powerful personal devices. Modern tablets and even smartphones, in terms of processing capabilities, can be compared to 4–5 years old computers, so they can be also used as effective computing systems. This introduces further complexity and uncertainty in the widely heterogeneous IoT world, but mainly enables novel unexplored opportunities. Albeit resource-constrained, a challenge is to think on how personal device processing, storage, sensing and networking capabilities could be exploited in IoT applications. In this direction, fog-edge-mist computing [17], aiming at exploiting onboard processing capabilities to compute sensed data locally, as well as mobile crowdsensing [12], implementing participatory and opportunistic contribution patterns for crowd-sourced applications through mobile devices, are good examples. But several other opportunities can arise from mobiles in IoT, by properly exploiting their resources, ubiquity and unique combination of (built-in) technologies such as the wide range of communication subsystems. To this extent, the good news is that mobile hardware is ready for lots of scenarios. The bad one is that mobile OS/-platforms are naturally constrained due to security/privacy concerns and tight control and restrictions from manufacturers and telcos.

In this paper, we mainly focus on networking aspects, aiming at unlocking and properly exploiting communication capabilities natively provided by personal, mobile devices in IoT contexts. This is not trivial since in geographical contexts such as the IoT one several network issues can arise when interconnecting things from different administrative domains, subject to NAT, firewalls and similar restrictions. Advanced network features, widely available in opensource environments such as Linux, are definitely required by several IoT scenarios (e.g. smart city), but severely impaired under mobile OS such as Android. This is the case of network virtualization, among others. IoT network virtualization implies reconfiguration capabilities on mobile devices [10,11], since the physical environment (channels, topologies, routers) in IoT scenarios is not always under control of the designer of the infrastructure, and may be opportunistically established, e.g., volunteer-contributed. Furthermore, the configuration of most deployments (or their extent, ownership, etc.) is usually not completely known in advance, in

contrast to Wireless Sensor Networks (WSN) where some network virtualization solutions have been proposed [14].

Focusing on such specific aspects, the main contribution of this paper lies in relaxing these limitations by proposing a level-2/level-3 tunneling-based solution for IoT virtual networking adopting standard UNIX tools and mobile-native services under Android environments.

To this purpose, the remainder of the paper is organised as follows: Sect. 2 lays out the problem and related work in literature, in Sect. 3 the approach to the solution is described, Sect. 4 outlines a usage scenario together with a brief evaluation of the core mechanisms here outlined, and Sect. 5 closes the work with a summary of the results and hints to future work.

2 Background and Related Work

Network virtualization is a quite hot topic in IoT. Several works address related issues from different perspectives, including software defined networking (SDN), network functions virtualization and service function chains. A survey on the topic is available in [8], mainly focusing on the application of SDN paradigm to IoT contexts. An interesting solution is proposed in [14], integrating nodes (resource-constrained or otherwise) into a secured virtual network or IoTVN, enabling end-to-end communication among IoT nodes. The main focus is on providing a lightweight solution able to run on resource constrained device, while security and other issues are left to future work. A slightly different approach that could be somehow related to the networking topic is the opportunistic IoT one. In [13] an IoT framework is proposed extending opportunistic networking towards participatory sensing with the main aim of enabling information sharing among things to support mobile social networking. Similarly, opportunistic mobile networking is addressed in [19] dealing with low level data forwarding issues through a framework able to support and optimize opportunistic sensing. The underlying technologies are mainly based on ad-hoc network solutions.

Although promising, all these approaches partially solve the problem of connecting remote nodes in a geographical IoT context with network barriers. Ad-hoc networks and similar approaches have limited scope, while, on the other hand, the SDN approach is not lightweight and could be hardly adapted to resource constrained devices. Furthermore, restrictions from operating systems and software platform running on mobile devices require to attack the problem case by case, dealing with specific platform-dependent issues. The target of this paper is Android, aiming to implement virtual network/VPN mechanisms for IoT.

To this purpose, there are various solutions for VPN functionality under UNIX-derived systems, and even OS-native (i.e., kernel-space) VPN-enabling interfaces are available under ubiquitous Linux-based systems specifically. Indeed, under Linux tunnels may be natively instantiated using `/dev/tun` interface [2,6]. Such a mechanism has the following modes of operation:

- `tun` (OSI level 3 frames should be written to the device and read from it)
- `tap` (OSI level 2 frames)

On Android accessing `/dev/tun` requires root permissions, which cannot be granted to an application without "rooting" the device. Moreover, any functionality implemented in kernel-space requires the ability to install loadable kernel modules (LKM), another option which is not available without administrative privileges.

3 The Proposed Solution

In order to implement virtual networking/VPN mechanisms for mobiles in the IoT context, the authors base on existing solution coming from IoT and Cloud, more specifically on Stack4Things (S4T) [15]. Stack4Things is an OpenStack-based IoT framework that helps in managing IoT device fleets without caring about their physical location, their network configuration, their underlying technology. It is a Cloud-oriented horizontal solution providing IoT object virtualization, customization, and orchestration.

Among S4Tcloud-enabled services are node remoting and network virtualization. The basic remoting mechanisms are based on the creation of generic TCP tunnels over WebSocket (WS), a way to get client-initiated connectivity to any server-side local (or remote) service. In this sense, the authors devised the design and implementation of an incremental enhancement to standard WS-based facilities, i.e., a *reverse* tunneling technique, as a way to provide server-initiated, e.g., Cloud-triggered, connectivity to any board/mobile-hosted service. Beyond mere remoting, level-agnostic network virtualization needs mechanisms to overlay network- and datalink-level addressing and traffic forwarding on top of such a facility. In [16] the authors describe a model of tunnel-based layering employed for Cloud-enabled setup of virtualized bridged networks among nodes across the Internet, and Fig. 1 shows a conceptual depiction of the aforementioned model.

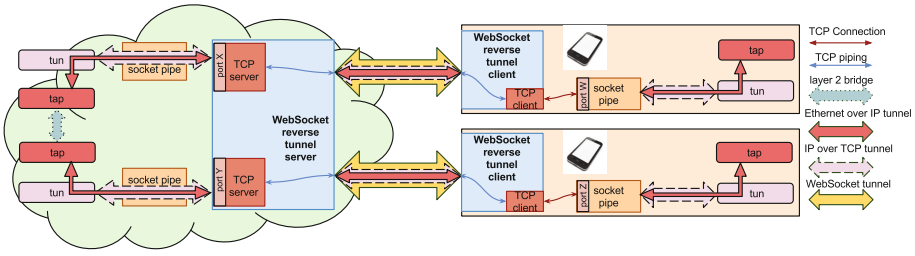

Fig. 1. Stack4Things tunnel-based layering: model.

Thus the ability to establish virtual networks among mobiles (as well as between a mobile and a server) requires at the very least core mechanisms to instantiate VPN-like tunnels between the corresponding endpoints. One way to actually create a tunnel between two Linux-like machines is using SOCAT [5].

This is a very simple program which can pipe two open file descriptors, like sockets, back-and-forth.

The most suitable way of creating tunnels on Android is to extend the `VpnService` class from Android API [7]. This class provides a simple way to create a L3 `tun` device. Actually VpnService is a thin wrapper around `/dev/tun`, which is opened in `tun` mode [4]. By default, a `/dev/tun`-based tunnel prepends (and expects to be prepended) each packet with the Packet Info (PI) structure: that is a 4 byte [3,6] struct with 2 fields: flags (2 bytes) and ether type (2 bytes). It doesn't seem very useful though, especially in L2 mode, but in L3 mode it allows to send IPv6 packets along with IPv4. Without PI only IPv4 packets might be sent. PI can be disabled by opening a tunnel with `iff-no-pi` option. Thus, in order to emulate `/dev/tun` on Android fully, `tap` mode along with PI support have been implemented leveraging the Java-based Android SDK.

Getting back to standard UNIX tooling, as leveraged in our solution, an essential requirement is fulfilled by Termux[1]. Termux is an open source app providing an Android terminal emulator and, more importantly for our purposes, a minimal Android-hosted Linux environment, and a fully Linux-compatible userland, unlike the Android-native environment. A unique feature of Termux is that it works directly with no rooting, or other complex procedure required (e.g., no user input). A base system is installed automatically by the app itself, and additional packages are then available by means of the Debian-compatible APT package manager.

A software running in Termux may then expect availability of standard, widespread userland UNIX tools, such as SOCAT, to be already installed in the environment, or at least readily provisionable by just invoking the package manager to download and install the corresponding software package. Plain SOCAT cannot be used on Android for piping to `/dev/tun` for an already stated reason: root permissions are required for that. Instead, we have created a script, which emulates `socat` binary, but actually creates a tunnel using Java application. For that, we used Android Broadcasts [1] to pass tunnel creation intentions to the APK which would actually create them.

On a first tunnel creation intent, Android forces the application to ask for consent from a user to open a tunnel. This is a security measure to prevent easy sniffing of the user traffic by an application. The consent dialog is created by Android OS and it explains to a user possible consequences of giving to an application access to tunneling the traffic through it. Consequently, to use a SOCAT-emulation script, the APK GUI should be opened first, to show the consent confirmation dialog to the user. Once consent is given, tunnels may be emulated behind scenes, thus not bothering further the user.

4 A Smart City Use Case

As described in [9], a Smart City developer may rely on an effective approach for interaction among Cloud-controlled smart objects, e.g., to let a mobile entity

[1] https://termux.com/.

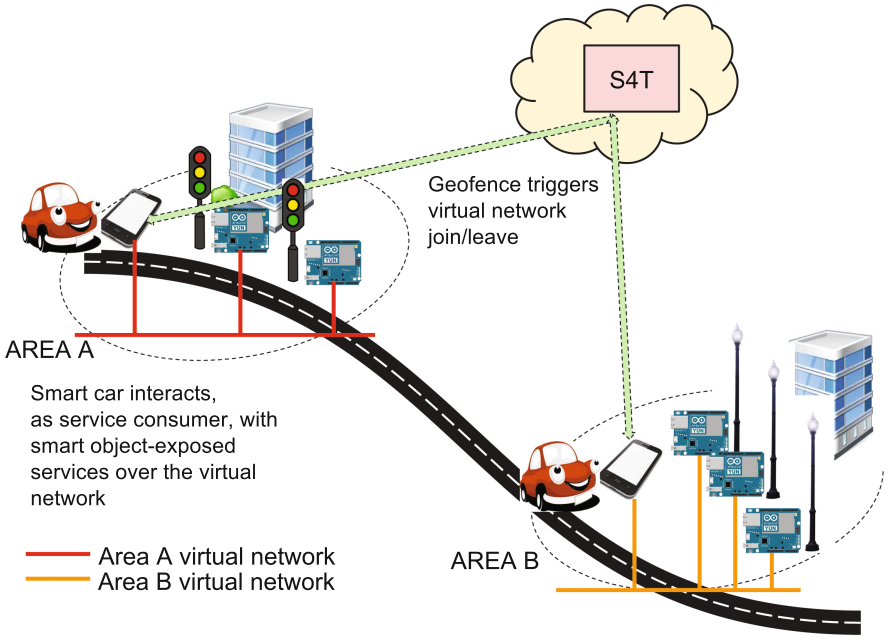

Fig. 2. VPN join, followed by service discovery

dynamically discover services and directly consume them, without the mediation of the Cloud at the application level, in a totally distributed fashion. However, for some popular service discovery frameworks, e.g., AllJoyn[2], the underlying technologies may require the discovery phase to be carried out within a single broadcast domain. Packing all services for the Smart City within such a global scope may incur in scalability issues.

Leaving aside other functionalities the Cloud, as described in [9], may provide (e.g., plugin injection, complex event processing), the virtual network instantiation and deallocation functionalities provided by Stack4Things can be very helpful in this regard. In fact, an, e.g., smart car, featuring an in-dash infotainment system based on a mass-market mobile OS (i.e., Android), may be dynamically added to virtual networks of much smaller scope, depending on the city area (e.g., a geofence-delimited one) it is traversing, discovering and consuming only the services that are listening within the broadcast domain associated with that specific area.

A service discovery-enabled client needs to be deployed dash-side for the set of services that the smart car is supposed to discover and consume, as well as discovery-enabled services on the relevant Smart City objects, such as, e.g., smart traffic lights and smart streetlights.

[2] http://allseenalliance.org.

Fully client-side geofences takes care of detecting the car entering specific areas of interest. Accordingly, addition or removal of the smart car to/from specific virtual networks is thus triggered by the smart car itself, the latter invoking the Cloud and passing the predefined (geofenced) area as parameter. As soon as the smart car gets added to a new virtual network, it discovers the services in the area and consumes them (see Fig. 2). At the application layer, the interactions between the smart car and a specific City-provided smart object are direct, without any mediation by the Stack4Things Cloud, which only provides communication services at the network layer.

A preliminary evaluation in this scenario is provided in the following, focusing on specific key performance indices, namely latency and throughput, related to the aforementioned core (enabling) mechanisms: wrapped SOCAT-based tunneling. This is tasked at quantifying the performance of the proposed tunneling mechanism, although it is a technology enabling a new feature in the IoT, thus without terms of comparison in this context so far.

Table 1. Throughput and latency measurements over WiFi.

Topology/Technology	iperf: *throughput [**Mbps**]*	ping: *latency (RTT) [**ms**]*
direct	92.5	0.23
vpn	81.75	1.501
s4t	79.7	0.625

Table 1 reports on the set of experiments that have been conducted. The experiments are based on the *iperf3* [18] tool for measuring throughput and the ubiquitous *ping* tool to gauge latency, the latter by means of ICMP echo requests to obtain Round-Trip Time values as estimation of delay. iPerf3 works by repeatedly sending an array of *len* bytes for *time* seconds, where *len* by default is 128 KB for TCP, and the default for *time* is 10 s. In both cases, the test setup consists in having a server ready, and generating traffic over TCP (iPerf3) or ICMP requests (ping) from the client (an Android mobile, or its emulated instance), which collects partial and final statistics. Values in the table are averages computed over a number of 1000 samples, where each chunk of 10 samples represent a single run for the tool. Variance values have not been included because negligible.

The first column indicates the kind of technology employed and under which topology: in particular, *direct* refers to tests between two hosts directly connected, over (WiFi) LAN. The *vpn* abbreviation refers to an OpenVPN server to which an OpenVPN client is connected, under the same roles as the two aforementioned hosts. Same happens for *s4t*, in this case with two hosts set up as if controlled by the S4T Cloud thus connected directly over a SOCAT-based tunnel.

For the sake of comparing under the most relevant conditions, OpenVPN has been tested in TCP mode, and various parameters (e.g., MTU of the TUN

interfaces, MSS for both iperf and the TCP tunnels) had already been tuned for S4T in the implementation phases, as also discussed below. As latencies are quite aligned in the various scenarios, the discussion has been focused on the more interesting and insightful values obtained for the throughput metric. Albeit actually latency may be considered the most relevant metric for the use case under consideration, as it is key for near real-time (e.g., multimedia) applications, and the most reliable metric in general for embedded systems, considering that throughput is naturally more susceptible to other factors, e.g., high CPU load or RAM usage, differences in the media interface, etc., the latter has been chosen.

It may be noticed that throughput for the S4T-based setups degrades only slightly, as can be seen in Table 1, most likely due to the overhead of inter-process piping. This as a result of trading off raw performance, at the price of high application-level complexity and an (internal) ad-hoc architecture, as is the case for OpenVPN, with the simplicity and flexibility of off-the-shelf tools acting as separate subsystems and taking care of different facets of the communication model, in line with the UNIX philosophy of using one (good) tool for each job.

5 Conclusions

In summary, in this work the authors have pursued the extension of an embedded system-powered IoT use case, developed to enable geo-localized Smart City services, to Android mobiles.

The involvement of personal devices featuring a tightly controlled environment, such as Android, called for the implementation of adaptations to the core Android-supported network virtualization mechanisms, lacking, e.g., L2 interface emulation. Other implementation choices, such as wrapping userspace Linux networking tools, derived from the requirement to preserve the same behavior, interfaces and layering model already employed by Stack4Things for embedded systems.

In terms of future work, it should be noted, that, unlike `/dev/tun` on Linux, `VpnService` cannot open multiple tunnels at once. However, this limitation might be overcome by multiplexing connections to several endpoints onto the single `VpnService` tunnel, in terms of Java-developed logic.

References

1. Broadcasts—Android Developers. https://developer.android.com/guide/components/broadcasts.html. Accessed 23 June 2017
2. linux/drivers/net/tun.c - Elixir - Free Electrons. http://elixir.free-electrons.com/linux/v4.11.5/source/drivers/net/tun.c. Accessed 23 June 2017
3. linux/include/uapi/linux/if_tun.h - Elixir - Free Electrons. http://elixir.free-electrons.com/linux/v4.11.5/source/include/uapi/linux/if_tun.h#L87. Accessed 23 June 2017

4. platform_frameworks_base/com_android_server_connectivity_Vpn.cpp at 52eb4e01a49fe2e94555c000de38bbcbbb13401b android/platform_frameworks_base - GitHub. https://github.com/android/platform_frameworks_base/blob/52eb4e01a49fe2e94555c000de38bbcbbb13401b/services/core/jni/com_android_server_connectivity_Vpn.cpp#L66. Accessed 23 June 2017
5. socat - multipurpose relay. http://www.dest-unreach.org/socat/. Accessed 23 June 2017
6. Universal TUN/TAP device driver. https://www.kernel.org/doc/Documentation/networking/tuntap.txt. Accessed 23 June 2017
7. VpnService—Android Developers. https://developer.android.com/reference/android/net/VpnService.html. Accessed 23 June 2017
8. Bizanis, N., Kuipers, F.A.: SDN and virtualization solutions for the Internet of Things: a survey. IEEE Access **4**, 5591–5606 (2016)
9. Bruneo, D., Distefano, S., Longo, F., Merlino, G., Puliafito, A., DAmico, V., Sapienza, M., Torrisi, G.: Stack4things as a fog computing platform for smart city applications. In: 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), April 2016
10. Chowdhury, N.M.K., Boutaba, R.: A survey of network virtualization. Comput. Netw. **54**(5), 862–876 (2010). http://www.sciencedirect.com/science/article/pii/S1389128609003387
11. Fischer, A., Botero, J., Till Beck, M., de Meer, H., Hesselbach, X.: Virtual network embedding: a survey. Commun. Surv. Tutor. IEEE **15**(4), 1888–1906 (2013)
12. Ganti, R.K., Ye, F., Lei, H.: Mobile crowdsensing: current state and future challenges. IEEE Commun. Mag. **49**(11), 32–39 (2011)
13. Guo, B., Zhang, D., Wang, Z., Yu, Z., Zhou, X.: Opportunistic IoT: exploring the harmonious interaction between human and the Internet of Things. J. Netw. Comput. Appl. **36**(6), 1531–1539 (2013)
14. Ishaq, I., Hoebeke, J., Moerman, I., Demeester, P.: Internet of Things virtual networks: bringing network virtualization to resource-constrained devices. In: 2012 IEEE International Conference on Green Computing and Communications, pp. 293–300, November 2012
15. Longo, F., Bruneo, D., Distefano, S., Merlino, G., Puliafito, A.: Stack4things: a sensing-and-actuation-as-a-service framework for IoT and cloud integration. Annales des Telecommun./Ann. Telecommun., pp. 1–18 (2016). https://www.scopus.com/inward/record.uri?eid=2-s2.0-84976292948&partnerID=40&md5=4f9e9d4a88b9d4e6b020331c7f92689c, cited by 0, Article in Press
16. Merlino, G., Bruneo, D., Longo, F., Distefano, S., Puliafito, A.: Cloud-based network virtualization: an IoT use case. In: Mitton, N., Kantarci, M.E., Gallais, A., Papavassiliou, S. (eds.) ADHOCNETS 2015. LNICSSITE, vol. 155, pp. 199–210. Springer, Cham (2015). doi:10.1007/978-3-319-25067-0_16
17. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
18. Tirumala, A., Qin, F., Dugan, J., Ferguson, J., Gibbs, K.: iPerf: the TCP/UDP bandwidth measurement tool. http://software.es.net/iperf/ (2005)
19. Zhao, D., Ma, H., Tang, S., Li, X.Y.: COUPON: a cooperative framework for building sensing maps in mobile opportunistic networks. IEEE Trans. Parallel Distrib. Sys. **26**(2), 392–402 (2015)