ACCEPTED IN WOWMOM 2023

TWIST: Thin-Waist Wireless Testbed for Measuring Interfering Traffic Stream Throughputs

Y. Thomas and S. Toumpis
MMlab, Department of Informatics, Athens University of Economics and Business (AUEB), Greece
E-mail: {thomasi, toumpis}@aueb.gr

***Abstract*—Due to the unpredictable nature of the wireless channel and its complicated interaction with the various parts of the protocol stack, evaluating accurately the effects of interference on the throughputs of competing traffic streams of wireless networks is typically impractical using analysis or simulations. However, knowing such throughputs is crucial to the performance evaluation of any wireless network designed to carry high-volume, such as multimedia, traffic and, on a more fundamental level, to the estimation of the network's Capacity Region (CR).**

In this context, we have developed the Thin-waist Wireless Interfering traffic Streams Testbed (TWIST), a small-scale testbed designed specifically for addressing the unique challenges of measuring efficiently the throughputs of competing traffic streams and estimating the CR. Central to TWIST is a monitoring and management function that orchestrates the conduction of measurements and the configuration of the nodes. The testbed is remotely controlled by a web application interface (API) through which remote processes, such as high-level language programs, can consume the testbed as a service, conducting sequences of measurements determined beforehand or online.

I. Introduction

A fundamental problem in wireless networks, and a watershed that sets them apart from wired ones, is the fact that transmissions interfere with each other; the effects on the network's performance are very hard to estimate through analysis or even simulations, due to the complexity of the interaction between the wireless channel and the various elements of the protocol stack.

However, determining accurately the throughputs of competing traffic streams is very important in the performance evaluation of wireless networks supporting high-volume, such as multimedia, traffic. On a more fundamental level, it is also important for assessing the *theoretical* throughput limitations of the network; in the related literature, these limitations are described in terms of the set of all combinations of data rates with which the links in the network can operate simultaneously, referred to as the **Capacity Region (CR)** [1].

With the purpose of supporting analysis and models with real-life implementations, numerous experimental testbeds for wireless networking research have been introduced. Nevertheless, the nature of estimating the throughputs of competing traffic streams presents individual challenges that typically wireless testbeds do not place on high priority; indeed, besides delivering an accurate and consistent measurement of traffic stream throughputs, the testbed must minimize the temporal overhead of each measurement, in order to enhance scalability, and must provide a low-latency centralized coordination plane in order to direct complex experiments precisely. To the best of our knowledge, there is no testbed specifically built to address these issues yet.

In this work, we introduce and describe the architecture of the **Thin-waist Wireless Interfering traffic Streams Testbed (TWIST)**, a small-scale, easy-to-use wireless testbed specifically built for calculating the throughputs of competing wireless traffic streams and supporting research on CRs and related topics. Currently, TWIST includes 20 Raspberry Pi nodes and covers an area of approximately $750\ \mathrm{m}^2$ (per floor) in three different floors. The design and implementation of TWIST addresses the aforementioned challenges and delivers an evaluation tool that estimates with low temporal overhead and accurately the measured throughputs in the face of complex interference patterns. In order to do so, TWIST introduces a novel monitoring and management plane for detecting the network state, configuring node operation and deploying complex experiments. In its current configuration, TWIST uses the internal IEEE 802.11 cards of the nodes for forming the wireless network; however, other MAC/PHY hardware can be used as well with minimum modifications.

Regarding its usage, TWIST offers a generic web application interface (API) through which remote processes can consume the testbed as a service, thus allowing researchers to remotely specify and conduct network measurements as needed. This API is conceptually the *thin waist* of the testbed; below it, there is a significant amount of mechanisms for ensuring the smooth operation of the network and the execution of the API, that the API's user need not be concerned with; above it, the user of the API can *emulate* a wide variety of protocols, also of significant complexity, using the API only in order to retrieve the throughputs of competing traffic streams. The testbed is available for use by the wider community, through accessing the API, or indirectly, through an extensive set of measurements that have been made available. The testbed has already been used in estimating the CR of a 16-node network [2].

The rest of the paper is organized as follows. In Section II we discuss related work on wireless testbeds and also the estimation of Capacity Regions (which was a prime motivation in developing the testbed). In Section III we introduce the TWIST testbed, in Section IV we present some important findings from its operation, and in Section V we elaborate on the challenges of building it. Finally, we deliver our conclusions and comment on our future work in Section VI.

II. Related Work

A. Wireless testbed research

As a preliminary comment, TWIST is a small-scale testbed that is similar to many of the already available testbeds in certain aspects, such as separating the control and data planes, supporting custom protocol installation, adopting centralized management, and offering online result reporting and use by the wider community. However, to the best of our knowledge, no other testbed focuses on online interfering traffic stream measurements in a timely and accurate manner or adopts its thin-waist architecture.

Historically, earlier testbeds were also relatively small-scale affairs; more recently, platforms have been introduced for operating larger-scale, interoperable testbeds that are available for use by the wider community. Notable examples along this thread are the recently completed EU-based Fed4FIRE and Fed4Fire+[1] projects and the ongoing US-based Platforms for Advanced Wireless Research Program (PAWR)[2]. The interested reader is referred to [3] where the current state of the art is reviewed in detail.

Regarding earlier small-scale testbeds, CorNet [4] is a collection of 48 software-defined radio nodes deployed within a four-floor building. The testbed promotes research on radio interference focusing on dynamic channel selection policies.

Also, the Carmen testbed [5] is a mesh network of Android-based devices. The devices are configurable with enough computational resources to support the tools and means to make wireless network performance measurements. Although the nodes have resource limitations associated with Android devices, Carmen excels at exploring mobility.

The W-iLab.t testbed [6] includes 200 sensor and IEEE 802.11-enabled nodes. W-iLab.t puts the focus on sensor networks and assists the interoperability of different wireless technologies, but could also be used for performing interfering traffic stream measurements if the control plane logic of TWIST is installed.

The OfficeLab testbed [7] consists of 40 nodes supporting IEEE 802.11 and sensor technologies. OfficeLab is recommended for development and testing of smart services based on IoT sensors and actuators in a real-life experimental environment.

Finally, perhaps the testbed most related to TWIST is introduced in [8], where the authors perform online optimization of 802.11 mesh networks. In particular, they first estimate online the CR of the network under the strong assumption that interference between links is binary (i.e., two links either do not interfere at all or cannot access the medium concurrently), and then use this CR estimate to perform end-to-end rate control. Our work here is complementary to that work, as we can use our measurements to provide a more accurate CR estimate, even when interference is not binary (which is, in fact, the case in our testbed, as we discuss later on).

[1]https://www.fed4fire.eu/
[2]https://advancedwireless.org/

Moving on more recent, larger-scale testbeds, CityLab allows researchers to experiment in scenarios where a variety of wireless technologies are used in parallel within an urban environment; one outcome of this testbed is that *future* interference can be predicted using a neural network model [9].

The large-scale Cloud Enhanced Open Software Defined Mobile Wireless Testbed for City-Scale Deployment (COSMOS) testbed [10], a PAWR testbed, covers a square mile of dense urban environment with software-defined radios and is supported by cloud/optical infrastructure.

The Platform for Open Wireless Data-driven Experimental Research (POWDER) testbed, part of PAWR, offers experimentation on a variety of environments (which include mobile environments through the use of shuttles) using software-programmable radios, with research covering a wide range from massive MIMO to low-power WANs [11].

The Aerial Experimentation and Research Platform for Advanced Wireless (AERPAW) testbed, part of PAWR, uses significant UAV resources, integrating UAV and 5G research [12].

Finally, the Road-, Air- and Water- based Future Internet Experimentation (RAWFIE) project offers a variety of testbeds that allow experimentation with autonomous land, air, and, unusually, sea vehicles [13].

B. Capacity region research

Various definitions for the CR exist in the literature. In this work, the CR of a network with a total of L links is the set of all L-dimensional vectors that describe combinations of link throughputs that are achievable either concurrently or using time division. Therefore, the CR completely captures the effects of interference between wireless links.

In principle, estimating the CR presents poor scalability due to the cost of measuring the performance of all 2^L link combinations, which we call **Transmission Modes (TMs)**. However, preliminary results in our prior, pre-TWIST work [1] suggest that novel pruning solutions based *(i)* on greedy algorithms as well as *(ii)* online Machine Learning (ML) can achieve an accurate approximation of CR by exploring only a small subset of TMs. Specifically, a greedy algorithm can be used for pruning the set of TMs that should be explored, e.g., exclude a TM where an individual node participates in multiple links or where a subset of the TM has been found to be inefficient. Moreover, ML techniques can be used to predict the performance of TMs: using the already available measurements as a training set, regression can be used in order to estimate the performance of a yet unmeasured TM and decide if it is worth measuring; if the decision is positive, then the TM is measured and it is added to the training set; otherwise, it is skipped, thus accelerating the CR-estimation process. Observe that the two pruning solutions are stateful (since the previous measurements define which TMs should (not) be measured next) and complementary (hence can be combined ad hoc in order to maximize the gains).

Our preliminary work in [1] provided the motivation for building TWIST. In this work, we focus on presenting the

testbed itself and its potential application by the wider community. In our parallel work [2] we develop the theory of estimating the CR and apply it using the measurements derived using the testbed.

III. Thin-waist Wireless Interfering traffic Streams Testbed (TWIST)

A. Testbed goal and requirements

The primary aim of the testbed is to measure the throughputs achieved by concurrently active, *multihop traffic streams*, which will be cross-interfering as well as self-interfering. We will refer to all these combinations of streams as **Stream Modes (SMs)**, which complements the definition of Transmission Modes (TMs), which are combinations of *transmitting links*. Observe that each TM is a SM, comprised of single-link traffic streams, but not the opposite.

This primary aim presents some unique requirements, besides delivering an accurate measurement of each stream throughput, that typically wireless testbeds do not consider:

a) Minimum temporal overhead: In the case the testbed is used for CR estimation, 2^N TMs exist in a topology of N links. While the number of TMs that must be measured can be substantially reduced by exploiting the aforementioned pruning solutions, minimizing the duration of the necessary measurements is important for accelerating the estimation and making the method practical for real-life wireless networks.

b) Simultaneous link activation: In order to capture interference accurately, the interfering streams must be activated in parallel, measuring each link's performance in the exact time frame. The measuring time frame of a SM is typically too short to amortize any performance advantages gained by the links that are activated first, thus rendering the (as much as possible) simultaneous activation of all involved links critical. The activation of tens of streams with the required millisecond-level precision, in a real testbed, is a challenging process from both a networking and systems perspective.

c) Timely result reporting: When the testbed is used for CR estimation, and because pruning solutions are stateful, it is necessary to report the result of each measurement, so that the next measurement may be decided upon. The same need is expected to exist when the testbed is used in many other settings, as discussed later on. Therefore, aggregated reports are not recommended and the requirement for reporting results with the least possible delay is highlighted.

B. The waist: Testbed as a service

The testbed offers a simple, yet versatile, API through which remote applications can explicitly request arbitrary SM measurements. This API is exploited in the study of [2] (where a ML-based pruning module is remotely implemented in order to create a CR estimate) and in potentially other settings, as discussed in Section III-F.

In order to assist the real-time interaction with the testbed as well as to avoid sending requests to the testbed when a previous experiment is still running, the API is blocking, hence the results are returned to the calling process when the requested experiment ends and, also, there is a check for pending requests performed before the execution of a new request. The communication with the API is carried over SSH,[3] i.e., the following command (via a Debian-like shell application) requests a TM measurement:

$ ssh *ACC@IP* "*API.sh DUR SM*"

where *ACC* and *IP* are the account name and the IP address of the machine that hosts the API, respectively, *API.sh* is the name of the script that handles the request, *DUR* is the duration of the experiment, in seconds, and *SM* is the Stream Mode to be explored. *SM* is a list of streams separated by white spaces (e.g., "*STREAM1 STREAM2 STREAM3*"), each stream being a pair of node IDs, denoting the source and the destination of the stream, separated by a comma (i.e., "*srcID,dstID*"). For example, "*50,51 52,53*" encodes a SM of two streams, from nodes 50 and 52 to nodes 51 and 53 respectively. The returned result is either N transmission rates (in Kbps) corresponding to the N streams of the SM or, in case of failure, an error code indicating the cause of failure.

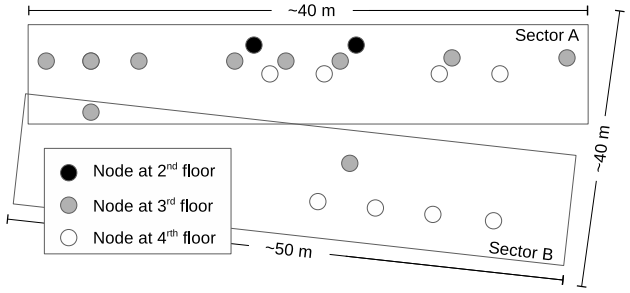

Figure 1. Testbed overview. 20 Pis scattered in a 2-sector campus building in 3 floors.

C. Below the waist: Infrastructure / Hardware

The testbed is located at a five-floor two-sector building of AUEB in Athens, Greece. As shown in Fig. 1, 20 nodes are placed in labs, offices and open spaces at Floors 3, 4, 5 in two sectors covering an area of approximately 750 m^2 at each floor, however ad hoc extensions and modifications within reason are expected to take place as needed.

The wireless network nodes are Raspberry Pis 4 B+[4], which are commodity single-board computers with enough computing resources to run open-source implementations of network protocols, such as OLSR,[5] Babel[6] and TCP (with various congestion control algorithms). Regarding hardware, the only extension to the stock Pis is the addition of an external USB-powered dual-band IEEE 802.11 interface; the additional external interface allows the parallel connection of the Pis to two individual networks, namely, the **data** network, where measurements take place, and the **control** network, where the communication between Pis and the experimenter takes place.

[3]https://linux.die.net/man/1/ssh
[4]https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
[5]https://manpages.org/olsrd/8
[6]https://linux.die.net/man/1/babel

The use of a different channel for controlling the network and transferring secondary information, such as network statistics and measurement, separates the control plane from the data plane, thus enhancing the isolation of the data network and, in turn, the integrity of the measurements.

Regarding software, the Pis run the Raspbian OS with the 5.4.72-v7l+ kernel and applications, drivers and services that are pulled via the stock Raspbian software repositories. The only third-party driver, that was downloaded from a private software repository due to the lack of native support, concerns the external IEEE 802.11 interface that is used in the control network (and thus has no bearing on the measurements).

D. Below the waist: Logic / Software

The testbed follows a centralized scheme which consists of the network nodes and a single **controller** node. The controller node interprets instructions received through the API and directs the network nodes to make SM measurements. Therefore, the cumbersome testbed logic is implemented at the controller, allowing the nodes to operate in a stateless and low-complexity fashion. Accordingly, the core testbed operation is encoded in two scripts, the **controller script** that runs at the controller node and the **listener script** that runs at each testbed node. The first is a sophisticated script that specifies the testbed operation, while the latter is a simple stateless script that receives and responds directly to incoming requests, such as activating a link and reporting measurements or statistics.

The script logic is implemented through Debian-compatible BASH scripts. While the implementation cost of BASH programming can be higher compared to more advanced programming languages, BASH scripts are very efficient in controlling real hardware devices due to their inherent support of system calls, thus delivering *faster* operation and more *direct implementation* of tools that manage hardware, such as network interfaces and the file system.

Moreover, using BASH scripts, we exploit native libraries of the controller node as well as the testbed nodes for accomplishing the most critical functionalities of the testbed. In the following, we summarize the basic libraries that we exploit:

a) brcmfmac: All network nodes exploit the same wireless device, which is controlled by the brcmfamc driver[7], allowing for a uniform testbed configuration. The setup supports IEEE 802.11ac, however the device managing tool (*iw*) reports simply that all nodes are using the IEEE 802.11 protocol.

b) ping: The connectivity of the network is discovered by sending ICMP packets via the Ping tool[8]. Using the control plane, the controller directs every node to "ping" every other node through the data plane, thus discovering the reachable unidirectional single-hop links.

c) iperf: The achievable throughput of a stream is detected via the iperf tool.[9] iperf deploys a typical TCP connection between two IP-enabled nodes sending the maximum achievable amount of "dummy" data for a predefined time interval and, at the connection's end (and optionally periodically), reports the overall throughput. Using the control plane, the controller directs the destination node to run iperf in server mode, that is to wait for incoming TCP connections. Then, it directs the source node to run iperf in client mode towards the destination node through the data plane.

d) routing: The testbed supports the BABEL and OLSR ad hoc routing protocols through their available implementations at the Raspbian repositories. The implementations are deployed as autonomous background system processes, hence the operation of routing protocols is transparent and orthogonal to the operation of the testbed. When deployed, the protocols silently update the routing table of the nodes, thus allowing the establishment of (iperf) connections to nodes that are also indirectly reachable; without a routing protocol, iperf connections to such nodes are rejected.

e) traceroute: When assessing the performance of multi-hop streams, it is insightful to detect the format of the dissemination paths. To this effect, the testbed exploits traceroute,[10] a tool which reports the intermediate hops in the dissemination path, allowing for a refined assessment of the performance, e.g., studying the performance versus the path length, or exploring the stability and responsiveness of routing protocols during network load oscillations.

f) ssh and netcat: The controller remotely executes local commands at the testbed nodes via the SSH and netcat[11] tools. While SSH natively supports remote command execution, it also induces performance overhead due to the requirement to secure the communication, hence is preferable when the performance requirements are loose, such as configuring nodes before the measurement. Netcat is a minimalistic networking tool, in the sense that it merely opens a TCP or UDP socket that can carry text, and it is light and fast but also costlier to implement; in order to remotely execute a command, the controller must explicitly implement the opening (and closing) of sockets, the translation of the received "text" to local commands, as well as port-multiplexing and process forking techniques to support non-blocking communication. Netcat is exploited in functions that are time-critical, such as taking measurements.

E. Below the waist: Operation

Having specified the hardware and software aspects of the testbed, we next describe the operation of two core functions.

a) SM measurement: A SM measurement is carried out in three consequent steps. First, the controller parses the SM to be measured and kills any pending or stalled iperf processes at the involved nodes in order to ensure the atomicity of each measurement. Then, the controller deploys the iperf servers, then it deploys the iperf clients and the associated transport of data, and finally waits until the connections end. At the third phase, the controller pulls the results, that are temporarily

[7]https://openwrt.org/docs/techref/driver.wlan/brcmfmac
[8]https://linux.die.net/man/8/ping
[9]https://iperf.fr/
[10]https://linux.die.net/man/8/traceroute
[11]https://linux.die.net/man/1/nc

stored at the receiver nodes, and updates the appropriate local log files. In order to ensure the simultaneous activation of all involved iperf clients, the controller spawns a parallel background process per client node; each process deploys the appropriate connection from the client node to the sender node. We measured the deployment latency between the 1^{st} and the 20^{th} links in 20-links TMs through monitoring the packets at the controller NIC, and the result over 100 deployments of different TMs was 46 ms on average. To further mitigate any impact of the deployment latency, we randomly shuffle the deployment order of the links at each measurement, thus distributing the impact of the latency uniformly to all links.

b) Topology discovery: As part of the network monitoring, a mechanism is in place for tracking the network topology on demand, independently of the operation of the routing protocol. Discovering the existing links in large topologies can be hard, since the number of candidate links is $N(N-1)$ for N network nodes. As a rule of thumb, for each node pair, 5 ICMP messages are sent consecutively with 0.2 s inter-transmission rate and a 0.5 s waiting period for the last ICMP message; 0, 1-3, and 3-5 responses indicate no connective, poor connectivity and good connectivity, respectively. We accelerate the process by simultaneously pinging all other nodes from one node at a time.

F. Above the waist: Emulation

Operators of the testbed can use the API, as described in Section III-B, in order to initiative successive SMs specified by the experiment they have designed, measuring the actual throughputs of the SMs but *emulating* those parts of their setup that exist above the transport layer. The emulation allows the fast and accurate performance evaluation of protocols, as it allows the execution in the testbed of that aspect of the experiment that cannot be emulated accurately, i.e., the transport of data under interference conditions.

As a first, implemented, example of this mixed testbed architecture, in [2] we have used the API to calculate the CR of a 16-node network by running off-site the Sequential Expansion Algorithm, described in detail in [2].

As a second, also implemented, example, in [2] we have also developed the **Network-Wide Time Division Protocol**, which accepts as input the calculated CR and executes a network-wide time division, in which time is divided in frames, and each frame is divided in slots during which a specific TM is activated. The testbed was used for evaluating the performance of the proposed scheme, by measuring the throughputs of each TM using the API, but the transport of data was emulated centrally by a high-level programming language, as opposed to being executed in the testbed.

As other potential examples of using the architecture, the testbed can be used to provide accurate throughput measurements needed in the design and analysis of blockchain networks [14] or networks performing federated learning [15].

One potential optimization of this approach is to maintain, above the waist, a data bank of past measurements and recycle them, when this does not affect the integrity of the results. For example, an experiment can run multiple times, for different sets of protocol parameters, and for each experiment measurements taken in past experiments may be reused, at most once, provided conditions in the network have not changed.

IV. Evaluation

In this section we examine the efficiency of the testbed in making SM measurements timely, accurately, and consistently and then we discuss our findings from conducting a few sets of measurements in conditions of interest. If not stated otherwise, the default configuration parameters are that the RTS/CTS handshake is not activated, the CUBIC variant of TCP is selected, the duration of TCP flows is 3 s, and a channel in the 2.4 GHz frequency band is used.

A. Measurement latency

We first examine the latency that is introduced by the testbed when measuring the throughputs of a SM. This latency comprises two parts, the setup latency and the assessment latency. As we are especially interested with the latency associated with computing CRs, in this section we limit ourselves to single-link streams, and therefore TMs instead of SMs.

a) Setup latency: This quantity captures the latency due to the orchestration of a single measurement. We detect two time-consuming functions: waiting for the deployment of iperf servers before launching the iperf clients and waiting for the termination of iperf flows before fetching the results.

First, server deployment should precede client deployment in order to avoid connections being rejected. Adding a deployment delay is preferred from the alternative solution, which is to keep servers persistently active, since the latter penalizes the atomicity of the measurements and, in turn, their credibility; in practice, a 0.5 s delay is found to be effective for 20-link TMs.

Second, pulling the results from the nodes can be time-consuming. The controller must instantly pull data from all the clients over the wireless control plane network in order to minimize the latency in the response of the API. This can lead to packet bursts and, in turn, connection failures that the controller overcomes through redundancy (redundant data pull requests and responses) and a timer-based retransmission mechanism with a 0.3 s time frame. An additional issue arises when the iperf connections terminate past the specified measurement duration, thus slightly delaying the report of the overall transmission rate. To address this issue, the controller enables iperf's periodic transmission rate reporting (1 s period) and, without any additional delay, pulls the latest reported measurement in case iperf has not finished in time.

We evaluate the set-up latency by deploying 100 TMs of 1 s duration consecutively, excluding any processing in between. We measure the overall completion time, subtract the duration of the actual measurements, that is 100 s, and then divide it by the number of TMs, that is 100. We repeat the experiment for TMs with 1, 10 and 20 links and present the results in Table. I. Again, the overhead increases with TM size, verifying that organizing efficiently the measurement of larger TMs is

TM size (links)	1	10	20
Setup latency (s)	1.03	1.98	3.03

Table I
SETUP LATENCY FOR TMS OF DIFFERENT SIZES.

indeed a hard task. The orchestration of 20 streams over a wireless best-effort network is challenging, and this fact must be taken into account by the users of the testbed.

b) Assessment latency: This quantity captures the latency needed for the method that estimates stream throughputs (in our case using iperf and TCP) to arrive at accurate measurements. Assessing the throughput at "steady state", or when performance have been stabilized, is critical for the soundness of any measurement, hence it is important to discover the shortest TCP connection that delivers results consistently. Thereupon, we measure a set of 50 TMs by deploying TCP flows for different time periods, namely, 1, 2, 5, 10, 15 and 30 s, and search for the point that overall throughput stabilizes. We repeat the experiment for 4 different sizes of TMs and plot the results in Fig 2. The results reveal that throughput converges after a 3 s measurement time frame regardless of the TM size, thus suggesting that 3 s is the shortest connection to measure transmission rate consistently.

To further investigate the consistency of measurements, we explore the Relative Standard Deviation (RSD), i.e., the standard deviation divided by the mean, of link and TM throughputs; a TM throughput is the sum of the throughputs of the links in the TM. We generate 20 random TMs of three different sizes, namely, 1, 5 and 10 links, and we repeat the measurement 10 times. Fig. 3 presents the results for measurements of different time periods, namely, 1, 2, 5, 10, 15 and 30 s. First, we observe that the link RSD increases as TM size increases regardless of connection duration, while the TM RSD is not significantly affected, thus revealing that the overall TM performance can be consistently estimated but the resources allocated to the individual TCP flows are more random. Second, for single-link TMs, a longer connection duration delivers more consistent performance reports, and indeed the RSD of 1 s connections present far greater RSD than longer connections. Third, a striking outcome of this experiment is that the link RSD is remarkably large, and does not decrease at all with the duration of the experiment, for larger-sizeds TMs. This suggests that the link throughputs are not ergodic, in the sense that time averages over long measurements are not equal to ensemble averages taken over multiple measurements. Similarly to the previous finding, we consider 3 s the "sweet spot" between consistent and fast measurements.

B. Effect of wireless channel

We investigate the impact of the used frequency band on data plane performance of the testbed. The selection of the channel in the testbed is direct, through the network configuration file of the Raspberry Pis, namely, file */etc/network/interfaces*. We compare channels 10 and 40 at

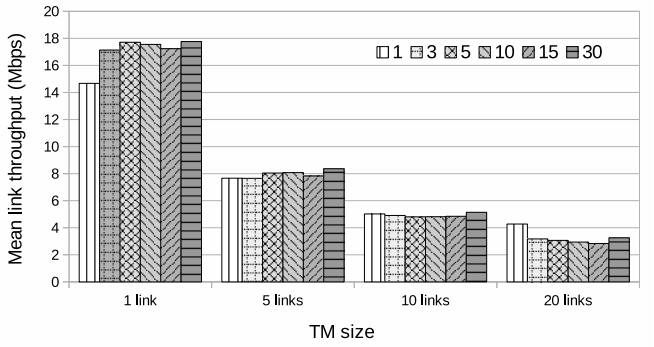

Figure 2. Mean link throughput of TMs for connections with different duration, namely, 1, 2, 5, 10, 15 and 30 s.

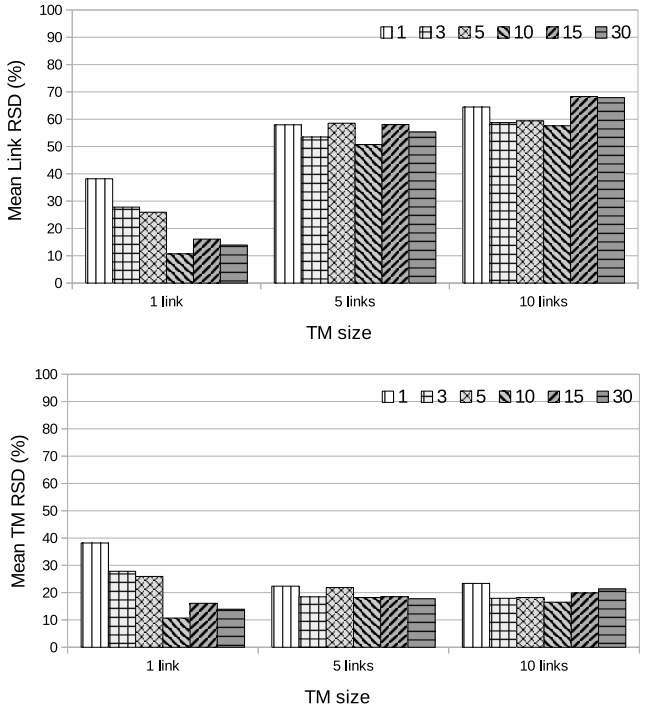

Figure 3. Mean link and TM RSD for connections with different duration, namely, 1, 2, 5, 10, 15 and 30 s.

2.4 and 5.2 GHz, respectively, since they are the least-used channels near our testbed. In Fig. 4, we plot the connected pairs of nodes for the two cases. As expected, the connectivity of the network is penalized by using a higher transmission frequency, due to the larger associated signal attenuation, thus resulting in a less dense topology.

We also explore how frequency selection can affect the mean and RSD of link throughputs. We generate 100 random TMs for three TM sizes (1, 5 and 10 links) and we repeat the measurement of each TM 10 times, using 3 s TCP flows at 2.4 and and 5.2 GHz. Fig. 5 presents the results normalized to the maximum measured value, which is 43 Mbps and 66% for throughput mean and RSD, respectively. The figure suggests that the 5.2 GHz band offers a significantly higher throughput with substantially less variability. The RSD reduction observed at the 5.2 GHz band becomes less apparent as the TM size

ACCEPTED IN WOWMOM 2023

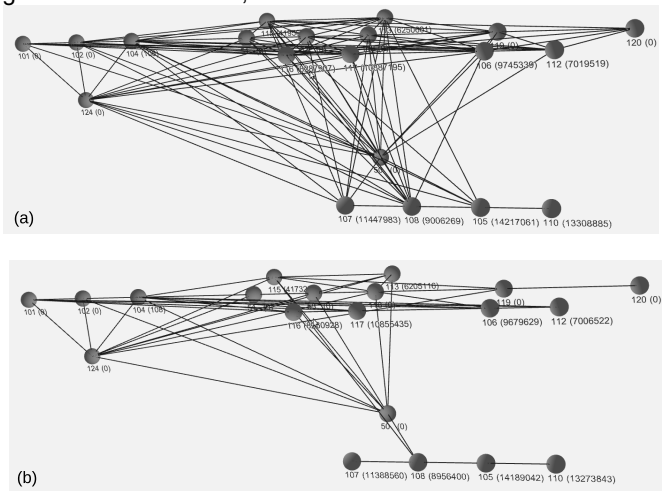

Figure 4. Network connectivity using different Wi-Fi channels, namely, 10 at 2.4 GHz and 40 at 5.2 GHz (subfigures *a* and *b*, respectively). Spheres and lines denote nodes and reachable pairs, respectively.

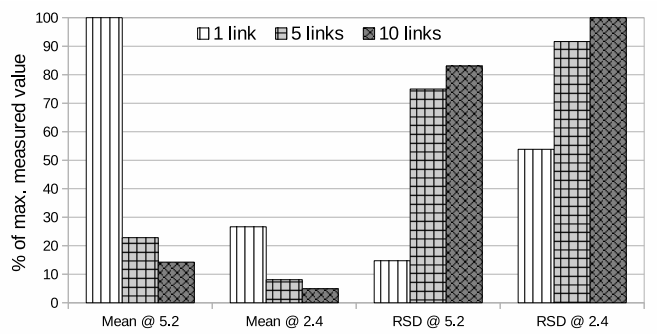

Figure 5. Mean and RSD of link throughput for TMs of different sizes at 2.4 GHz and 5.2 GHz. Results are normalized to the maximum measured value (43 Mbps and 66%, respectively).

grows, albeit still significant for TMs of 10 links (roughly 17% reduction of RSD). Finally, RSD increases with TM size regardless of the channel, unveiling again that flow competition challenges the convergence of individual link throughputs.

C. Binarity of interference

Interference between links is often taken to be binary, i.e., two links either do not interfere at all, or interfere so much that only one of them can be active at any time. This assumption has often been adopted in network models, as it allows interference to be modeled in terms of a graph [8].

An interesting question, therefore, is whether interference is really binary in our testbed. To that effect, we have performed the following experiment in a network comprised of 16 nodes communicating over a 2.4 GHz channel with the RTS/CTS mechanism activated. We found all links capable of transmitting with a rate of at least 3 Mbps when not interfered with, and, for each pair of such links, we plotted a point in the scatter plot of Fig. 6, at the location

$$\left(\frac{R_{i,j}}{R_i}, \frac{R_{j,i}}{R_j}\right),$$

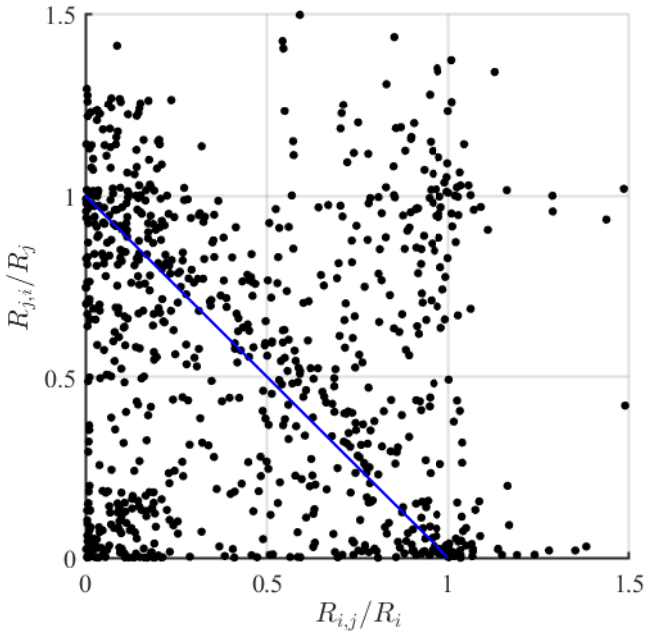

Figure 6. Interference between pairs of links.

where R_i and R_j are the throughputs of links i and j respectively, when they are the only active ones, and $R_{i,j}$ and $R_{j,i}$ are the throughputs of the links of i and j respectively when both are active, and no other link is active. All transmissions were for 15 s. In order to keep the number of points on a manageable level, we only considered links that do not share nodes, and for each receiver its own transmitter is closer than the interfering transmitter. The scatter plot contains 864 points, of which 30 are outside the plotted area $[0, 1.5] \times [0, 1.5]$.

A number of comments are in order. First, note that many of the points lie outside the Cartesian product $[0, 1] \times [0, 1]$, due to variability of the measurements, consistent with the findings of Section IV-A. Second, note that if IEEE 802.11 was capable of performing perfect time division, there would have been no points below the line connecting points $(0, 1)$ and $(1, 0)$. However, numerous points are, in fact, below that line, suggesting a very poor time division; in fact, there is a concentration around the point $(0, 0)$, suggesting that in numerous instances IEEE 802.11 failed altogether. Third, there are numerous points concentrated around the points $(1, 0)$ and $(0, 1)$ corresponding to cases where a strong link completely dominates a weaker one. Fourth, there is also a concentration of points around the point $(1, 1)$, corresponding to non-interfering links. Finally, there are quite a few points uniformly spread in the triangle between the points $(1, 0)$, $(0, 1)$, and $(1, 1)$; all these correspond to links that interfere partially. Therefore, we can state that, in this setup, and contrary to the common assumption, interference is *not* binary.

D. Effect of RTS/CTS

The Request To Send / Clear To Send (RTS/CTS) mechanism of IEEE 802.11 can influence network performance

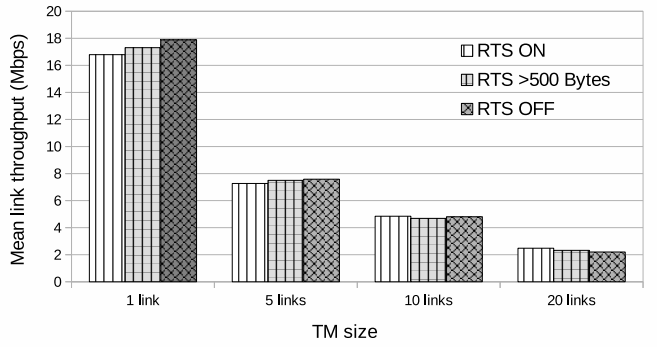

Figure 7. Mean link throughput of TMs of different sizes using RTS/CTS with different thresholds.

especially in dense wireless networks, since it can reduce frame collisions in case of hidden terminals, but also cause some performance degradation, in the opposite case, due to the exchange of additional control packets. By default, the mechanism is disabled in the network nodes of our testbed, but its activation is straightforward using the *iwconfig* configuration tool.[12] Fig. 7 depicts the mean link throughput of TMs of different sizes for different packet size thresholds on which the RTS/CTS mechanism is activated. The results show that the mechanism can be detrimental for performance when flow competition is weak (TM size 1, 5) but starts offering gains when many links are simultaneously active (TM size 10, 20).

We also investigate the impact of the mechanism on measurement consistency by taking 20 consecutive throughput measurements of randomly generated TMs of different sizes with and without RTS/CTS and measuring the RSD and average of the link throughputs; the results offer similar conclusions with results from previous experiments, hence we refrain from discussing all data, and instead focus on the most affected (by RTS/CTS) TM size, single-link TMs. Table II presents the results of 10^4 TM measurements. TMs are categorized in four groups: all TMs, TMs with RSD less than 10%, 10-20% and greater than 20%. We can, therefore, explore separately the performance of the most, the moderately and the least consistent TMs, respectively. First, the results indicate that RTS/CTS reduces RSD by roughly 10% in all measurements, and thus can significantly enhance the consistency of measurements. Second, the greatest improvement is presented at the least consistent TMs where RSD is reduced to 38% from 55%. It is worth indicating that the least consistent TMs constitute roughly the 33% of the TMs in this experiment and present the lowest transmission rate, achieving roughly the 68% of the mean performance measured of all TMs, thus revealing a correlation between low throughput and high RSD.

E. Effect of TCP flavors

The throughput of any TCP connection depends on the congestion control algorithm that is used. Currently, CUBIC TCP [16] is the default algorithm for Linux-based nodes, however algorithms that are specifically tailored for wireless

[12]https://linux.die.net/man/8/iwconfig

	RTS/ CTS	RSD $0-100\%$	RSD $<10\%$	RSD $10-20\%$	RSD $>20\%$
TM number (%)	ON	100	48.9	19	31.9
	OFF	100	38.4	22.6	38.8
Average RSD (%)	ON	16.4	4	11.3	38.3
	OFF	26.1	4.7	11.9	55.4
Norm. (%) Throughput	ON	100	124.9	80.1	67.8
	OFF	100	129.5	90.4	69.1

Table II
RSD AND THROUGHPUT WITH/WITHOUT RTS/CTS.

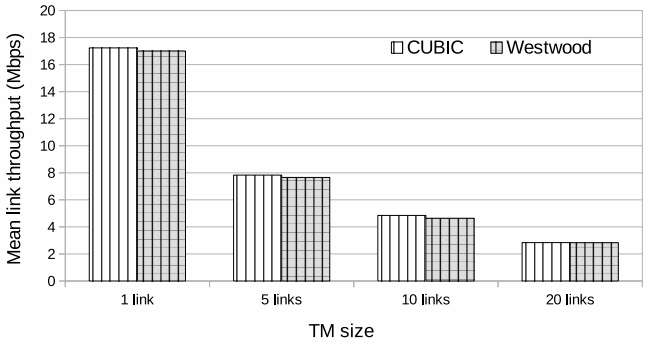

Figure 8. Mean link throughput of TMs of different sizes using CUBIC and Westwood congestion control algorithms.

mediums, such as Westwood [17], can have a measurable impact on the network performance of the testbed. Thereafter, we compare the CUBIC and WestWood algorithms in our testbed by measuring the same TMs and present the result in Fig. 8. The figure does not reveal a significantly performance advantage of any algorithm regardless of TM size.

F. Testbed isolation

Placed at an AUEB building where uncontrolled IEEE 802.11 networks are available to students and personnel, the data plane network of the testbed may potentially suffer from interference by those networks that operate in overlapping frequency bands. A constant interference is not considered concerning since it implicitly shapes the testbed characteristics but, being static, it does not invalidate the measurements. On the other hand, a temporarily dynamic interference can lead to inconsistent measurements and, in turn, invalid conclusions. Therefore, we study the performance of all network links throughout a week, focusing on three different time zones: working days and hours (Monday to Friday 9:00-21:00), when the building is open for students, off-hours (Monday to Friday 22:00-8:00), when the building is empty, and weekends, when the building is very thinly populated. During a 7-days period, we periodically discover active links and measure their throughput (every 75 minutes). The results are presented in Fig. 9, where we plot the standard deviation and the mean throughput of the 230 links with the highest availability; the links are plotted in ascending order based on their performance to make the results easier to read. Although we observe that the throughput and resilience during the weekends is slightly better, the differences are considered too small to

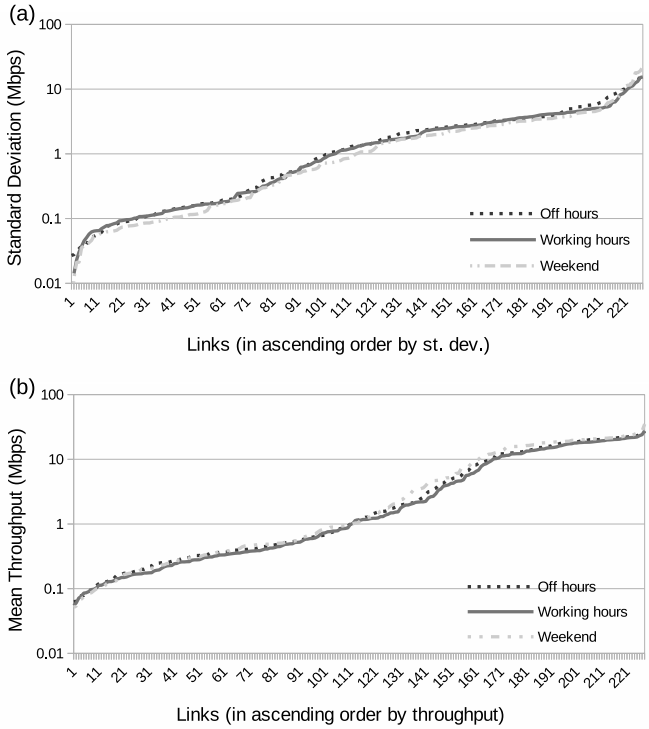

Figure 9. Standard deviation of link throughput during campus working hours (9:00-21:00), off hours (22:00-8:00) and the weekend.

make a difference, thus indicating that the testbed performs consistently in time.

V. Challenges

In this section, we discuss some important challenges that have affected the design and operation of the testbed, and have been addressed in varying degrees; more thoroughly addressing these challenges, in order to expand the capabilities of the testbed, is part of future work.

A. Isolating the data network

The results of our preliminary evaluation (Section IV-F) show that the performance of the testbed is relatively consistent between different days or different parts of the day, hence the impact of potentially interfering transmissions is not apparent. Therefore, the conduction of the same experiment is expected to produce the same result regardless of the operation time frame. Nevertheless, given that the surrounding environment is uncontrolled, a periodic automated check of the surrounding conditions is needed.

B. Synchronizing link activation

As discussed in Section III-A, achieving instant deployment of multiple links in different parts of the network is a very hard problem. Indeed, the centralized nature of the controller is susceptible to an inherent bias present on the control network locations of nodes; even if commands are emitted instantly from the controller, the nodes that receive information from the controller faster (in the control network) will also start their transmissions faster. We have explored this bias by sending ICMP packets from the controller to the nodes and found that the least and maximum *round trip* latency are 3.2 and 8.2 ms, respectively, and so the resulting maximum *one-way* deployment advantage can be roughly $(8.2-3.2)/2 = 2.5$ ms. Although this value can be devastating when operating in the order of microseconds, it is far less significant when measuring link performance in the order of seconds.

To improve synchronization, we could resort to a distributed design wherein the schedule of link activations is delivered, through the control channel, to the network nodes (who would have to be equipped with accurate synchronized clocks) that then perform independently the actions prescribed for them by the schedule.

This distributed approach fits better with deployments in which transmission durations are on the order of microseconds and the communication latency of the network should be below that level. Indeed, related work suggests that TDMA can be enabled with commodity hardware [18], [19]. In a purely centralized design of TDMA, where the controller directs the nodes in real-time by sending explicit commands, the process seems impractical considering current hardware. However, in a hybrid design, where the controller directs the nodes pro-actively, that is, sends a bundle of time slots along with their execution times and rests its confidence on nodes to distributively get synchronized and execute the received commands in the scheduled time, the process is very likely to deliver better results.

C. Adding passive measurements

In this paper, we consider *active* measurements where traffic stream throughputs are measured by deploying dedicated streams through a central controller, instead of *passive* measurements where the throughputs of traffic streams created not by the controller, but by other applications, are measured opportunistically. Expanding the testbed in this direction is feasible, but would require the development of a centralized monitoring tool akin to our controller. While the passive approach presents no temporal overhead, the active approach offers a direct and more controlled assessment of link characteristics, which might be a prime requirement for the operator. Nevertheless, we expect that a hybrid measurement pattern that exploits both techniques is expected to offer the most gains, therefore is considered an important area for future work.

D. Handling errors

When experimenting with real software on real, inexpensive devices, it is inherently expected to face failures, due, e.g., to bugs and malfunctioning. In particular, during the development of the testbed, erroneous TCP connections were frequent. Four different errors were documented, each arising from a different cause: when the server node is not reachable, iperf would return “No route to host” message; when iperf failed to establish a connection, iperf would report “Operation now in progress” message; when one iperf application did not start due to control-plane failure, the iperf log would

be empty; finally, under unknown conditions, iperf would report spurious results (exceeding 70 Mbps). All these issues are automatically addressed by the controller who rejects the measurement and repeated it.

E. Wired control plane

Connecting the controller and the nodes through wired connections is an obvious measure to reduce setup latency (Section IV-A). Nevertheless, the use of wired connections severely penalizes the portability of the testbed, complicating topology changes and challenging the testbed's support of node mobility.

VI. Conclusions and future work

We have introduced TWIST, a testbed specifically built for supporting research on the interference experienced by competing wireless streams and, as a special case, the estimation of capacity regions in wireless networks.

In our evaluation of the testbed we assess thoroughly the temporal overhead of taking measurements. Notably, TWIST requires only 5.4 s for an accurate 10-link TM measurement (2.4 s overhead and 3 s measurement period to get credible transmission rate assessment), which correspond to around 90 minutes to exhaustively measure every TM in a 10-link topology. However, repeated measurements of the same TM display significant variability, especially when the TM involves multiple competing links, the RTS/CTS mechanism is deactivated, and the channel used is in the 2.4 GHz band, even when the measurement lasts much longer (see Fig. 3); this variability is not due to how we take measurements, but a feature of the PHY/MAC layer used in the testbed, which should be taken into account when designing protocols. Other important findings are that the interference in the testbed is not binary, in many cases IEEE 802.11 fails to prevent collisions, even in the case of two links with the RTS/CTS mechanism activated, and that the CUBIC and Westwood flavors of TCP perform very similarly.

The thin-waist architecture of TWIST allows its straightforward usage in addressing a variety of research problems. Other researchers can use the testbed remotely, or can make use of the extended data sets that have been created by the testbed, or can adopt its thin-waist architecture in their testbeds.

Various avenues for future work are open, some already discussed. Most importantly, first, it is clear that passive measurements can accelerate the measurement process, however their accuracy and their use together with active measurements needs to be further investigated. Second, the hybrid operation where the controller pro-actively sends a TM deployment schedule and the clock-synchronized nodes materialize it distributively is expected to reduce the setup latency of the testbed, however it remains to be answered whether such a scheme can enable a microsecond-level TDMA traffic schedule. Finally, alternative protocols may be implemented in the MAC/PHY layer, notably IEEE 802.11ax; in this case, a particular topic to investigate is the extend to which it achieves its main goal of enhancing the throughput-per-area [20].

Acknowledgment

The research project was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "1st Call for H.F.R.I. Research Projects to support Faculty Members & Researchers and the Procurement of high-cost research equipment grant" (Project Number: HFRI-FM17-352, Project Title: Wireless Mobile Delay-Tolerant Network Analysis and Experimentation, Project acronym: LEMONADE).

References

[1] Y. Thomas, N. Smyrnioudis, and S. Toumpis, "Experimental measurement of the capacity region of wireless networks," in *Proc. WiOpt*, 2021, pp. 1–8.
[2] Y. Thomas, S. Toumpis, and N. Smyrnioudis, "Estimating and utilizing wireless network capacity regions," 2022, submitted for publication, https://mm.aueb.gr/lemonade/papers/2022CRsubmission.pdf.
[3] S. Sharma, S. Urumkar, G. Fontanesi, B. Ramamurthy, and A. Nag, "Future wireless networking experiments escaping simulations," *Future Internet*, vol. 14, no. 4, p. 120, 2022.
[4] T. R. Newman, A. He, J. Gaeddert, B. Hilburn, T. Bose, and J. H. Reed, "Virginia tech cognitive radio network testbed and open source cognitive radio framework," in *PRoc. TRIDENTCOM*, 2009, pp. 1–3.
[5] M. Danieletto, G. Quer, R. R. Rao, and M. Zorzi, "CARMEN: a cognitive networking testbed on android os devices," *IEEE Communications Magazine*, vol. 52, no. 9, pp. 98–107, 2014.
[6] S. Bouckaert, W. Vandenberghe, B. Jooris, I. Moerman, and P. Demeester, "The w-iLab.t testbed," in *Proc. Tridentcom*, 2010, pp. 145–154.
[7] P. Bonte, F. Ongenae, J. Nelis, T. Vanhove, and F. De Turck, "User-friendly and scalable platform for the design of intelligent IoT services: a smart office use case," in *Proc. ISWC*, 2016, pp. 1–4.
[8] T. Salonidis, G. Sotiropoulos, R. Guerin, and R. Govindan, "Online optimization of 802.11 mesh networks," in *Proc. CoNEXT*, 2009, pp. 61–72.
[9] J. Struye, B. Braem, S. Latré, and J. Marquez-Barja, "The citylab testbed—large-scale multi-technology wireless experimentation in a city environment: Neural network-based interference prediction in a smart city," in *Proc. IEEE INFOCOM (WKSHPS)*, 2018, pp. 529–534.
[10] D. Raychaudhuri, I. Seskar, G. Zussman, T. Korakis, D. Kilper, T. Chen, J. Kolodziejski, M. Sherman, Z. Kostic, X. Gu *et al.*, "Challenge: COSMOS: A city-scale programmable testbed for experimentation with advanced wireless," in *Proc. MOBICOM*, 2020, pp. 1–13.
[11] Z. Zhang, "ZCNET: Achieving high capacity in low power wide area networks," in *Proc. MASS*, 2020, pp. 702–710.
[12] U. Bhattacherjee, E. Ozturk, O. Ozdemir, I. Guvenc, M. L. Sichitiu, and H. Dai, "Experimental study of outdoor UAV localization and tracking using passive RF sensing," in *Proc. WINTECH*, 2022, pp. 31–38.
[13] E. Aliaj, G. Dimaki, P. Getsopoulos, Y. Thomas, N. Fotiou, S. Toumpis, I. Koutsopoulos, V. Siris, and G. C. Polyzos, "A platform for wireless maritime networking experimentation," in *Proc. GIIS*, 2018, pp. 1–6.
[14] N. Papadis, S. Borst, A. Walid, M. Grissa, and L. Tassiulas, "Stochastic models and wide-area network measurements for blockchain design and analysis," in *Proc. IEEE INFOCOM*. IEEE, 2018, pp. 2546–2554.
[15] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Select. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, 2019.
[16] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.
[17] M. Gerla, M. Y. Sanadidi, R. Wang, A. Zanella, C. Casetti, and S. Mascolo, "TCP westwood: Congestion window control using bandwidth estimation," in *Proc. IEEE GLOBECOM*, vol. 3, 2001, pp. 1698–1702.
[18] Z. Yang, J. Zhang, K. Tan, Q. Zhang, and Y. Zhang, "Enabling TDMA for today's wireless LANs," in *Proc. INFOCOM*, 2015, pp. 1436–1444.
[19] S. Zehl, A. Zubow, and A. Wolisz, "hmac: Enabling hybrid TDMA/CSMA on IEEE 802.11 hardware," *arXiv preprint 1611.05376*, 2016.
[20] E. Khorov, A. Kiryanov, A. Lyakhov, and G. Bianchi, "A tutorial on IEEE 802.11ax high efficiency WLANs," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 197–216, 2019.