On the Exploitation of the Android OS for the Design of a Wireless Mesh Network Testbed

Matteo Danieletto[⋆,◇], **Giorgio Quer**[◇], **Ramesh R. Rao**[◇], **Michele Zorzi**[⋆,◇]
[⋆]DEI, University of Padova, via Gradenigo 6/B – 35131, Padova, Italy
[◇]University of California, San Diego – La Jolla, CA 92093, USA

***Abstract*—Wireless devices running the Android operating system offer a novel opportunity to study network behaviors and to observe and modify in real time key networking parameters. This opens up an unprecedented opportunity to study, test and evaluate the performance of techniques operating at different layers of the protocol stack and adopting the cognitive networking paradigm. In this paper, we describe our novel IEEE 802.11 mesh network testbed that integrates Android based devices. The aim is to build a flexible testbed to observe in-stack and out-stack parameters of interest, that can be used to test many networking techniques in both civilian and tactical and hostile scenarios. We provide the implementation details to create an ad hoc network among these inexpensive commercial devices, and specify how to observe and modify the networking parameters at different layers of the protocol stack. Through some examples we show the stability of the network and discuss the time evolution of some parameters of interest.**

I. Introduction and Related Work

Novel MAC, routing, transport and application layer techniques for tactical and civilian wireless mesh networks have been proposed by the networking community. However, it is difficult to find appropriate tools to compare the performance of such techniques in a realistic scenario, thus most of the comparative analyses are performed in very specific scenarios.

In the literature, there are two approaches to compare the network performance that are recognized to be sufficiently realistic. The former is based on a discrete event network simulator, and the different versions of the Network Simulator (NS) [1] have proved to be a realistic simulation tool in most cases. The advantage of this approach is that the software developed to test a specific network scenario can be easily adapted and reused by the networking community to test other techniques in slightly different scenarios. The main problem is that it is never completely clear how realistic such simulations are, as it is not possible to predict and simulate all the factors that play an important role in a real network scenario.

The latter approach includes most of the network testbeds presented in the literature, which are based on specific and often expensive hardware. Among them, CalRadio [2] is an open and reprogrammable IEEE 802.11b-compatible development platform for experimental purposes at the Data Link layer, designed and developed at the California Institute for Telecommunication and Information Technology, UC San Diego. Another interesting testbed is presented in [3], where a mobile node communicates to a fixed infrastructure composed of wireless sensors via IEEE 802.15.4 in order to locate its position as a function of the Received Signal Strength Indicator (RSSI). Another testbed solution is Roofnet [4], an experimental 802.11b/g mesh network at the Computer Science and Artificial Intelligence Laboratory of the Massachusetts Institute of Technology (MIT). In Roofnet there are 50 nodes deployed on the MIT campus with the purpose of testing different routing protocols. However, such network includes only fixed nodes, and there is no mobility involved.

The main advantage of developing a real testbed is that it allows to test the networking techniques in a real scenario, possibly very similar to the tactical or civilian scenario in which the networking techniques will be adopted. The drawback is that the nodes in such testbeds are realized with specific and often expensive hardware, making them very hard to be reused by other members of the networking community without a considerable investment of time and resources in order to replicate such testbed. Two examples of civilian testbeds can be found in [5] and [6]. The first one is the Virginia Tech Cognitive Radio Network (VT-CoRNET), developed using ad hoc hardware, i.e., 48 software-defined radio nodes and the Universal Software Radio Peripheral (USRP2). The second one is the Open-access Radio Testbed for next-generation wireless network (ORBIT), developed at WINLAB, Rutgers University, using 400 programmable radio nodes and 28 USRP2 radios.

These testbeds can enable the cognitive networking paradigm [7], according to which it is possible to learn in an automated fashion the behavior of the network by observing some key networking parameters, and to exploit this knowledge to predict the future performance of the network and act accordingly. On the other side, in a tactical environment, there are specific constraints, e.g., due to hierarchical and heterogeneous radio communication, which are typical of such networks [8] and should be kept into account when applying the cognitive network paradigm.

In our testbed we want to allow the testing of cognitive networking techniques, but at the same time we want to create a tool that is flexible enough to be used in different tactical and civilian scenarios, easy to manage, based on relatively inexpensive hardware, and that can be easily recreated by other researchers around the world. To meet these goals, we have chosen to use tablets and smartphones running the Android Operating System (OS), since these devices are commercially available, relatively inexpensive, mobile, and highly customizable. With these devices, we have realized the Android Wireless Mesh Network (AWMN) testbed, based on the IEEE 802.11 standard.

Using commercial devices in tactical networks may create some issues, since the communication should follow the National Security Agency standards for data encryption. However, [9] details how to create an architecture where

commercial and tactical networks can interact by using a specific gateway.

We stress the fact that the possibility to observe and modify in-stack parameters (i.e., those parameters that can be directly observed from the protocol stack) at different layers is the basis to implement the cognitive network paradigm beyond the physical layer [10], e.g., see how this paradigm has been adopted in a tactical multi–hop wireless network in [11]. Furthermore, since the commercial devices used to implement this testbed are equipped with other sensors like a GPS, a gyroscope and an accelerometer, it is possible to observe with such devices also out-stack parameters (i.e., those parameters that are not directly related to the protocol stack, e.g., environmental or positioning data), that can be exploited to design and test novel cognitive networking protocols using this additional out-stack information. For all these reasons, this work opens up a very promising research opportunity, since with the proposed network testbed it will be possible to test in a real and mobile scenario a vast variety of single–hop and multi–hop networking protocols.

In a nutshell, the main contributions of this paper are:

- a qualitative comparison among different networking testbeds and a description of the main advantages of a testbed based on Android;
- the realization of a wireless mesh network among Android devices with the description of the software modifications needed to enable such network using commercial devices;
- a study on how to observe and modify key TCP/IP in-stack parameters in an Android based system;
- some preliminary results to show the realistic time evolution of such parameters in our system.

The rest of the paper is organized as follows. In Sec. II we overview the main features of the Android OS. In Sec. III we describe the software changes needed to enable mesh networking in an Android device, while in Sec. IV we detail how to modify the Android OS in order to measure some key networking parameters. Then, in Sec. V we describe the testbed deployment and provide some preliminary results on the time evolution of such parameters. Sec. VI concludes the paper and presents some future work.

II. Android-based Networks

In this section we give an overview of the Android-based network ecosystem and describe some preliminary Android-based network implementations currently available. We discuss the major drawbacks of such implementations, and investigate the challenges to design a more flexible Android-based mesh network, like the one proposed in this paper.

A. The Android ecosystem

In the last few years, Android has rapidly become the most popular Operating System for smartphones and tablets, since it is running on about 70% of the smartphones in the world. Android is a very flexible OS for mainly two reasons. The former is that the Software Development Kit (SDK) for Android, i.e., the toolkit needed to develop an Application (APP), is released for all the three main commercial OSs for personal computers: Microsoft Windows, GNU/Linux, and Apple OSX. As a result of this open source philosophy, there exists a huge software community that uses Android and shares key information to develop new APPs. A developer can use the Application Programming Interface (API) released with the SDK to design a new APP, independently of the specific device used, since the APIs are common to every Android device. Furthermore, such software community shares also information on how to modify the Android system to improve the performance of the mobile devices. The latter reason is that the Android OS is based on the Linux kernel. Such kernel is released and publicly available, and it is possible to modify and compile it with the Native Development Kit (NDK), a free toolset released by the Android team. Indeed, developers with programming skills on Linux kernel module can design, write and enable new features in a relatively simple way.

The Android OS structure is divided into layers, where each layer provides different services to the corresponding upper layer. In our work we modify the software at the Application framework layer to manage the network, and the Linux Kernel, where the TCP/IP protocol stack is implemented, and from where we can observe and modify the network parameters of interest.

B. Android: advantages and challenges

In a standard Android OS, the only node to node communication mode allowed is a high level mode based on a client/server architecture. This communication mode basically exploits a standard protocol stack, where the transport layer is redesigned to create a direct communication between devices. This mode is named Wi-Fi Direct [12]. An advantage of Wi-Fi Direct is that it implements over the data link layer the Wi-Fi Protected Access (WPA2) to encrypt the messages and to ensure a secure communication. The main disadvantage is that, despite the fact that this communication mode allows the direct connection between two devices, the routing can be implemented only as an application over the transport layer. Thus, with Wi-Fi Direct it is not possible to create an efficient multi-hop network among devices.

An important advantage of the Android OS over similar OSs for mobile devices is that it is a very flexible platform that can be easily modified by an expert developer. Thus, we exploit this characteristic of the Android OS to enable a more realistic and efficient ad hoc communication mode. By modifying the Linux kernel source code, it is indeed possible to modify also the communication mode of the mobile device and to enable the pure ad hoc communication mode.

Another important advantage of the Android platform is that it is flexible enough to allow the observation and the possibility to modify network parameters at different layers of the protocol stack, thus allowing a real implementation of cross-layer techniques that follows a cognitive network paradigm. Furthermore, it is also possible to observe other out-stack parameters, that can also be integrated in a cognitive network technique.

C. Existing Android-based mesh networks

The main advantage of an Android based testbed over other networking testbeds is that it is composed of inexpensive

commercial devices and is relatively easy to replicate. Such testbed should be able to monitor and to modify some TCP/IP in-stack parameters in order to test novel cognitive networking protocols. In the literature, to the best of our knowledge, there does not exist a testbed with such characteristics. The few Android based testbeds available do not allow the possibility to observe and modify in-stack parameters, but are instead designed for a more specific application.

An interesting Android based testbed is named Serval [13]. In Serval, an Android–based multi–hop ad hoc network is implemented among smartphones running the Android OS. In this testbed, the mobile phones can communicate even in the absence of a fixed infrastructure via the ad hoc network mode. Another interesting open source software to create a mesh network among wireless devices is named Commotion [14]. In this project, Android devices and Linux based laptops can communicate in an ad hoc mode.

Both these approaches are based on an ad hoc communication mode that can be extremely helpful (1) after a sudden failure of the mobile network infrastructure, e.g., in the case of a natural disaster, (2) when the cellular network is overloaded, e.g., in an overcrowded public event, or (3) in the absence of a network infrastructure, e.g., in a hostile tactical environment. However, in Serval as well as in Commotion, only the ad hoc network mode is implemented, and the focus of such testbeds is not on the observation of the network parameters to allow the implementation of cognitive network techniques.

III. Android Wireless Mesh Network

In this section we present the Android Wireless Mesh Network (AWMN) and we describe the software changes needed to enable the Mobile Ad hoc NETwork (MANET) by which we connect the devices in our testbed. In the following we first show how to create an ad hoc network among Android devices, and then we describe the multi–hop routing protocol chosen.

A. Ad hoc network

There exist different modes, named service sets, in which an IEEE 802.11 wireless local area network can be set up. The standard one is the Basic Service Set (BSS), in which a central controller (the access point) manages the connections among the devices, organized in a star topology. Another mode is named Independent Basic Service Set (IBSS), also known as ad hoc mode, in which each device manages its own connections without a central controller. If we activate this mode, we can create a mesh network that allows multi–hop communications.

By default, the IBSS mode is disabled in the most recent versions of Android. Depending on the specific device, the IBSS mode is disabled either at the Application Framework or in the Linux kernel. In the first case, in order to connect a device in IBSS mode to a network we use the *iw* tool, i.e., a tool to configure the IEEE 802.11 driver. In other words, the *iw* can force the device to connect to an ad hoc network. This is the case of, e.g., a Samsung Galaxy Tab 7.0 device with *Atheros AR6003* chipset and *ath6kl* driver.

In the second case, when the IBSS mode is disabled in the Linux kernel, it is necessary to act directly on the Linux kernel and to modify the driver source code to activate the IBSS mode and create an ad hoc network. This is the case of, e.g., a Nexus 7 device with *BCM4330* chipset and *bcmdhd* Linux driver. We used the following patch:

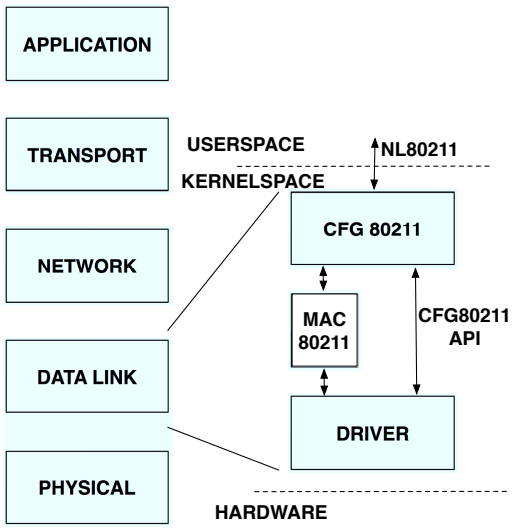


Fig. 1: TCP/IP protocol stack in an Android device.

```
+++ b/drivers/net/wireless/bcmdhd/wl_cfg80211.c
-  BIT(NL80211_IFTYPE_STATION)
+ BIT(NL80211_IFTYPE_STATION) |
    BIT(NL80211_IFTYPE_ADHOC)
```

In most devices, in order to perform these operations, it is necessary to log in as a root user and to flash the ROM memory.

B. Multi–hop network

Once the ad hoc network is set up, it is necessary to manage the multi–hop routing at the network layer. We have adopted the Optimized Link State Routing protocol (OLSR) [15]. OLSR works over the Datalink layer and is a proactive routing protocol for MANETs. OLSR sends control packets to exchange regularly topology information with other nodes of the network.

IV. Network parameters: Observation and Action

In this section we report the list of observable TCP/IP protocol stack parameter, and we detail how to observe and store them by modifying the Android software at different layers.

The TCP/IP protocol stack for the Android OS is depicted in Fig. 1. Our task is to observe and possibly modify some key parameters at different layers of this protocol stack, as well as to observe out-stack parameters of interest. In the following, we give a brief description of the key parameters observed.

A. Transport layer

Regarding the transport layer, there exist different ways to observe the parameters of interest. One of them is to read the values measured by *debugfs*, a filesystem designed

for debugging purposes. However, with this method it is not possible to read all the parameters of interest, nor to modify in real time some key parameters like the congestion window. Therefore, we have chosen to write a new kernel module in order to observe the C structure *struct tcp_sock*, i.e., a data structure with a new record written by the congestion avoidance algorithm every time a new TCP ack is received. Since *struct tcp_sock* is in the kernel space, we have to copy its values in the user space to process them and go back to the kernel space to modify it, e.g., in order to directly modify the value of the congestion window.

For this reason, we need to create a communication channel between the kernel space and the user space, i.e., an Inter-Process Communication (IPC), created via a *Netlink* socket, that provides full-duplex communication between a user-space process and a kernel module. In this way the application can observe and modify the parameters of interest in an asynchronous way.

The observed parameters are:

- Congestion Window (CWND): the congestion window value at the sender;
- Slow Start Threshold (SSTHR): the slow start threshold used by the congestion avoidance algorithm;
- Packets outstanding: the number of packets transmitted but not yet acked;
- Packets acked: the number of packets acked by the last ack packet received;
- Packets in flight: the number of packets outstanding plus the number of packets retransmitted. This number is a function of the size of the CWND (the larger the CWND the more packets in flight, and consequently the higher the probability to have a timeout or a packet loss);
- Round Trip Time (RTT) [ms]: the time interval from when a packet is sent to when the sender receives the corresponding ack;
- Smoothed Round Trip Time (SRTT): a weighted average of the past RTT. $SRTT[i] = (1-\beta) \cdot (SRTT[i-1]) + \beta \cdot RTT[i]$, with $0 < \beta \leq 1$;
- RTO: the retransmission timeout which is equal to twice the SRTT and varies between 200 ms and 2 s. A wrong setting of the RTO can induce a high rate of packet dropping.

The above parameters are readable inside the *struct tcp_sock* whenever the TCP congestion avoidance algorithm receives an acknowledgement.

B. Data Link layer

At the Data Link Layer (DLL) we can observe for each device the following parameters:

- the number of frames transmitted (TX) and received (RX), as well as the number of bytes TX and RX; these parameters are recorded in the file */proc/net/dev*;
- the transmission channel, the transmission power and the Received Signal Strength Indicator (RSSI); these parameters can be observed with the specific driver Input/Output Ctrl function, named *ioctl*.

Since the implementation of the DLL depends on the specific driver of the device, the other parameters listed in Tab. I can be observed only for some specific drivers. For the drivers *b45* and *ath5kl*, that are used by the *Broadcom bdcm4331* chipset and by the *Atheros AR5413* chipset, respectively, we can observe also the number of fragmented packets transmitted and received (# Fragments TX/RX) , the number of frames dropped (# Frames RX dropped), the number of frames retransmitted (# Frames TX retry count) and the number of frames whose retransmission failed (# Frames TX retry failed).

There are also other important parameters that can be used to estimate the Quality of Service of 802.11e [16]. These parameters can be observed and changed only in the presence of some specific drivers. These parameters are the Arbitration Inter-Frame Spacing (AIFS) time and the Contention Window range, specified by the minimum value CWmin and by the maximum value CWmax allowed. The whole set of parameters are observable because inside the driver there are some access points (hooks) to gather these quantities.

C. Out-stack

By exploiting the Android APIs we can observe also other out-stack parameters of interest, i.e., parameters that do not belong to a specific layer of the TCP/IP protocol stack. In general, a commercial tablet is already equipped with the following sensors:

- accelerometer: it can measure the value of the acceleration, in m/s^2, and its direction in the three axes;
- gyroscope: it is used to measure the orientation of the device in the three axes;
- Global Position System (GPS): if outdoor, it provides the geographical position of the device.

V. Testbed: Experimental results

The software solutions presented in the previous sections have been integrated in the deployment of the AWNM testbed, and have been tested through several experiments reported in this section. In the first experiment we tested the stability of the software that manages the ad hoc network, then in another experiment we tested the collection of the TCP/IP parameters in a single–hop network, and finally in another experiment we extended this experiment to a multi–hop network.

For each experiment, we report the time evolution of some measured network parameters.

We have deployed a wireless mesh network with the following devices: a laptop with chipset *Broadcom bdcm4331* and driver *b43*; a Nexus 7 tablet with chipset *Broadcom bdcm4330* and driver *bdcmhd*; a Galaxy Tab 7.0 (P6210) tablet with chipset *Atheros AR6003* and driver *ath6kl*; an Alix board *3d2* with chipset *Atheros AR5413* and driver *ath5kl*. The Alix *3d2* board is an inexpensive board with low power consumption and limited capabilities, equipped with a 500 MHz (LX800) AMD Geode LX CPU, which can be exploited as a router or a firewall.

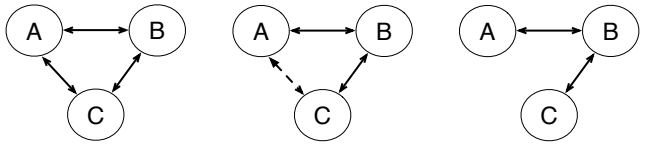
A. Experimental setup

We performed the networking experiments in our laboratory (indoor scenario). Due to the limited size of the laboratory, all nodes were in line of sight of each others (the indoor range for

TABLE I: TCP/IP stack parameters observed and modified for different devices integrated into our network testbed.

Observable/Writable	Layer	Parameters	Laptop b43 (bdcm4331)	Alix 3d2 ath5kl (ar5413)	Tab-7 P6210 ath6kl (ar6003)	Nexus 7 bdcmhd (bdcm4330)
Observable Writable	TCP	CWND	r/w	r/w	r/w	r/w
	TCP	SSTHR	r/w	r/w	r/w	r/w
	TCP	RTO	r/w	r/w	r/w	r/w
	MAC	CWmin	r/w	r/w	NO	NO
	MAC	CWmax	r/w	r/w	NO	NO
	MAC	AIFS	r/w	r/w	NO	NO
	MAC	TX power	r/w	r/w	r	r/w
Observable	TCP	IP address	r	r	r	r
	TCP	Port	r	r	r	r
	TCP	# lost packets	r	r	r	r
	TCP	# Packets in flight	r	r	r	r
	TCP	# Packets outstanding	r	r	r	r
	TCP	# Packets acked	r	r	r	r
	TCP	# Packets lost	r	r	r	r
	TCP	Throughput	r	r	r	r
	TCP	RTTvar	r	r	r	r
	TCP	SRTT	r	r	r	r
	MAC	Transmission Channel	r	r	r	r
	MAC	RSSI	r	r	r	r
	MAC	Bytes RX	r	r	r	r
	MAC	# Frames RX	r	r	r	r
	MAC	# Frames RX duplicate	r	r	r	tbi
	MAC	# Frames RX fragments	r	r	r	tbi
	MAC	# Frames RX dropped	r	r	tbi	tbi
	MAC	Bytes TX	r	r	r	r
	MAC	# Frames TX	r	r	r	r
	MAC	# Fragments TX	r	r	r	r
	MAC	# Frames TX retry count	r	r	r	tbi
	MAC	# Frames TX retry failed	r	r	tbi	tbi
	MAC	Inactive station	r	r	tbi	tbi

r = observable, w = writable, tbi = to be implemented, TX= transmitted, RX= received, # = number.



(a) All nodes are within coverage.

(b) *iptables* forces node C to drop all packets from A.

(c) OLSR updates the routing table.

Fig. 2: Changes in the network topology with *iptables*.

such devices is approximately 40 meters), as shown in Fig. 2-(a) for a simple case with 3 nodes. In the figure, node A is the source of traffic and node C is the intended receiver. In order to create a multi–hop network, we have used *iptables*, a tool available in every Linux distribution to set up different rules inside the firewall. In particular, we forced node C to drop all packets coming from node A, see Fig. 2-(b). In this way, according to the OLSR routing protocol, since node A is not receiving any acknowledge message from node C, its routing table is updated and node A sends packets to node C via node B, as shown in Fig. 2-(c), and the multi–hop network is set up.

The *iptables* commands to drop or to accept the incoming packets are:

```
iptables -A INPUT -m mac --mac-source MAC  -j DROP

iptables -A INPUT -m mac --mac-source MAC  -j ACCEPT

iptables flush
```

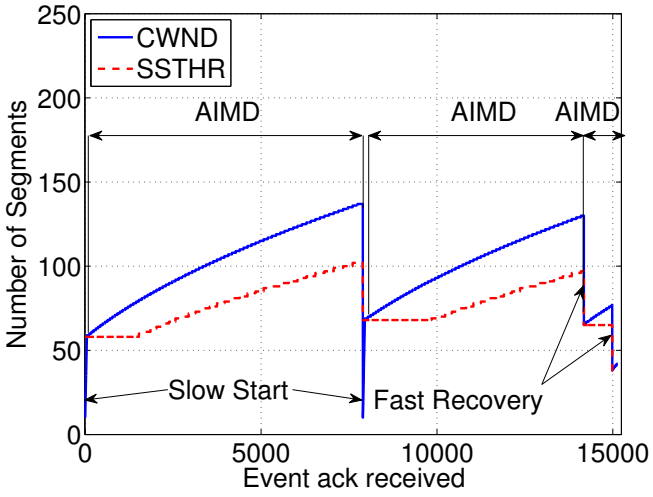
The data traffic is synthetically generated at node A using *iperf* [17], a common testing tool in the networking community.

B. Experimental results

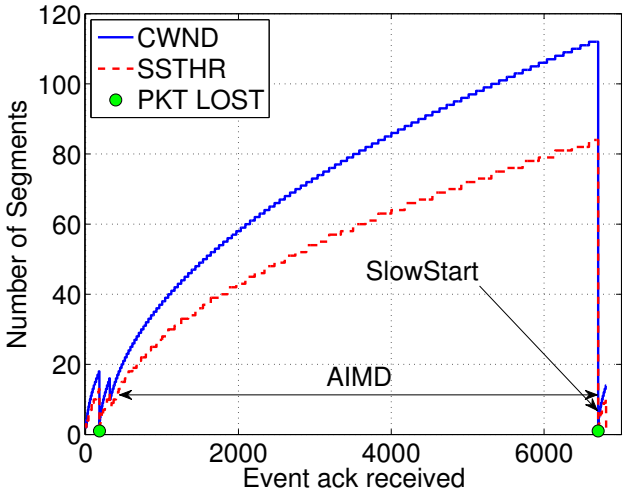
In the first experiment, there are four nodes that communicate through a multi–hop network, i.e., a Nexus 7, a Galaxy Tab 7.0, an Alix 3d2, and a laptop. The data is generated by *iperf*, and a stream of data is generated every 60 seconds and lasts for 10 seconds. The experiment is three days long, and during this period the topology of the network is changed dynamically every 6 minutes using the *iptables* tool. The purposes of this experiment are to test if the network testbed is stable and to figure out if there is any problem in updating the routing table with the OLSR protocol. We have verified that the routing protocol works correctly and that the network testbed can easily handle dynamic changes in the topology.

In the second experiment, we have tested the software tools to observe the parameters, in order to check the reliability of such software and to observe the TCP and MAC parameters as a function of time. The list of all the parameters observed is reported in Tab. I, where we detail for each device which parameters are observable, and which can also be modified. For this experiment, which lasted a total of 8 hours, a single–hop topology has been created. In the figures we report a time interval of 30 seconds only, since we want to focus on a single stream of data generated by *iperf*.

In Fig. 3 we can observe the two phases of the TCP Reno congestion control, i.e., the slow start mode, and the Additive Increase/ Multiplicative Decrease (AIMD) mode. In Fig. 3-(a)



(a) Fast recovery mode with TCP Reno congestion avoidance algorithm.



(b) Slow start mode with TCP Reno congestion avoidance algorithm.

Fig. 3: Behavior of TCP RENO showing both fast recovery and slow start mode.

TABLE II: Parameters observed at the receiver node.

Average RSSI [dBm]	Average RTT [ms]	RX packets	RX fragments	RX duplicates	RX Dropped	TX retry count	Data RX [MBytes]
-62.1	65.7	15104	261	14	12	1035	$\sim$ 22

we show the values of the CWND and SSTHR during the slow start mechanism that occurs when a new TCP connection is opened. In this case we set manually the CWND to 10 packets. The values of CWND and SSTHR are updated at every ack packet received, so we represent their value as a function of the number of acks received in a temporal window of 10 seconds[1] . Initially, the CWND exponentially increases until it reaches the current value of SSTHR, then the AIMD mode starts. We can also observe the fast recovery event that occurs after three duplicate acks are received. In this case CWND and SSTHR are both set to a value equal to half of the previous value of the CWND.

Instead, in Fig. 3-(b) a packet is lost and the TCP Reno protocol goes into slow start mode with the CWND set equal to 1 packet. In this experiment we have also verified that the CWND value is equal to the sum of the packets in flight and packets acked parameters when there are no retransmissions. In Fig. 4 we show the number of packets per second transmitted and received by the receiver node as a function of time. The total data received corresponds to a stream with bitrate of 17 Mbps generated at the source node by *iperf*, lasting for 10 seconds for a total of approximately 22 MB of data. We notice that the number of packets sent from the receiver are half of the number of packets received. This is due to the TCP delayed acknowledgment, a technique used to improve network performance in which the receiver combines together into a single response two acks, reducing the protocol overhead.

For this experiment in Tab. II we report a summary of the observed data during the 30 seconds of the experiment. Finally,

[1]Note that the fact that the value of the SSTHR observed is not constant between consecutive timeout events is due to an implementation detail of the corresponding TCP counter in the Linux kernel.

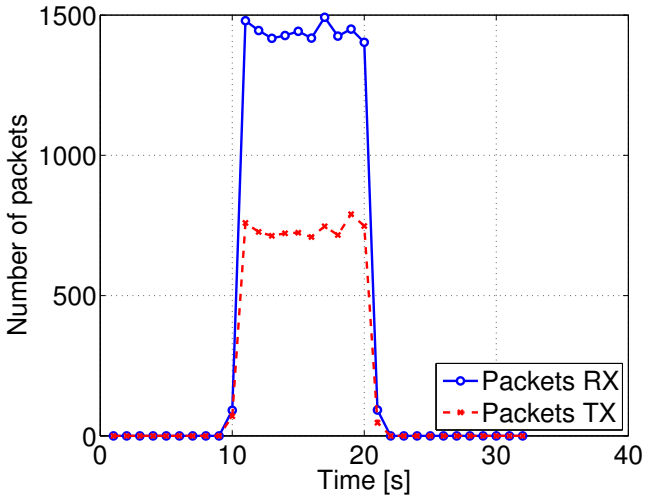


Fig. 4: Packets transmitted and received by the receiver node C.

in the last experiment we tested the software reliability and observed some TCP and MAC parameters as a function of time in a multi–hop scenario. The network topology is shown in Fig. 2-(c), where node A is the source node, node C is the destination node and node B is the relay node. In Fig. 5 we show the number of MAC packets sent by node A to node B, and the corresponding number of MAC packets sent by node B to node C. As expected, these two numbers are very close, since B is just a relay for the communication from A to C. In the same figure, we also show the number of MAC packets sent from node B to node A, which correspond to the TCP acks sent by C to A and forwarded by B. We also notice that after 28 seconds node C becomes congested due to a timeout

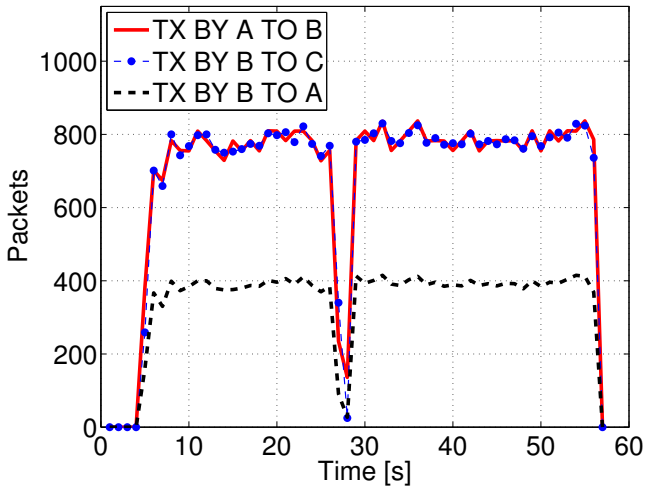


Fig. 5: Packets transmitted by the source node A and the relay node B.

event, thus the number of packets transmitted and received rapidly goes to zero.

VI. Conclusions

In this paper we described an IEEE 802.11 mesh network testbed, which is scalable and integrates relatively non-expensive Android based devices. We have provided the software details to create an ad hoc network among these inexpensive commercial devices, and specified how to observe and modify the networking parameters at different layers of the protocol stack. The software developed to observe and manage the TCP and IP layers is fully re-usable on any GNU/Linux device. Instead, the implementation of the MAC layer depends on the specific chipset of the device. The proposed testbed can be an important tool for future performance comparisons among cognitive networking techniques.

In future works, we plan to extend the current set of capabilities of the proposed network testbed to the observation of other out-stack parameters, and to integrate such observations into a novel cognitive networking framework that can exploit both in-stack and out-stack parameters. Also, we are planning to develop an Android APP to display in real time the node networking status in terms of traffic data generated, congestion of the node and number of hops needed to reach the other nodes. Such APP can allow the user to select a cognitive networking technique among a set of techniques implemented in the device and to test in real time the performance of such techniques, thus opening up new opportunities to switch in real time among different cognitive networking techniques depending on the specific networking scenario and on the user needs.

Acknowledgments

This work was partially supported by the U.S. Army Research Office, under Grants No. W911NF-11-1-0538 and W911NF-11-1-0336, and by Fondazione Cariparo through the program "Progetti di Eccellenza 2011-2012."

References

[1] "The ns-3 network simulator," Last time accessed: Sept. 2013. [Online]. Available: http://www.nsnam.org/

[2] A. Jow, C. Schurgers, and D. Palmer, "Calradio: a portable, flexible 802.11 wireless research platform," in *1st International Workshop on System Evaluation for Mobile Platforms*, ser. MobiEval '07, New York, NY, USA, 2007, pp. 49–54.

[3] A. Bardella, M. Danieletto, E. Menegatti, A. Zanella, A. Pretto, and P. Zanuttigh, "Autonomous robot exploration in smart environments exploiting wireless sensors and visual features," *Annales des Télécommunications*, vol. 67, no. 7-8, pp. 297–311, 2012.

[4] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11b mesh network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 121–132, Aug. 2004.

[5] T. Newman, A. He, J. Gaeddert, B. Hilburn, T. Bose, and J. Reed, "Virginia tech cognitive radio network testbed and open source cognitive radio framework," in *Proc. 5th Int. Conf. Testbeds and Research Infrastructures for the Development of Networks and Communities and Workshops*, 2009.

[6] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh, "Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols," in *IEEE Wireless Communications and Networking Conference*, vol. 3, 2005, pp. 1664–1669.

[7] J. Mitola and G. Q. Maguire Jr., "Cognitive radio: making software radios more personal," *IEEE Personal Communications*, vol. 6, no. 4, pp. 13–18, 1999.

[8] O. Younis, L. Kant, A. Mcauley, K. Manousakis, D. Shallcross, K. Sinkar, K. Chang, K. Young, C. Graff, and M. Patel, "Cognitive tactical network models," *IEEE Communications Magazine*, vol. 48, no. 10, pp. 70–77, Oct. 2010.

[9] G. Elmasry, R. Welsh, M. Jain, B. Hoe, K. Jakubowski, K. Whittaker, and G. Oddo, "The role of network operations in bringing commercial wireless to tactical networks," in *IEEE MILCOM*, Baltimore, MD, US, Nov. 2011.

[10] R. W. Thomas, D. H. Friend, L. A. DaSilva, and A. B. MacKenzie, "Cognitive Networks: Adaptation and Learning to Achieve End-to End Performance Objectives," *IEEE Commun. Mag.*, vol. 44, no. 12, Dec. 2006.

[11] G. Quer, H. Meenakshisundaram, B. Tamma, B. S. Manoj, R. Rao, and M. Zorzi, "Using Bayesian Networks for Cognitive Control of Multi-hop Wireless Networks," in *IEEE MILCOM*, San Jose, CA, US, Nov. 2010.

[12] Wi-Fi Alliance, "Mobile Ad-Hoc Networking: Wi-Fi certified IBSS with Wi-Fi Protected Setup," Dec. 2012. [Online]. Available: http://www.wi-fi.org/

[13] P. Gardner-Stephen and S. Palaniswamy, "Serval mesh software-wifi multi model management," in *1st International Conference on Wireless Technologies for Humanitarian Relief*, ser. ACWR '11. New York, NY, USA: ACM, 2011, pp. 71–77.

[14] "Commotion Project," Last time accessed: Sept. 2013. [Online]. Available: https://code.commotionwireless.net/projects/commotion

[15] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized Link State Routing protocol for ad hoc networks," in *IEEE International Multi Topic Conference (INMIC)*, 2001.

[16] "IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements - part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, pp. 1–1076, 2007.

[17] "Iperf website," Last time accessed: Sept. 2013. [Online]. Available: http://iperf.sourceforge.net/