Implementation of the AeroRP and AeroNP Protocols in Python

Mohammed J.F. Alenazi, Egemen K. Çetinkaya, and Justin P. Rohrer
Faculty Advisor:
James P.G. Sterbenz
Department of Electrical Engineering & Computer Science
Information & Telecommunication Technology Center
The University of Kansas
Lawrence, KS 66045
{malenazi,ekc,rohrej,jpgs}@ittc.ku.edu

ABSTRACT

The domain-specific ANTP protocol suite consisting of AeroTP, AeroRP, and AeroNP has been developed to cope with the challenges in highly-dynamic airborne telemetry networks. These protocols have been designed and modelled through simulation methodology. In this paper, we present an implementation of the AeroRP and AeroNP components in Python. Initially, we implement and test through an emulated wireless environment on the PlanetLab testbed. Further, we present our prototype implementation that is deployed in a real-world wireless environment using radio-controlled vehicles.

I. INTRODUCTION AND MOTIVATION

Emerging airborne telemetry networks are proposed to cope with the shortcomings of the point-to-point link based telemetry systems. A typical airborne telemetry network consists of three types of nodes: airborne test articles (TAs), relay nodes (RNs), and ground stations (GS). The fast moving TAs send telemetry data to a GS through the RNs, which have a higher power antenna. There are several characteristics of the airborne telemetry environment and this type of network pose unique challenges to end-to-end data communications. The challenges include: high mobility, constrained bandwidth, limited transmission range, and intermittent connectivity. The current TCP/IP-based Internet architecture fails or works poorly in this challenged environment [1, 2]. The Aeronautical Network and Transport Protocol (ANTP) suite [2, 3] is designed to mitigate these challenges and provide edge-to-edge compatibility with the legacy Internet architecture. The suite includes: *AeroTP* – a TCP-friendly transport protocol [4] with multiple reliability and QoS modes, *AeroNP* – an IP-compatible network protocol (addressing and forwarding) [5], and *AeroRP* – a routing protocol [5], which exploits location information to mitigate the short contact times of high-velocity TAs. Using the ns-3 network simulator [6], a model was implemented for each component of the suite and simulated with several scenarios. The simulation results of ANTP protocols showed superior results over other baseline protocols [7, 8, 9, 10].

In our previous work [11], we introduced a preliminary system architecture for ANTP suite implementation. In this paper, we present Python implementation of the aeronautical protocols AeroRP and AeroNP. Moreover, we implement several tools to ease the development and performance analysis phases of the experiments. Finally, we run experiments on emulated and real-time environments and present our implementation results.

The remainder of this paper is organized as follows: Section II gives the background of the ANTP suite and presents past implementations of MANET protocols. Section III presents the system components implementation. Section IV explains the implementation of the system functions. Experiments and results are contained in Section V. Section VI concludes and gives future directions for this work.

II. BACKGROUND

This section gives an overview of the AeroNP and AeroRP protocols found in the ANTP suite, as well as past implementations of MANET protocols.

A. ANTP Suite

The ANTP suite consists of three protocols as mentioned in the introduction; here we will briefly review AeroRP and AeroNP. AeroRP is a geographic routing protocol that can be configured to run on one of three modes: ad-hoc mode, GS-location mode, and GS-topology mode. In addition, It has two parallel phases: neighbor discovery and data forwarding. In neighbor discovery phase, each node advertises its presence using beacons, which have the location and velocity of the node. Once a beacon is received by other nodes, they update their routing table accordingly. For GS-location and GS-topology modes, the routing table is updated with GS update messages. In GS-location mode, the GS broadcasts the current geolocations of all the nodes. In GS-topology mode, the GS broadcasts the current topology by sending the link information among all the nodes in the network. In the data forwarding phase, the nodes forward the packet based on the selected AeroRP mode. In ad-hoc and GS-location modes, the time to intercept (TTI) metric is used to determine the next hop. A node computes the TTI for all of its neighbors and then chooses the neighbor with the minimum TTI value as the next hop. In GS-topology mode, the nodes forward packets to the next hop of the shortest path computed from an intermediate node to the destination [10, 12].

AeroNP is a network protocol designed specifically for the highly-dynamic telemetry environment. There are two AeroNP packet headers: basic and extended. The basic header includes source and destination addresses with other fields to provide other services such as QoS (Quality of Service), congestion-control, and error detection [10, 12]. In addition to all basic header fields, the extended header includes the location and velocity for both the source and the destination, which are used by AeroRP to update its routing table and forward the packet.

B. Past Implementations

Simulation is a cost- and time-efficient approach to model a system. However, to realistically evaluate a system design, real-world implementation is necessary. A comprehensive coverage of previous MANET (mobile ad hoc network) implementations has been presented [13]. Modern operating systems do not provide the needed services for handling outstanding packets for reactive protocols because the initial design assumption is that the topology rarely changes. To implement MANET protocols in current operating systems, five challenges have been identified: handling outstanding packets, updating the route cache, intermixing forwarding and routing functions, new routing models, and cross-layer interactions [14].

Past implementations of MANET protocols used different approaches in terms of implementation tools, testbeds, and operating systems. The AODV (ad hoc on-demand distance vector) protocol was implemented using the Ad-hoc Support Library (ASL) [14]. ASL was implemented using C language as a shared user-space library and tested using Linux 2.4. Additionally, the authors showed how DSR (dy-

namic source routing) has been implemented using the same library. Another MANET protocol, OLSR (optimized link state routing), has been implemented and made publicly available [15]. This implementation was tested on several laptops using Fedora Core 4 Linux with kernel 2.6.x and in different mobility scenarios [16]. AODV and DSDV (destination-sequenced distance-vector) routing protocol implementations have been compared [17], using the Multi-threaded Routing Toolkit [18] to implement DSDV for their comparison study. The two routing protocols have been tested using a set of testbeds including two laptops and three desktops. A metallic anti-static bag is used to decrease the range from 250 meters to 5 meters. The BLR (beaconless routing protocol) was implemented using the tuntap virtual device [19] as an interface to 802.11 devices [20]. The testbed consisted of several laptops running Linux 2.5 with a GPS receiver. To instrument a 330 m transmission range in an open environment, the experiments were performed in an environment with many obstacles such as trees and buildings to reduce the range of each node.

III. SYSTEM COMPONENTS IMPLEMENTATION

A preliminary implementation architecture was introduced in our previous work [11], which presents how ANTP protocols are interconnected. In this paper, we focus on the AeroRP and AeroNP implementations and study their performance; AeroTP implementation is explained in a companion paper [21]. In this section, we present the design decisions and tools used in implementing the system components.

C. System Architecture

The system architecture is designed to provide several features: maintainability, reliability, and data analysis accessibility. To achieve maintainability, the system is designed based on the OOP (object oriented programming) paradigm. This approach aims to eliminate the dependency between the data structures, which gives us the option to more easily upgrade components. For reliability, the system employs try-catch error handling to avoid any I/O errors during runtime. For performance analysis, the system provides a shared logging system that can aggregate the logs in a single Web server. Based on these considerations, the implementation is divided into several components as shown in Figure 1.

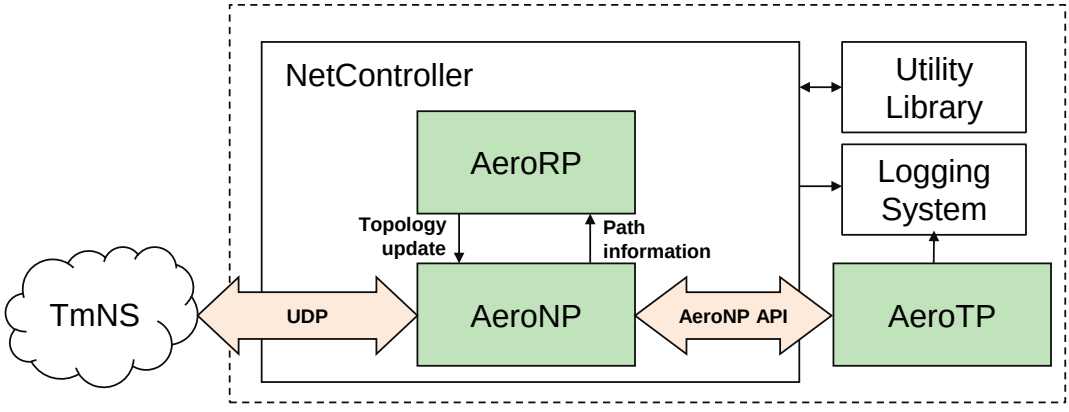


Figure 1: System Architecture

The implementation can be configured to work on three modes: local-emulation, PlanetLab-emulation, and real-time. In local-emulation mode, nodes run as threads in a single machine. The node IDs are gen-

erated from a range specified by the user and each thread is assigned a node ID. The mobility of each node is emulated using the GPS emulator, which we will discuss in Section D. The threads exchange packets directly based on the node ID without any emulated delay, which means there is just processing delay and no propagation delay is added. The main objective of local-emulation mode is to ease the debugging process during the implementation and provide some preliminary results. In PlanetLab-emulation, the implementation runs on GpENI PlanetLab nodes [22, 23]. The node ID is same as the IP address of the PlanetLab node. Similarly, mobility of nodes is emulated using the GPS emulator. The nodes communicate with each other using UDP packets via an Ethernet interface. In real-time mode, the implementation runs on Linux-based embedded devices with built-in GPS and an 802.11 interface. Each device is configured to run in ad-hoc mode and assigned a static IP address. The mobility is not emulated in this case but location is provided by the built-in GPS device. The nodes communicate with each other using UDP packets via an 802.11 interface. For both PlanetLab-emulation and real-time modes, AeroNP PDUs (protocol data units) are encapsulated in UDP datagrams. Thus, we run AeroRP and AeroNP processes in the user-space instead of the kernel space to avoid the challenges presented in Section B.

D. GPS Emulator Implementation

The main objective of the GPS emulator is to add the mobility notion to local-emulation and PlanetLab-emulation. Basically, it uses a mobility model such as our 3D Gauss-Markov [24], random waypoint, or random direction to generate the location and velocity of a given node [25]. The initial position and velocity are either provided by the user or randomly generated by the emulator. The GPS emulator is fully implemented in Python and is designed to run as a separate thread. The GPS emulator accepts several user parameters to construct the emulator object: randomness seed, initial position, initial velocity, and mobility model dependent parameters.

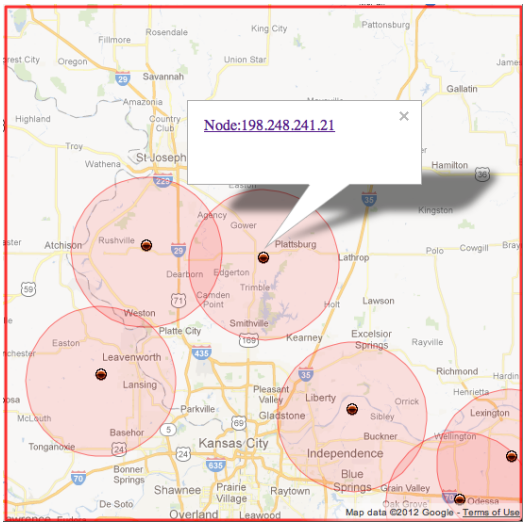
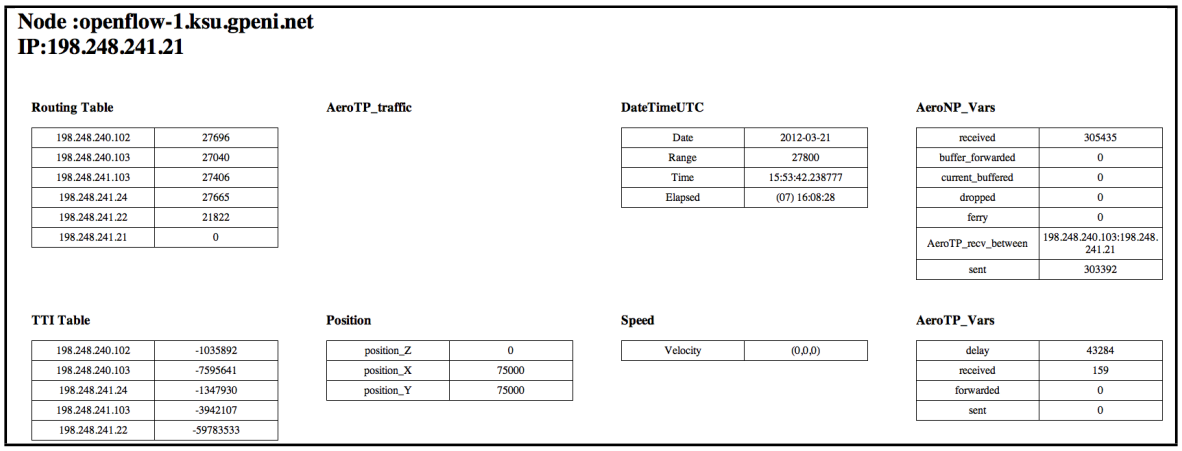


Figure 2: The visualization system: Nodes monitoring page

E. Visualization and Logging System Implementation

The visualization system is added to ease the development and debugging phase of the implementation as well as providing logging data for performance analysis. Furthermore, it provides a demonstration capability to show ATNP protocol operations. It is implemented as a web-based interface with integration of the Google Maps API to show the nodal locations and velocity in real time as shown in Figure 2. The

visualization system consists of a client and server. The client is installed at each node to provide a remote logging capability to the server. Any component in the system can send a log entry to the client, which sends the entry to the server along with additional timestamps and identification headers. On the other end, the server parses these entries from several clients and presents them in the web-interface instantaneously. In addition, the server saves the log entries in a text file for further performance analysis.



Node :openflow-1.ksu.gpeni.net
IP:198.248.241.21

Routing Table

198.248.240.102	27696
198.248.240.103	27040
198.248.241.103	27406
198.248.241.24	27665
198.248.241.22	21822
198.248.241.21	0

AeroTP_traffic

DateTimeUTC

Date	2012-03-21
Range	27800
Time	15:53:42.238777
Elapsed	(07) 16:08:28

AeroNP_Vars

received	305435
buffer_forwarded	0
current_buffered	0
dropped	0
ferry	0
AeroTP_recv_between	198.248.240.103:198.248.241.21
sent	303392

TTI Table

198.248.240.102	-1035892
198.248.240.103	-7595641
198.248.241.24	-1347930
198.248.241.103	-3942107
198.248.241.22	-59783533

Position

position_Z	0
position_X	75000
position_Y	75000

Speed

Velocity	(0,0,0)

AeroTP_Vars

delay	43284
received	159
forwarded	0
sent	0

Figure 3: The visualization system: Detailed node's information page

The web-interface consists of two main pages, which are implemented using two languages: JavaScript and PHP. The first page is the map view page, which shows the current location and velocity of each node. It also shows an approximated transmission range and the AeroTP PDUs. Each node is represented in the map as a marker that can be clicked by the user to show the IP address of the node and a link to the second page of the web-interface, which shows a detailed page listing all the logged entries for this node as shown in Figure 3.

F. AeroRP Component

The AeroRP component is represented as a class with two public functions intended to be used by AeroNP. The first function finds the next hop based on a given destination node. The second function processes updates that fetch incoming AeroNP packets to update the routing table inside the AeroRP component. As shown in Figure 4, AeroRP has other private functions such as the routing algorithm and neighbor discovery controller. The neighbor discovery controller is responsible for advertising the node's location by sending hello messages. The routing algorithm determines the next-hop based on the TTI metric if the protocol runs in ad-hoc or GS-location mode. The shortest-path algorithm is used to determine the next-hop if the protocol runs in GS-topology mode.

G. AeroNP Component

The AeroNP component is represented as a class with several public communication functions such as `send`, `listen`, and `broadcast`. The `send` function can be used by both AeroRP and AeroTP. For AeroTP, the `send` function is called through the AeroNP API interface. The `listen` function can be accessed from the NetController container that receives incoming packets. As shown in Figure 5, AeroNP has other private functions designed as helper functions for the main communication functions. For example, congestion control monitors the congestion level of the network. The `header-update` function is used to update the header with the location and velocity information as well as the congestion indicator.

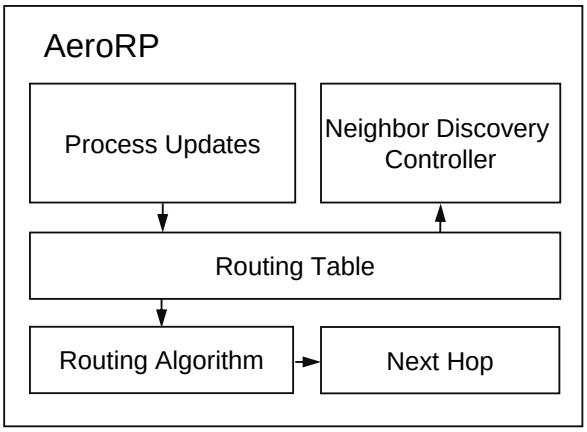


Figure 4: AeroRP component function block

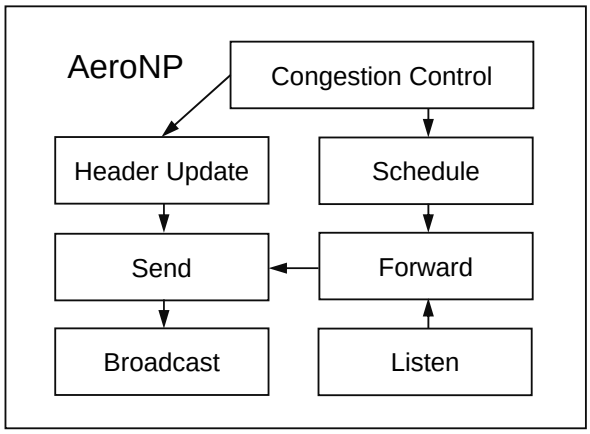


Figure 5: AeroNP component function block

IV. SYSTEM FUNCTIONS IMPLEMENTATION

In this section, we explain details of the AeroRP and AeroNP functions implementation. We present the data structures used in implementing each function.

H. AeroRP Implementation

The AeroRP class uses three other classes to represent routing table entities: `GeolocationTuple`, `NeighborTuple`, and `RoutingTable`. The `GeolocationTuple` class represents a node inside the routing table. It maintains several pieces of information about a node: IP address, location, velocity, range, and timestamp. Furthermore, it maintains a list of neighbors represented as `NeighborTuple` class. The `NeighborTuple` is also implemented as a class but it has less information than `GeolocationTuple`. The main function of this class is to represent the neighbors of each node. It maintains IP address, start time, expire time, and link cost. The `RoutingTable` is a main component of the AeroRP. It maintains a list of the nodes in the network represented as `GeolocationTuple`.

The AeroRP class has two public functions designed to be used by AeroNP classes: `next-hop` and `processUpdates`. The `next-hop` function determines the next hop for a given destination address. For ad-hoc and GS-location modes, it performs the routing algorithm based on the TTI metric. For GS-topology mode, a shortest-path algorithm is used to determine the next-hop. The `processUpdates` function is designed to be called by AeroNP. Incoming packets will be received by the `listen` function inside the AeroNP class. Once a packet is received, a copy is sent to this function, which checks if the packet has any routing table updates.

I. AeroNP Implementation

The AeroNP class uses one of the two packet formats to represent a single AeroNP packet: basic header and extended header. The basic header format is designed to be relatively compact whereas the extended header carries optional location and velocity information that is utilized by the AeroRP routing protocol. `BasicHeader` and `ExtendedHeader` are implemented as classes. The field values of the headers are stored as binary format using packed structures. For each field of the header, there are two accessors to set and get the value of that field. To get the binary representation of the header, each class has a `get-datagram` function.

AeroNP has three public functions `send`, `broadcast`, and `listen`. These functions are designed to be used by AeroRP and AeroTP. The `send` function encapsulates AeroTP and AeroRP PDUs into an AeroNP PDU. The AeroNP payload is obtained by the `get-datagram` function of AeroNP packet. The next-hop address, obtained from the `next-hop` function, is assigned as a destination address for the UDP socket over which the payload is sent. The `broadcast` function is a modular network layer function. For local-emulated and PlanetLab-emulated modes, it uses unicast sending functionality. For real-time mode, the `broadcast` function sends the packet to the broadcast address of the network. The `listen` is a function that is implemented as an infinite while loop. A UDP socket is created and set to listen for incoming packets. If a packet is of extended type header, a copy of the packet is passed to the function `ProcessUpdates` that is located in the AeroRP class to update the table based on the information located.

AeroNP has two private functions: `forward` and `headerUpdate`. The `forward` is a private function that can be called by the `listen` function. Once a packet is received, it is passed to the AeroTP protocol if destination address is the local host address and the protocol ID is the corresponding AeroTP ID; otherwise, it is forwarded to the next hop. The `headerUpdate` is a private function that can be called by the `listen` function. It updates the AeroNP packet header and can accept extended and basic header types. Based on the type, it inserts the geolocation information generated by the GPS location emulator into an AeroNP extended header packet when the packet is sent from the current node. Moreover, it sets the congestion indicator that is obtained from the congestion controller.

J. AeroNP API

The AeroNP API is an AeroNP interface designed for intercommunication between AeroNP and AeroTP. It uses UNIX sockets to exchange packets. The AeroNP API interface has two ends; one is accessed by the AeroNP and the other is accessed by AeroTP; each end has read and write functions. For the AeroNP end, the functions are `send2tp` that delivers packets to AeroTP and `readNPbuffer` that reads packets from AeroTP. For AeroTP, the functions are `send2np` that writes packets to AeroNP and `readNPbuffer` that reads packets from AeroNP. Using the size attribute of the buffer, each end can detect if there are packets to be sent or received.

V. EXPERIMENTS AND RESULTS

In this section, we present our testbed environment and a preliminary experiment, which is configured in local-emulation mode. Then, we present performance analysis of this experiment. Finally, we present a preliminary real-time experiment to check basic functions of the devices and the implementation code.

K. Testbeds

As we explained in Section C., the implementation is designed to operate on one of three modes: local-emulation, PlanetLab-emulation, and real-time. The mode is selected based on the testbed where the implementation will be executed. Local-emulation is intended to work on a single machine. We test this mode on a high-performance Linux box equipped with 72 GB of memory. In PlanetLab-emulation, the implementation runs on a distributed environment consisting of multiple Linux nodes. The hardware specifications for the nodes vary. In real-time mode, the implementation runs on Nokia N810 or N900 smart phones. The phones are attached to remote-controlled cars as shown in Figure 6 and 7. Different mobility scenarios are preformed to test the AeroRP functionality including neighbor discovery and data forwarding. We have selected the Nokia phone because it has a built-in GPS device and 802.11 inter-

face [26]. The phone runs a Linux distribution called Maemo 5 [27], which supports Python. Moreover, the Maemo distribution has a Python GPS library that can be used to acquire location readings from the built-in GPS.

Figure 6: Remote-controlled cars are used for mobility

Figure 7: A Nokia N900 phone is attached to each car

L. Local-emulation Experiment

We performed the local-emulation over an area of 150 $\times$ 150 km^2. All the emulations are averaged over 10 runs with each simulation running for 1000 s. The communication model is peer-to-peer with as many flows as the number of nodes in the network. All the TAs are configured to send 1 packet/s. All the nodes have a transmission range of 15 nmi. The mobility model used is random direction with node velocities ranging from 100 – 1000 m/s. The maximum buffer size is set to 200 packets. We compare the AeroRP and AeroNP implementation performance in the three modes: ad-hoc, GS-Topology, and GS-Location. The performance metrics for the evaluation of AeroRP are packet delivery ratio (PDR) and delay. The PDR is computed as the number of packets received divided by the number of packets sent by the application. The delay is computed as the time interval between the source originating time and the destination delivery time of an application packet.

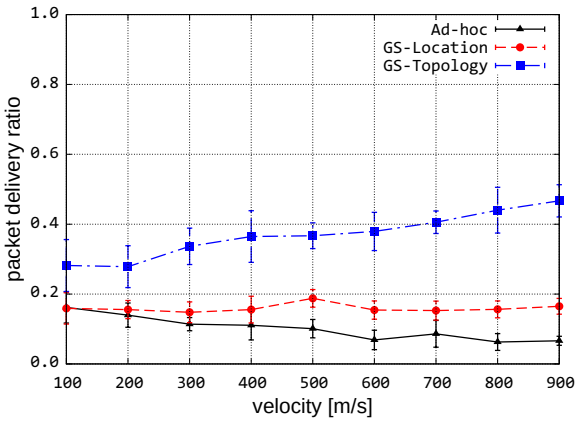


Figure 8: Effect of node velocity on PDR

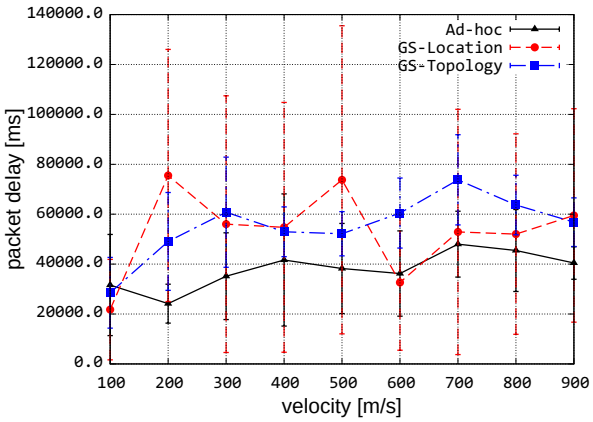


Figure 9: Effect of node velocity on end-to-end delay

In our previous work [12], we ran several simulations to check the performance of AeroRP using ns-3 [6]. One of the scenarios includes the impact of node density on network performance by varying the

number of nodes from 10 to 60 nodes. In an effort to mimic the same scenario in local-emulation mode, we vary the node density; however, the experiment failed after running more than 10 nodes. The reason could be that the system is overwhelmed with many concurrent processes, which exhausted the host system resources. Therefore, we set the number of nodes to 10 nodes and vary the velocity from 100 m/s to 1000 m/s.

As shown in Figure 8, the PDR for the local-emulation mode is about 20% for all the routing modes. Compared to our previous simulation results [12], we can see that AeroRP simulation and local-emulation results are similar (shown in Figure 8 and Figure 7, cf. [12]). On the other hand, the packet delay is about 40 seconds for all of the modes as shown in Figure 9. In the ns-3 simulation results, we observed that the delay is about 3 seconds, which is a significant difference. We assume that the local-emulation may produce more queuing and processing delay but the delay magnitude is beyond our expectation. We speculate that the current local-emulation does not reflect accurate results because of running many concurrent processes that may affect the performance of the whole experiment due to process scheduling.

M. Preliminary Real-time Experiment

We run a simple experiment between two phones to check the functionality of the devices as well as basic functionality of the implementation code. The 802.11 interfaces of the two phones are set to ad-hoc mode and they are given static IP addresses: 10.0.0.56 for the traffic source and 10.0.0.53 for the destination. For this experiment, the nodes are stationary and their locations are hard coded. The source sends AeroTP PDUs to destination. The experiment shows that the Python implementation of AeroRP neighbor discovery and data forwarding functions as expected shown in Figure 10 and 11.

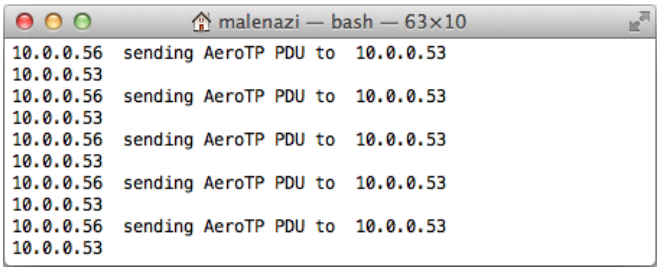


Figure 10: Source screenshot

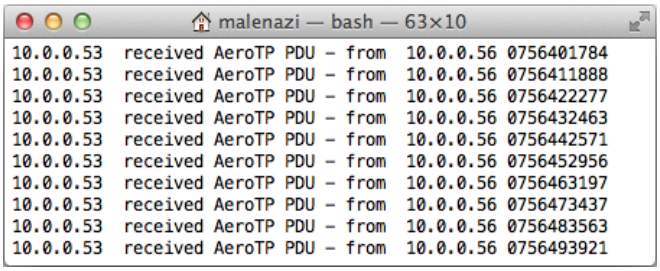


Figure 11: Destination screenshot

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have implemented AeroRP and AeroNP protocols in Python. In addition, we implemented several tools and services to ease the phases of the development and performance analysis such as GPS emulator, logging system, and real-time visualization system. The implementation has been designed to work in several modes: local-emulation, PlanetLab-emulation, and real-time. The results for the local-emulation mode are presented and discussed.

For future work, we are planning to test the implementation using varying number of PlanetLab nodes to see the impact of node density. In addition, we will study the impact of different mobility models such as randomway point and Gauss-Markov on the network performance. We will run and study more realistic scenarios on the Nokia N810 or N900 with real GPS devices using remote-controlled cars. Also as part of future work, we are planning to run the implementation on multiple Nokia phones to check the multi-hop functionality and to see how the performance results compare to simulation and emulation.

ACKNOWLEDGEMENTS

The authors would like to thank the Test Resource Management Center (TRMC) Test and Evaluation/Science and Technology (T&E/S&T) Program for their support. This work was funded in part by the T&E/S&T Program through the Army PEO STRI Contracting Office, contract number W900KK-09-C-0019 for AeroNP and AeroTP: Aeronautical Network and Transport Protocols for iNET (ANTP). The Executing Agent and Program Manager work out of the AFFTC. This work was also funded in part by the International Foundation for Telemetering (IFT). We would like to thank Kip Temple and the membership of the iNET working group for discussions that led to this work.

REFERENCES

[1] J. P. Rohrer, A. Jabbar, E. K. Çetinkaya, and J. P. Sterbenz, "Airborne telemetry networks: Challenges and solutions in the ANTP suite," in *Proceedings of the IEEE Military Communications Conference (MILCOM)*, (San Jose, CA), pp. 74–79, November 2010.

[2] J. P. Rohrer, A. Jabbar, E. K. Çetinkaya, E. Perrins, and J. P. Sterbenz, "Highly-dynamic cross-layered aeronautical network architecture," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, pp. 2742–2765, October 2011.

[3] J. P. Rohrer, A. Jabbar, E. Perrins, and J. P. G. Sterbenz, "Cross-layer architectural framework for highly-mobile multihop airborne telemetry networks," in *Proceedings of the IEEE Military Communications Conference (MILCOM)*, (San Diego, CA, USA), pp. 1–9, November 2008.

[4] J. P. Rohrer, E. Perrins, and J. P. G. Sterbenz, "End-to-end disruption-tolerant transport protocol issues and design for airborne telemetry networks," in *Proceedings of the International Telemetering Conference (ITC)*, (San Diego, CA), October 2008.

[5] A. Jabbar, E. Perrins, and J. P. G. Sterbenz, "A cross-layered protocol architecture for highly-dynamic multihop airborne telemetry networks," in *Proceedings of the International Telemetering Conference (ITC)*, (San Diego, CA), October 2008.

[6] "The ns-3 network simulator." `http://www.nsnam.org`, July 2009.

[7] K. S. Pathapati, T. A. N. Nguyen, J. P. Rohrer, and J. P. Sterbenz, "Performance analysis of the AeroTP transport protocol for highly-dynamic airborne telemetry networks," in *Proceedings of the International Telemetering Conference (ITC)*, (Las Vegas, NV), October 2011.

[8] K. Peters, A. Jabbar, E. K. Çetinkaya, and J. P. Sterbenz, "A geographical routing protocol for highly-dynamic aeronautical networks," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, (Cancun, Mexico), pp. 492–497, March 2011.

[9] J. P. Rohrer, E. K. Çetinkaya, H. Narra, D. Broyles, K. Peters, and J. P. G. Sterbenz, "AeroRP Performance in Highly-Dynamic Airborne Networks using 3D Gauss-Markov Mobility Model," in *Proceedings of the IEEE Military Communications Conference (MILCOM)*, (Baltimore, MD), pp. 834–841, November 2011.

[10] H. Narra, E. K. Çetinkaya, and J. P. Sterbenz, "Performance Analysis of AeroRP with Ground Station Advertisements," in *Proceedings of the ACM MobiHoc Workshop on Airborne Networks and Communications*, (Hilton Head Island, SC), pp. 43–47, June 2012.

[11] M. Alenazi, S. A. Gogi, D. Zhang, E. K. Çetinkaya, J. P. Rohrer, and J. P. G. Sterbenz, "ANTP Protocol Suite Software Implementation Architecture in Python," in *Proceedings of the International Telemetering Conference (ITC)*, (Las Vegas, NV), October 2011.

[12] H. Narra, E. K. Çetinkaya, and J. P. Sterbenz, "Performance analysis of aerorp with ground station updates in highly-dynamic airborne telemetry networks," in *Proceedings of the International Telemetering Conference (ITC)*, (Las Vegas, NV), October 2011.

[13] 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.

[14] V. Kawadia, Y. Zhang, and B. Gupta, "System services for ad-hoc routing: Architecture, implementation and experiences," in *Proceedings of the 1st international conference on Mobile systems, applications and services*, pp. 99–112, ACM, 2003.

[15] A. Tonnesen, T. Lopatic, H. Gredler, B. Petrovitsch, A. Kaplan, and S. Tcke, "OLSRD: An adhoc wireless mesh routing deamon," 2008.

[16] L. Barolli, M. Ikeda, F. Xhafa, and A. Duresi, "A Testbed for MANETs: Implementation, Experiences and Learned Lessons," *IEEE Systems*, vol. 4, no. 2, pp. 243–252, 2010.

[17] K. Chin, J. Judge, A. Williams, and R. Kermode, "Implementation experience with MANET routing protocols," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 5, pp. 49–59, 2002.

[18] C. Labovitz and F. Jahanian, "Experimentation with Multi-threaded, Distributed Routing Technology in the Internet," tech. rep., Merit Network. Inc. and The University of Michigan, 1995.

[19] M. Krasnyansky and M. Yevmenkin, "Virtual point-to-point (tun) and ethernet (tap) devices," 2006.

[20] T. Braun, M. Heissenbüttel, and T. Roth, "Performance of the beacon-less routing protocol in realistic scenarios," *Ad Hoc Networks*, vol. 8, pp. 96–107, January 2010.

[21] S. A. Gogi, D. Zhang, E. K. Çetinkaya, J. P. Rohrer, and J. P. G. Sterbenz, "Implementation of the AeroTP Transport Protocol in Python," in *Proceedings of the International Telemetering Conference (ITC)*, (San Diego, CA), October 2012.

[22] "GpENI: Great Plains Environment for Network Innovation." `http://gpeni.net`, November 2009.

[23] J. P. G. Sterbenz et al., "The Great plains Environment for Network Innovation (GpENI): A programmable testbed for future internet architecture research," in *Proceedings of the 6th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom)*, (Berlin, Germany), pp. 428–441, May 2010.

[24] M. J. Alenazi, C. Sahin, and J. P. G. Sterbenz, "Design Improvement and Implementation of 3D Gauss-Markov Mobility Model," in *Proceedings of the International Telemetering Conference (ITC)*, (San Diego, CA), October 2012.

[25] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless Communications and Mobile Computing*, vol. 2, no. 5, pp. 483–502, 2002.

[26] "Nokia N900." `http://europe.nokia.com/find-products/devices/nokia-n900`, 2012.

[27] "Maemo linux platform." `http://maemo.org`, 2012.