On Software Tools and Stack Architectures for Wireless Network Experiments

A. S. Abdallah*, A. B. MacKenzie*, L. A. DaSilva*†, and M. S. Thompson**
* Wireless @ Virginia Tech
{yarab, mackenab, ldasilva}@vt.edu
†CTVR, Trinity College Dublin
** Electrical Engineering Department, Bucknell University
michael.thompson@bucknell.edu

***Abstract*—Simulation is still the most widely adopted performance evaluation technique in mobile ad hoc network research, in spite of a growing number of questions about the fidelity of this technique. Implementation-based testing and evaluation of wireless networks tends to produce believable results, but the technique sometimes suffers from poor repeatability, high implementation cost, and complex experimental logistics. The topic of this paper is software tools to enable implementation-based experimental research on wireless networks. We review some of the existing tools and propose the Flexible Internetwork Stack (FINS) framework, our open-source solution for network protocol implementation, integration, and testing. FINS aims to provide researchers with monitoring, logging, and reconfiguring utilities similar to the ones provided by simulation environments or emulation testbeds.** [1]

I. Introduction

Various software tools are required to enable implementation-based testing and evaluation of wireless network protocols. Such tools have been developed by individual researchers over the years, although their impact on research community is still relatively small. This is in contrast to simulation tools, like ns-2 or OPNET Modeler, which tend to be used broadly across research institutions. Some researchers have gone as far as proposing new architectures for the network protocol stack, flexible enough to support implementation-based experiments and fast prototyping of network protocols.

In this paper, we introduce an architectural taxonomy and functional classification of software tools and new network architectures that support implementation-based experiments across the network stack. We focus on tools and architectures that provide dynamic and re-configurable features to conduct network experiments, in particular in mobile ad hoc networks (MANETs). We also discuss some of the principal design challenges, concepts, and requirements of future software tools to enable wireless network experimental work. Finally, we propose the Flexible Internetwork Stack (FINS) Framework to address these challenges, and we discuss its design requirements and architecture.

A number of research studies have employed implementation-based experiments on MANETs, as surveyed in [1]. However, the vast majority of MANET research still relies primarily on simulation. Simulation-based studies face questions about their fidelity [2]. An alternative is emulation, a technique which combines simulation and implementation. In [1], an emulator is defined as a set of tools which allow the user to imitate a layer or more from the network stack in a simulation environment while running the remaining layers as real implemented systems. Emulation testbeds have many advantages but they inherit some of the disadvantages of simulation environments [3].

Implementation-based experiments run a fully functioning network stack on top of a real world platform such as a laptop or a handheld device. The MANIAC competition [4] and the study in [5] are examples of implementation-based experiments. Implementation-based experiments on wireless networks face a set of logistical, evaluation, and implementation challenges. Logistical challenges include the need for a significant number of people, equipments and spaces. Evaluation challenges include the lack of repeatability and the lack of standard benchmarking scenarios. Implementation challenges include the lack of experimental software tools, the different ways the protocol stack is implemented across different platforms, as well as the high cost of implementation. This paper discusses software tools that enable experimental research in emerging wireless networks. In particular, we describe the development of a framework, FINS, that provides the researchers with monitoring, logging, and reconfiguring utilities similar to the ones provided by simulation environments or emulation testbeds.

II. Software Tools and Hardware Devices for Wireless Network Experimentation

We begin by providing two different taxonomies for software tools and architectures which enable wireless network experiments. Our classification covers software frameworks, application programming interfaces (APIs) and libraries, as well as new architectures for the network protocol stack. The term *experimental wireless network tools* will be used for the rest of this paper to refer to all of these. Then, we discuss the use of mobile devices for wireless network experiments.

A. Software Tools

Our first classification scheme is based on the architectural concepts adopted. Experimental wireless network tools can be divided into *clean slate tools* and *hybrid tools*. In *clean slate tools*, researchers propose an innovative architecture for the

[1]This material is based upon work supported by the National Science Foundation under Grant Nos. 0916300 (Virginia Tech) and 0916283 (Bucknell University).

network protocol stack. These architectural ideas aim to avoid the disadvantages of the layered architectures of the TCP/IP stack while retaining some degree of modularity. Projects such as Autonomic Network Architecture (ANA) [6], and CellNet framework [7] are examples of clean slate tools.

In contrast, *hybrid tools* modify or combine features of the layered stack with some clean slate features. Such modifications allow the implementation of cross-layer solutions and inter-communication between any number of non-adjacent layers. X-kernel [8], Iris [9], X-layer [10], OpenOnload [11], and XL-interface [12] are examples of hybrid tools.

Our second classification scheme classifies the *experimental wireless network tools* into four classes with respect to their functional purposes. The first class contains tools which aim to facilitate the testing and development of new protocols. For example, tools such as CLICK [13], ProtoLib [14], and Profab [15] provide researchers with a library of basic and intermediate level functions. These functions can be used to implement larger modules such as a protocol or even an entire layer. Some tools accelerate the development cycle by providing an API which is used to implement a virtual platform-independent layer. This virtual layer can be used to run experimental protocols on top of a real stack, sacrificing some performance for the sake of a unified implementation of the experimental protocols regardless of the platform. Examples of platform-independent APIs include VIPE [16] and GEA [17].

The second class contains the tools which provide generic interfaces to interact with the existing layers and protocols. These APIs provide a standard interface for the flow of data between layers, as well as reading or tuning parameters within a protocol or layer during run time. They work as a kind of standard wrapper to access pre-implemented existing protocols or layers during an experiment. Examples of such tools include ULLA [18], DEC [19], Universal Convergence Layer (UCL) [20], and XIAN [21].

The third class contains tools which provide managerial features over a set of nodes. These tools give the user the ability to batch experimental scenarios, broadcast the scenarios among nodes, redirect the movement of nodes during the experiments, and reconfigure network topologies. The goal of these tools is to give researchers the same flexibility and usability during implementation-based experiments as provided by a typical network simulation tool. APE [22] and MTM [23] are examples of this class.

The last class covers the tools which provide monitoring and logging whether they adopt a centralized or distributed approach. Monitoring and logging tasks include trace collection, trace aggregation, topology visualization, and data filtering. Examples of this class include MMAN [24], APE View [25], and Promox [26].

B. Mobile Devices

Mobile devices, including smartphones, PDAs, and tablets are a natural fit for experimental networking research. Major reasons that mobile devices are well suited for networking research, especially when mobility is required, include:

- Small Size: Mobile devices are easier to transport and store than even the smallest laptops. Additionally, it is simple for a person to carry around a device.
- Low Power: Mobile devices are designed to have a battery life that is measured in days rather than hours. This means that experimenters can run longer experiments.
- Simple Interface: Most mobile devices include touch screen interfaces which can be used while moving (unlike laptop keyboards which are difficult to use while moving).
- Low Cost: Most mobile devices cost $500 or less, meaning more devices can be purchased for the same amount of money.
- Communication Capabilities: Most mobile devices include cellular connectivity, GPS receivers, IEEE 802.11, and Bluetooth. Hence they offer more communication capabilities than are found on most current laptops.

The major drawback of mobile devices is limited resources. Mobile processors are less computationally-capable than desktop and laptop processors. This is acceptable for many networking experiments which are not computationally demanding. Additionally, mobile devices have limited storage space. The current storage medium of choice is the microSD card, offering up to 16 and 32 GB of space. Depending on the experiment, this may not be enough storage space.

III. Design Challenges, Concepts, and Requirements

In this section we discuss some design considerations, implementation concerns, and functional requirements for future *experimental wireless network tools*. Then, we compare a selected set of previous and existing tools that represent different taxonomies, according to which tools fulfill these design concepts and requirements.

Cost of Implementation: Although clean slate tools are attractive from the flexibility point of view, they face a high cost of implementation. Due to the major differences in the structure of the conventional network stack compared to the clean slate architectures, major modifications to (or a completely new implementation of) legacy protocols and applications might be required.

Continuity of Support: The lack of continuity in research projects causes many tools to become obsolete due to the absence of a development and support team to update the tool. As long as the proposed tool has not been made into a commercial product, the only way to maintain continuity is to build a strong open source development community of researchers, developers, and other interested persons around the tool.

Generality: One of the reasons why most of the previous tools have not been deployed more widely is that they are specialized for certain experiments. This reduces the ability of interested researchers to extend the tool to reuse it in other experiments. For example, when a tool provides a customized environment to evaluate the performance of routing protocols in a MANET, it may be difficult to reuse it for a comparison study on transport protocols.

Ease of Use: The availability of comprehensive documentation, ease of installation, ability to self-boot, and the availability of managerial functions within the tool are major criteria to make the tools widely adopted among the research community. Support for various hardware platforms or even different network hardware is also a major criterion that affects the widespread deployment of a tool.

Reusability: The high reusability of a software tool's components, contributes to its success. CLICK [13] is a good example for high reusability. To achieve similar high reusability, future tools should adopt a modular architecture. Also, increasing the granularity of modules increases their reusability. The way the basic elements in CLICK have been used to implement middle level modules, which in turn integrated to implement high level modules such as IPv4 and IPv6, is an example of an highly modular and granular architecture.

Portability: The ability to run the same tool on different platforms makes the tool superior as compared to tools that run on single platform. As mentioned section II-B, mobile devices are currently platforms of great interest for wireless mobile network experiments. Proposing a tool that runs on multiple platforms including mobile devices will help researchers tackle experimental scenarios that are currently challenging given the limited mobility and power constraints of conventional laptops.

Meters and Knobs: Having a generic interface which allows the user to control the behavior of the network protocols during run time, regardless of their internal implementation details, is an important functional requirement for such tools. This control may be imposed directly from the command line, through a GUI window, or even autonomically through a cognitive engine module. This feature, which we refer to as a meters and knobs interface, enables the researchers to implement solutions which depend on bottom up event driven notifications as well as top down dynamic responses. These are sufficient to realize many cross-layer solutions. A meters and knobs interface also provides an initial step for collecting data traces for the logging process discussed below.

Monitoring and Logging: This requirement is related to the previous one. A process should be responsible for collecting traces and metrics, recording these traces with time-stamps, and providing the ability to retrieve this information later. This process may be centralized or distributed among nodes. Out of band communication should be implemented if possible to guarantee minimum influence on the performance of the experimental network. The ability to access time-stamped records during run time is another feature which supports the implementation of cognitive protocols.

Re-configurability of the Network Stack: Dynamic linking and de-linking of modules is important to support experiments within dynamic or cognitive networks. The ability to add modules at run time and to link or de-link previously loaded modules facilitates the implementation of testing and benchmarking scenarios. The clonable network stack project [27] is an example of support for re-configurability. In the clonable network stack, FreeBSD's kernel is modified to enable running multiple independent instances of the network stack concurrently.

TABLE I: Comparison of various experimental network tools with respect to desirable criteria

Criterion	CLICK	ANA	X-kernel	Iris	UCL	APE	MMAN
Cost of implementation	M	H	M	M	L	L	L
Continuity	H	M	N	N	N	N	N
Generality	M	H	M	H	N	L	N
Ease of use	H	M	L	M	N	L	L
Reusability	H	H	M	H	L	L	L
Portability	L	N	N	H	N	N	N
Meters and knobs	L	H	L	M	N	N	N
Monitoring and Logging	L	H	N	M	N	H	H
Re-configurability	N	H	L	M	N	N	N
(H) High (M) Medium (L) Low (N) Not Available							

As shown in Table I, hybrid tools which work only as development tools, such as CLICK [13] and X-kernel [8], have limited support for meters and knobs or monitoring and logging. This is in contrast to hybrid development tools which are equipped with cross-layer interfacing capabilities, such as Iris [9]. The reason is that the cross-layer interfaces permit the user to implement meter reading and knob tuning. Also, the latter type of tool provides the user with a higher level of flexibility and accessibility across the network stack. This combination of functions makes such tools more powerful for development and testing of cross-layer solutions as well as experiments related to cognitive or reconfigurable nodes.

Clean slate architectures represented by ANA in Table I excel in re-configurability, generality, and reusability. But they have the highest implementation cost due to the need to re-implement the whole network stack. They are also not easy to use because of the difficulty in adapting to implement the new non-layered architecture. Due to this burden of adapting to clean slate architectures and the high implementation cost, we suggest that future proposals should focus on hybrid architectures to balance the trade-off between cost and usability. This allows researchers to focus on adding new features to implementation environments rather than reinventing the wheel.

IV. Flexible Internetwork Stack Framework

In this section we propose a new tool called the Flexible Internetwork Stack Framework (FINS Framework). The main objectives, motivations, as well as the system architecture of the FINS Framework are described. We also suggest some use cases and candidate experiments, showing how the FINS Framework facilitates implementing them.

A. Objectives and Motivations

The FINS Framework is a modular software framework which aims to lower the barrier for implementation-based wireless network experiments across the network stack by providing access to functionality that is usually implemented as part of the operating system kernel. The goal of the FINS Framework is to provide researchers with generic modules and interfaces which can be used to:

- build conventional and innovative network stacks using existing or new layers, protocols, and algorithms;
- monitor and log the behavior of the stack components;

- dynamically control the behavior of the network stack during run-time by modifying certain control parameters bound to each module; and
- dynamically load, link and de-link stack components to support a wide range of experimental benchmarking scenarios.

The main motivation for the FINS Framework is to encourage the research community to adopt implementation-based experiments as a trusted technique to evaluate the performance of new network protocols or algorithms. The FINS project is inspired by the MANIAC Challenge experimental competition [28], which gave us better insight into the real performance of the TCP/IP stack in wireless ad hoc networks. For example, we noted significant differences in how ad hoc routing protocols perform in real environments compared to simulation scenarios, pointing to the necessity of implementation-based experiments. Also, the technical challenges faced to implement the MANIAC Challenge API and the logistical challenges of the MANIAC experiments motivate us to present the FINS Framework to the research community.

B. System Architecture, Design, and Implementation

In order to preserve the advantages of previously discussed tools and avoid their pitfalls, the FINS Framework follows the guidelines and requirements discussed in the previous section. the FINS Framework adopts a hybrid architecture. Currently, the FINS Framework maintains the layered structure between the physical and data link layers, while enabling clean slate design of the network, transport and application layers. The FINS Framework takes into consideration two major constraints: lowering implementation cost and forcing no changes to legacy applications. Backward compatibility with legacy applications is not available in most of previous tools. Researchers will be able to run experiments using real data traffic generated by real network applications, instead of modeling the traffic patterns and using artificial traffic generators.

The central component of the FINS framework, as shown in Figure 1, is the FINS Switch. It is responsible for forwarding control and data traffic between the various modules above the data link layer. Instead of moving the data between adjacent layers as in the conventional stack, the data traffic coming in from (or out to) the data link layer will be directed to the switch module, which forwards the data to the appropriate protocol module as specified into the linking table, discussed later. The FINS Framework uses its own frame format which differs based on whether the FINS frame contains conventional protocol data units, control instructions or notifications. In order to maintain backward compatibility with legacy applications, the FINS Framework retains the conventional applications socket interface. A stub module, the socket stub, is used to redirect the traffic coming down from (or up to) the application to the switch and vice versa. A data link stub is responsible for capturing the incoming traffic and injecting the outgoing traffic from/to the data link layer. Both stubs make sure that the FINS Framework is transparent to the applications and the layers below the network layer.

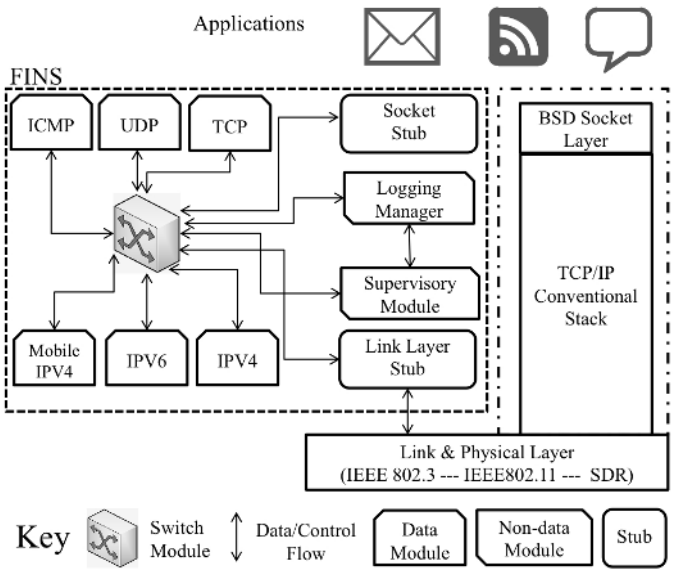

Fig. 1: FINS System Architecture Vs. Conventional Stack

FINS modules (excluding the switch) are classified into two categories: data and non-data modules. Data modules are modules which implement the functional behavior of a network protocol or algorithm, such as IPv4, TCP, ICMP, and ARP. Protocol modules can be also programmed to send statistics periodically to another interested module through the switch. A non-data module is a module which contributes toward the control or managerial tasks of the FINS Framework. For example, a logging module is programmed to collect various metrics about the network stack. This is done through the control API provided as part of FINS. The supervisory module, another non-data module, behaves as a master module. It has the capability to control the FINS framework including the switch module itself. The supervisory module can send control instructions to any of the modules linked to the switch. It can also change the path the data travels among the modules by reconfiguring the linking table which the switch module maintains for forwarding FINS frames. The switch module broadcasts a copy of the linking table to other modules whenever an update takes place. Other modules use their local cached copies to determine the appropriate ID(s) of the destination module(s)to fill into FINS frames, while the switch only forwards the frames to the destination(s).

The initial version of the FINS Framework will include modules for IPv4, UDP, TCP, socket stub and Ethernet stub. These basic modules are provided to significantly reduce the overhead and development cost of using the FINS Framework as compared to other tools. The FINS Framework provides a control interface which implements the meters and knobs feature for this set of modules and stubs. Table II shows a list of example meters and knobs. This set of meters and knobs enables FINS users to implement and evaluate cross-layer solutions.

The FINS Framework uses C to implement all its components and to integrate them with the 2.6 Linux kernel. The 2.6 kernel running on regular laptops equipped with standard

TABLE II: FINS Meters and Knobs

Meters	Knobs
End-to-end metrics (loss, latency, jitter)	Application behavior
Packet loss notifications Retransmission information Acknowledgement observation	TCP window size Retransmission strategy
Available routes and neighbors ICMP information	Per-packet next-hop selection Routing-table modification IP fragmentation options
Reachable neighbors Frame error rate Available access points Beacon information	Transmit power Physical data rate Retransmit behavior RTS/CTS

802.11 wireless interfaces has been chosen as the initial target platform. FINS modules are implemented in user space, which provides many advantages over the traditional network stack such as the high level of flexibility, ease of use, reusability, and reduction of the implementation cost. It also allows future use of high level programming languages (such as C++ and Java) to implement protocol modules and integrate them with FINS, as long as these programming languages support C APIs.

Due to the importance of portability for a network experimental tool as discussed in Section IV, as well as the promising advantages of using mobile devices as discussed in Section II-B, we plan on releasing the FINS Framework for both traditional laptops and mobile computing devices. We are currently examining the Google Android platform as it is open source and is available on a growing number of devices.

C. Use Cases and Candidate Experiments

Several cross-layer proposals have been presented to improve transport and routing protocols in wireless networks [29], [30]. The challenges associated with implementing these solutions within the conventional stack are the reasons that most of these proposals lived and died in simulation. The authors of [31] provided a classification scheme for cross-layer solutions based on the way the layers exchange information with each other. Figure 2 illustrates how FINS supports implementing these schemes with its flexible architecture and efficient control interface. In Figure 2, the FINS Framework allows the exchange of control information bi-directionally between two adjacent or non-adjacent layers. The FINS Framework supports "vertical calibration" by allowing to implement a calibration manager module, which reads the appropriate meters and turns the appropriate knobs across the stack. Layers can be merged by implementing a module which combines the functionality of two or more layers together. For example, to experiment with merging TCP and IP to seek better performance, the FINS Framework allows to code a module which behaves as a combined TCP-IP module.

In the context of disruption-tolerant networking, as shown in Figure 3, the bundle protocol is implemented as a single module, instead of two layers as suggested in its RFC [32]. The conventional stack requires a convergence layer adapter between the bundle protocol agent and the traditional TCP layer. The agent takes care of the main protocol functions, while the adapter interfaces the agent to TCP. With the FINS Framework, the overhead of implementing an adapting layer does not exist.

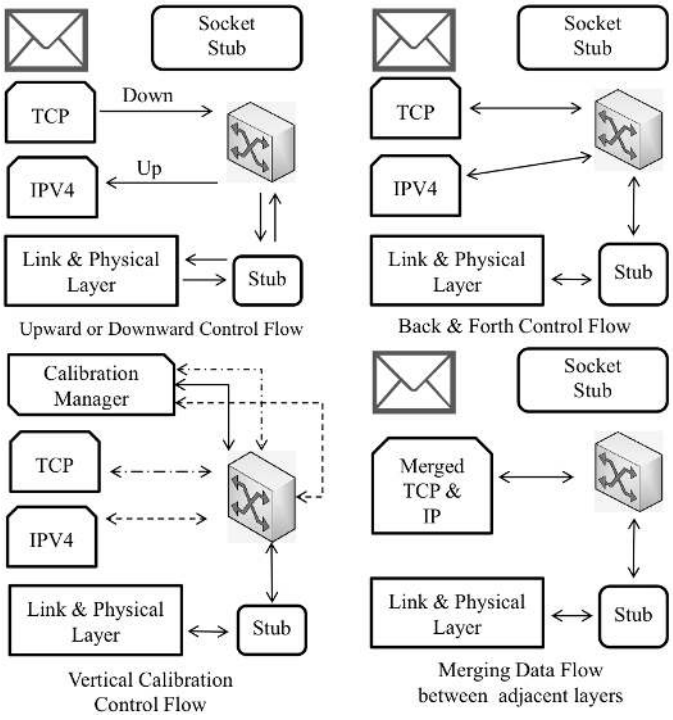

Fig. 2: Cross-layer designs using FINS

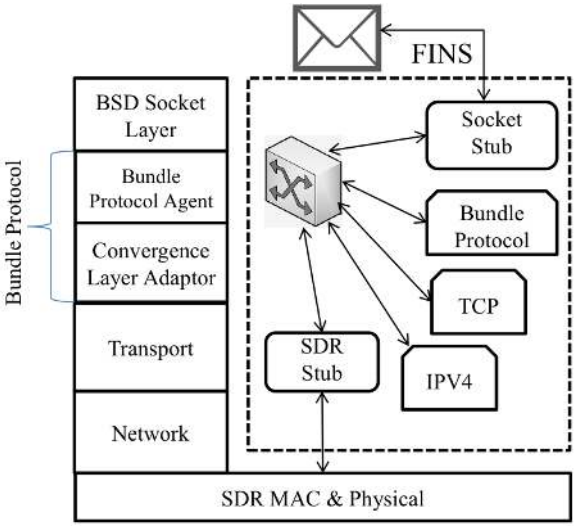

Fig. 3: Bundle protocol implementation using FINS vs traditional stack

The FINS Framework user can also utilize the supported cross-layer schemes to implement new context-aware applications that collect information from lower layers and adapt their behavior to match current conditions. The application can be implemented as a module that talks directly to the switch, rather than through the socket stub, collects meters from other layers and sets its own knobs.

The FINS Framework can also be used for the implementation of a cognitive protocol stack, where a cognitive engine uses the meters and knobs feature to learn from the current and previous conditions of the network node as well as the local network neighborhood [33]. The engine learns, then makes decisions to adapt and sends control instructions to

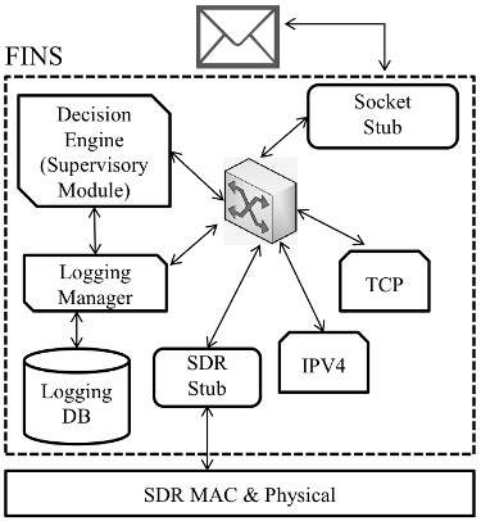

Fig. 4: A cognitive node implementation using FINS

the modules. Features for run-time structural reconfigurations, such as linking and de-linking modules, are supported in the FINS Framework architecture as previously mentioned. Figure 4 illustrates how the FINS Framework can be used to implement a cognitive engine running on an SDR platform. What is needed is to replace the data link stub with a new stub to interface the FINS Framework switch with the SDR modules.

V. Conclusion

The implementation and testing of new protocols and network resource management solutions may be costly (in time and effort) and present logistical challenges, especially in medium- and large-scale network experiments. In this paper, we proposed a new tool, the flexible internetwork stack (FINS) framework, which addresses some of these challenges by making available a full protocol implementation, from a socket stub to a MAC/PHY stub (using IEEE 802.11), to be used by researchers engaged in wireless network prototyping and experimental evaluation. Development on the FINS Framework is ongoing, and the current status of this open-source tool is discussed and regularly updated at www.finsframework.org.

References

[1] W. Kiess and M. Mauve, "A survey on real-world implementations of mobile ad-hoc networks," *Ad Hoc Networks*, vol. 5, no. 3, pp. 324 – 339, 2007.

[2] S. Kurkowski, T. Camp, and M. Colagrosso, "MANET simulation studies: The incredibles," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 9, pp. 50–61, 2005.

[3] P. De, A. Raniwala, S. Sharma, and T. Chiueh, "Design considerations for a multihop wireless network testbed," *IEEE Communications Magazine*, vol. 43, no. 10, pp. 102–109, Oct. 2005.

[4] V. Srivastava, A. B. Hilal, M. S. Thompson, J. N. Chattha, A. B. MacKenzie, and L. A. DaSilva, "Characterizing mobile ad hoc networks :- the MANIAC challenge experiment," in *WiNTECH '08: Proceedings of The Third ACM International Workshop on Wireless network testbeds, Experimental Evaluation and Characterization*, 2008, pp. 65–72.

[5] V. Lenders, J. Wagner, S. Heimlicher, M. May, and B. Plattner, "An empirical study of the impact of mobility on link failures in an 802.11 ad hoc network," *IEEE Wireless Communications*, vol. 15, no. 6, pp. 16–21, December 2008.

[6] C. J. Ghazi Bouabene and C. Tschudin, "Virtual network stacks," in *PRESTO Sigcomm Workshop*, 2008.

[7] H. Hassan, R. Eltarras, and M. Eltoweissy, "Towards a framework for evolvable network design," in *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. SpringerLink, 2009, pp. 390–401.

[8] N. C. Hutchinson and L. L. Peterson, "The X-Kernel: An architecture for implementing network protocols," *IEEE Transactions on Software Engineering*, vol. 17, pp. 64–76, 1991.

[9] L. Doyle and et al., "Experiences from the iris testbed in dynamic spectrum access and cognitive radio experimentation," in *IEEE Symposium on New Frontiers in Dynamic Spectrum*, 2010.

[10] S. P. Aaron Beach, Mike Gartrell and R. Han, "X-Layer an experimental implementation of a cross-layer network protocol stack for wireless sensor networks," Department of Computer Science University of Colorado at Boulder, Tech. Rep., December 2008.

[11] "OpenOnload," July 2010. [Online]. Available: http://www.openonload.org/

[12] G. Conti, Marco; Maselli and G. Turi, "Design of a flexible cross-layer interface for Ad Hoc networks," *Challenges in Ad Hoc Networking*, vol. 197, pp. 189–198, 2006.

[13] "CLICK Router," January 2010. [Online]. Available: http://read.cs.ucla.edu/click/

[14] "Protolib," January 2010. [Online]. Available: http://cs.itd.nrl.navy.mil/work/protolib/index.php

[15] S. G. Georg Kunz, Olaf Landsiedel and K. Wehrle, "Protocol factory: Reuse for network experimentation," in *6th USENIX Symposium on Networked Systems Design and Implementation*, 2009.

[16] O. Landsiedel, S. Kunz, Georg, and W. Klaus, "A virtual platform for network experimentation," in *VISA '09: Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, 2009, pp. 45–52.

[17] A. Herms and D. Mahrenholz, "Unified development and deployment of network protocols," in *In: MeshNets*, 2005.

[18] M. Sooriyabandara, T. Farnham, and Wellens, "Unified link layer API: A generic and open API to manage wireless media access," *Computer Communications*, vol. 31, no. 5, pp. 962–979, March 2008.

[19] G. Noubir, W. Qian, B. Thapa, and Y. Wang, "Experimentation-oriented platform for development and evaluation of MANET cross-layer protocols," *Ad Hoc Networks*, vol. 7, no. 2, pp. 443 – 459, 2009.

[20] L. Sánchez, J. Lanza, and L. Muñoz, "Experimental assessment of a cross-layer solution for TCP/IP traffic optimization on heterogeneous personal networking environments," *Personal Wireless Communications*, pp. 284–296, 2006.

[21] H. Aïache, V. Conan, L. Lebrun, J. Leguay, S. Rousseau, and D. Thoumin, "XIAN automated management and nano-protocol to design cross-layer metrics for ad hoc networking," in *NETWORKING Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, 2008, pp. 1–13.

[22] H. Lundgren, D. Lundberg, J. Nielsen, E. Nordstrom, and C. Tschudin, "A large-scale testbed for reproducible ad hoc protocol evaluations," in *IEEE Wireless Communications and Networking Conference, WCNC2002.*, vol. 1, 2002, pp. 412 – 418.

[23] R. Lent, "A testbed validation tool for MANET implementations," in *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2005, pp. 381–388.

[24] H. Kazemi, G. C. Hadjichristofi, and L. A. DaSilva, "MMAN - a monitor for mobile ad hoc networks: Design, implementation and experimental evaluation," in *Proceedings of the Third ACM International Workshop on Wireless Network Testbeds, Experimental evaluation and CHaracterization (WiNTECH)*, 2008.

[25] *A large-scale testbed for reproducible ad hoc protocol evaluations*, vol. 1, 2002.

[26] E. Weingärtner, C. Terwelp, and K. Wehrle, "ProMoX: A protocol stack monitoring framework," Aachen Uni., Germany, Tech. Rep., 2009.

[27] M. Zec, "Implementing a clonable network stack in the FreeBSD kernel," in *Proceedings of the USENIX 2003 Annual Technical Conference*, 2003, pp. 137–150.

[28] "MANIAC Challenge Website," 2010 July. [Online]. Available: http://www.maniacchallenge.org/

[29] T. Goff, N. Abu-ghazaleh, D. Phatak, and R. K. B, "Preemptive routing in ad hoc networks," in *Proc. ACM/IEEE MobiCom*, 2001.

[30] S. Thangam and E. Kirubakaran, "A survey on cross-layer based approach for improving TCP performance in multi hop mobile adhoc networks," in *International Conference on Education Technology and Computer*, 2009, pp. 294–298.

[31] V. Srivastava and M. Motani, "Cross-layer design: a survey and the road ahead," *Communications Magazine, IEEE*, vol. 43, no. 12, pp. 112–119, Dec. 2005.

[32] K. Scott and S. Burleigh, *Bundle Protocol Specification*, Network Working Group, November 2007.

[33] D. Friend, M. EINainay, Y. Shi, and A. MacKenzie, "Architecture and performance of an island genetic algorithm-based cognitive network," in *5th IEEE Consumer Communications and Networking Conference, 2008. CCNC 2008.*, 2008, pp. 993 –997.