Implementation of Aeronautical Network Protocols

Mohammed J.F. Alenazi*, Santosh Ajith Gogi*, Dongsheng Zhang*
Egemen K. Çetinkaya*, Justin P. Rohrer†, and James P.G. Sterbenz‡

The University of Kansas, Lawrence, KS, 66045, USA

The aeronautical ANTP protocol suite consisting of AeroTP, AeroRP, and AeroNP has been designed to cope with the challenges in highly-dynamic airborne networks. Via simulations, these protocols have shown significant improvement compared with other traditional protocols. In this paper, we present a prototype implementation of ANTP protocol suite using Python. The implementation is tested in a real-time environment using Linux devices equipped with GPS receiver and radio-controlled vehicles for mobility. The results of several mobility scenarios show correct implementation of protocol's functions and services.

I. Introduction

Data communication in a highly dynamic airborne environment has several challenges, which results in a need for designing new protocol stack that mitigates the impact of these challenges. The challenges in this dynamic airborne environment include highly mobile nodes, unreliable wireless channel, and asymmetric channel between data source and sink.[1] A typical example of an airborne data communication environment is a telemetry network, which consists of three types of nodes: airborne nodes (ANs), relay nodes (RNs), and ground stations (GS) as depicted in Figure 1. Fast moving ANs send telemetry data to a GS through one another or RNs, and RNs have a higher power antenna.

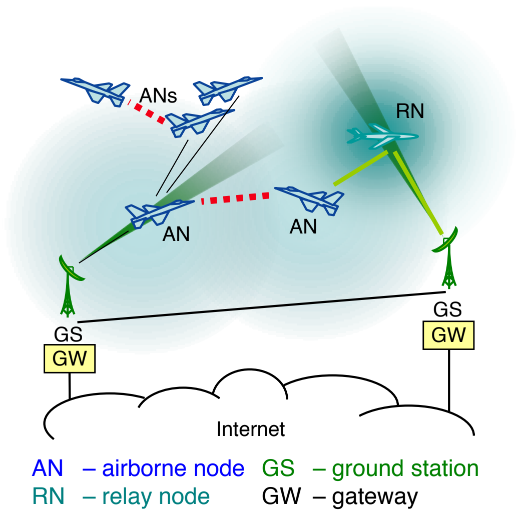


Figure 1. Dynamic airborne telemetry environment

*Graduate Student, Department of Electrical Engineering & Computer Science and Information & Telecommunication Technology Center, The University of Kansas, Lawrence, KS, 66045, USA.

†Justin P. Rohrer is now with the Department of Computer Science, Naval Postgraduate School, Monterey, CA 93943–5285 (e-mail: jprohrer@nps.edu); work performed primarily at The University of Kansas

‡Associate Professor, Department of Electrical Engineering & Computer Science and Information & Telecommunication Technology Center, The University of Kansas, Lawrence, KS, 66045, USA.

Previously, we have developed the domain-specific ANTP (Aeronautical Network and Transport Protocol) protocol suite that operates in this highly-dynamic aeronautical environment.[2,3] Later, we have shown superior performance of ANTP protocols over traditional network protocols using the ns-3 discrete network simulator.[4–8] The next step towards deployment of the ANTP suite is developing a cross-platform implementation of the protocols. Towards this end, we introduced a preliminary system architecture for ANTP suite implementation and presented preliminary results.[9–11] In this paper, we present our implementation of ANTP suite, tools we utilize during implementation, and results of real-time experiments. The implementation is tested in a real-time environment using Linux devices equipped with GPS receiver and radio-controlled vehicles for mobility.

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 & Related Work

This section gives an overview of transport, routing, and network protocols in the ANTP suite, as well as past implementations of MANET protocols.

A. ANTP Suite

The Aeronautical Network and Transport Protocol (ANTP) suite[2,3] is designed to mitigate challenges in highly-dynamic airborne environment and provide edge-to-edge compatibility with the legacy Internet architecture. The ANTP protocol suite shown in Figure 2, which includes: *AeroTP* – a TCP-friendly transport protocol[12] with multiple reliability and QoS modes, *AeroNP* – an IP-compatible network protocol (addressing and forwarding),[13] *AeroRP* – a routing protocol,[13] which exploits velocity and location to forward packets, and *AeroGW* that provides necessary translation functions via gateways[14] between domain-specific ANTP suite and traditional networking protocols in the ground and airborne network. The simulation results of ANTP protocols showed superior results over other traditonal protocols.[4–7,15,16]

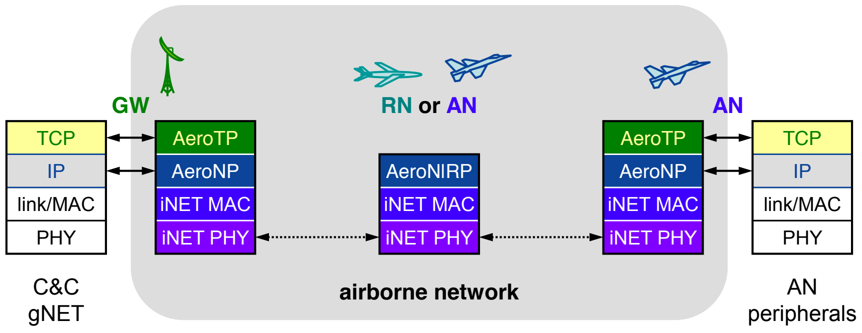


Figure 2. Airborne network protocol architecture

AeroTP has been designed and evaluated previously using the ns-3 simulator.[7,15,17] In a lossy network environment, AeroTP reliable-mode performs significantly better compared to the traditional TCP. With a higher BER (bit error rate), end-to-end delay in TCP increases by orders of magnitude whereas AeroTP has less than an order of magnitude increase. AeroTP quasi-reliable mode has much less loss due to the FEC recovery mechanism compared to UDP that drops packets with increasing error rates. Furthermore, AeroTP reliable mode has less overhead compared to TCP, while quasi-reliable mode does not cause increased delay but has significant overhead.

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.[4, 5, 18]

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 services such as QoS (Quality of Service), congestion-control, and error detection.[5, 18] 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.[19] Modern operating systems do not provide services to handle 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.[20]

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).[20] ASL was implemented using C language as a shared user-space library and tested using Linux 2.4. Additionally, the authors showed how DSR (dynamic source routing) has been implemented using the same library. Another MANET protocol, OLSR (optimized link state routing), has been implemented and made publicly available.[21] This implementation, was tested on several laptops using Fedora Core 4 Linux with kernel 2.6.x and in different mobility scenarios.[22] AODV and DSDV (destination-sequenced distance-vector) routing protocol implementations have been compared,[23] using the Multi-threaded Routing Toolkit[24] 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[25] as an interface to 802.11 devices.[26] 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.

Many designs and implementations of network and transport protocols have been carried out. A transport layer protocol that relies on end-to-end mechanisms to detect network state was implemented in the Linux kernel and results were also verified with simulations.[27] The significant impact of a wireless multihop channel on TCP throughput and loss gives insight why TCP cannot be used in wireless networks.[28] Our work is different as AeroTP is a domain-specific transport protocol designed for end-to-end communication in airborne telemetry networks.

III. System Architecture

The system architecture is designed to provide several features: maintainability, reliability, and data analysis accessibility. The system architecture is shown in Figure 3, which was initially introduced in our previous work.[9–11] In this section, we explain the main components AeroTP, AeroRP, and AeroNP. Furthermore, the utility library consists of several functions essential to the implementation process. The library is shared by all protocols of the system. The logging system component is added to check operational correctness and to monitor the performance of the Aero protocols.

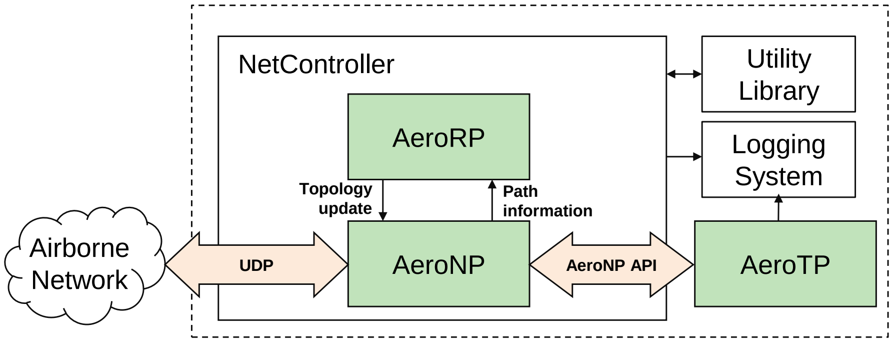


Figure 3. System Architecture

A. AeroTP Component

The AeroTP component is represented as a class with several functions. AeroTP utilizes services provided by network layer protocol AeroNP. AeroTP reliable and near-reliable mode use ARQ mechanisms to provide full reliability within the ANs.[17] The source and destination use control messages for connection establishment and termination. AeroTP quasi-reliable mode utilizes forward error correcting codes to achieve arbitrary level of statistical reliability.

B. 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. 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.

C. 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. AeroNP has other private functions designed as helper functions for the main communication functions.

IV. System Functions Implementation

In this section, we explain details of the AeroTP, AeroRP, and AeroNP functions implementation. We present the data structures used in implementing each function. The development and testing of the ANTP protocols is carried out in the high-level scripting language Python. It provides a number of data structures and libraries, in particular, networking libraries such as sockets, packing binary data, and CRC (cyclic redundancy check). Python-based implementations require less programming time leading to fast prototyping of applications.[29]

A. Implementation Operating Modes

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 generated 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 later in this section. 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.[30,31] The node ID is the 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 AeroTP, AeroRP, and AeroNP processes in the user-space instead of the kernel space to avoid the challenges presented in Section B.

B. AeroTP Implementation

AeroTP is implemented using a centralized top-level controller module that guides the flow setup, termination, and data transfer as shown in Figure 4, and employs supplemental modules designed for specific services. The controller runs independently on a POSIX operating system environment as a dæmon on both source and destination nodes to provide services to the application layer for end-to-end communication. The current implementation uses UDP sockets for compatibility with the security restrictions in PlanetLab testbed[32] systems, which is one of the emulation environments using the GpENI testbed.

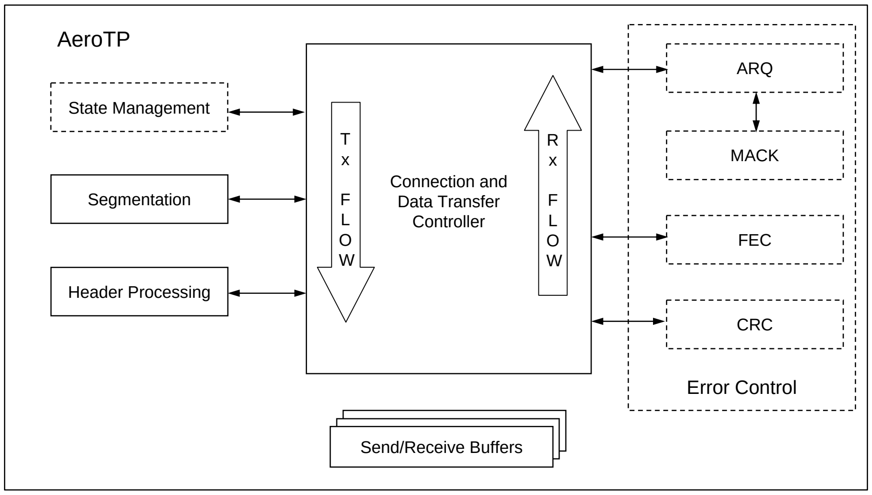


Figure 4. AeroTP implementation architecture

The controller flow at the source begins with connection setup that uses the state machine in connection-oriented modes. During the connection establishment, the overhead of a three-way handshake is eliminated by an opportunistic connection approach that overlaps data with control. Segmentation is followed with optional FEC error correcting codes appended to the payload. Header processing involves the application layer QoS and error control requirements embedded into the `AeroTP_Header` object. Depending on the class of service used by the application, error controlling mechanisms are employed in the subsequent steps. These functions are complimented appropriately at the destination by decomposing the segment based on service mode, maintaining state, and aggregating ACKs followed by the connection teardown. In addition to these modules, a consistent and efficient buffer-handling mechanism augments the main controller using sender and receiver buffers.

1. *End-to-end Reliability with ARQ mechanism*

AeroTP reliable and near-reliable mode use ARQ mechanisms to provide full reliability within the airborne network.[17] MACK (multiple ACK) is used as the acknowledgement mechanism. It dynamically aggregates a number of ACKs hinging upon the transmission rate and loss rate. The source and destination use control messages (ASYN, ASYNACK, AFIN, AFINACK) for connection establishment and termination as shown in Figure 5. The source initiates an end-to-end connection by sending the ASYN `AeroTP_Header` object with the SYN flag set. A piggyback flag is set to provide options to use traditional three-way handshake or opportunistic connection establishment. Once the ASYN is received from the source, the connection state is set to ESTABLISHED (Figure 8) at the destination and the sequence number of the ASYN is recorded in a MACK control message to be sent back to the source. The first acknowledged sequence number is inserted in the MACK header and the following acknowledged sequence numbers are appended as the payload. The structure of a MACK control message is shown in Figure 6 where n and m are the first and last acknowledged TPDU sequence numbers, respectively.

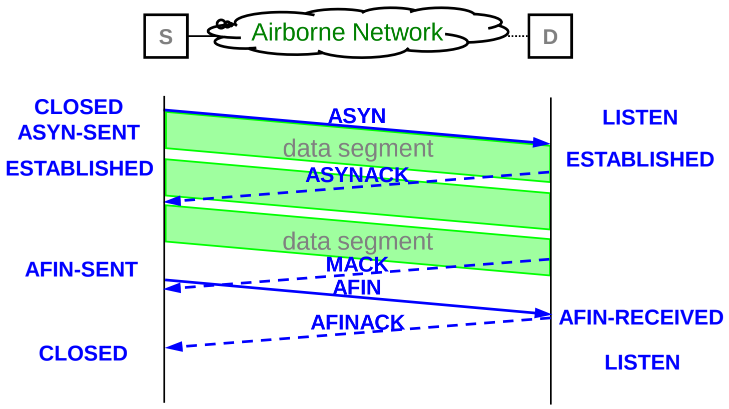


Figure 5. ARQ connection management

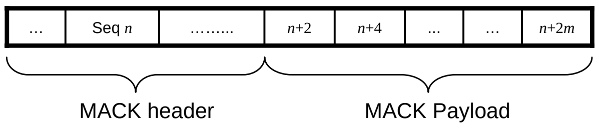


Figure 6. MACK header and payload

After the source sends out the ASYN, it moves to the ESTABLISHED state immediately. The source appends TPDUs (transport protocol data units) to the transmission buffer and sends them to the destination in FIFO (first in, first out) order. The AeroTP protocol employs a selective-repeat ARQ mechanism to provide a reliable edge-to-edge connection. Whenever the source sends out a TPDU, the sequence number and TPDU are stored in an additional buffer used for possible retransmission, implemented by using a hash table (`dict()` data structure in Python) with the sequence number as the keys. Figure 7 is an example showing the retransmission procedure. By checking the next MACK from the destination, the source will remove the sequence number with its counterpart TPDU sequence numbers (2, 6, 8, 10, and 12) in the retransmission buffer and append the unacknowledged TPDUs (sequence number 4) to sender buffer.

After the transmission buffer is empty, the source will send out an AFIN immediately to signal the end of the current connection. However, it is possible that some of the last sent several TPDUs are lost during the transmission. Therefore, an alternative retransmission mechanism is exploited. When the last MACK arrives at the source, if all the sequence numbers of the last set of TPDUs are acknowledged, the source will just wait for AFINACK to terminate the connection; if some sequence numbers are missed, it triggers

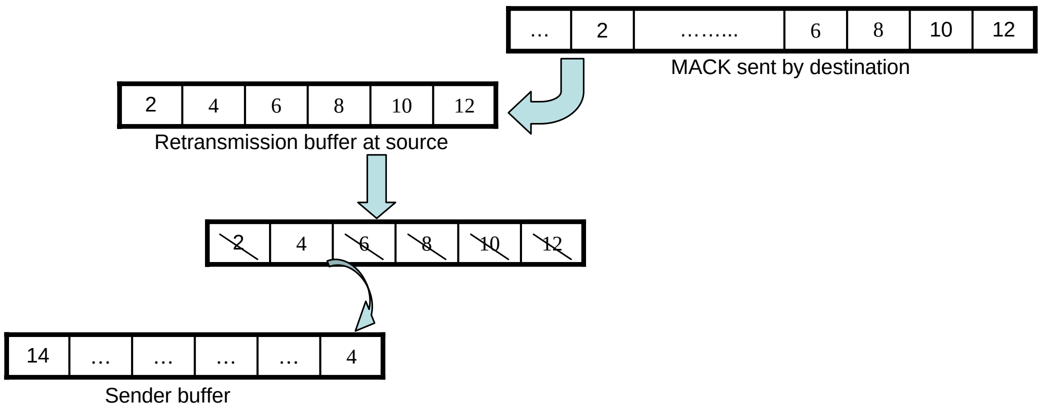


Figure 7. AeroTP ARQ retransmission mechanism

a time-out mechanism for the retransmission of the last set of TPDUs. In this case, the destination will send an ACK back to the source immediately for each single TPDU so that each TPDU is acknowledged individually. A time-out thread is employed for each TPDU by using the Python class `threading.Timer`, and when the timer expires it will trigger automatic retransmission of that TPDU. When AFINACK reaches the source and no TPDUs are in the retransmission hash table, the source moves to the CLOSED state and the destination moves to the LISTEN state. The state management module described next, is used for effective event transitions.

The state management module is a FSM (finite state machine) based module[33] that effectively manages the state transitions in the case of AeroTP connection-oriented modes. The state transition diagram is shown in Figure 8. The module maintains AeroTP states and possible event transitions with action handlers. A set of tuples each representing event transitions (`input_symbol`, `current_state`, `action`, `next_state`) with optional `input_symbol` are initialized that vary depending on the AeroTP reliability mode. Then, based on an event occurrence, the state transitions from `current_state` to `new_state` and takes corresponding action. The events are listed in Table 1.

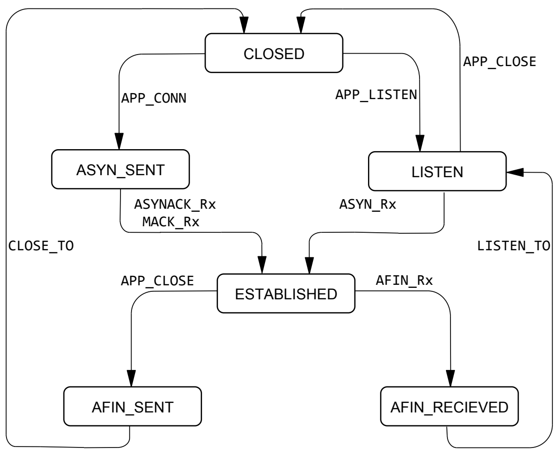


Figure 8. AeroTP state machine

2. Quasi-Reliability with FEC mechanism

Due to the inherent behavior of bit-errors resulting from noise and interference in wireless links with dynamic mobility, relying alone on retransmissions to efficiently provide reliable data transfer is not sufficient. An open-loop error recovery mechanism such as FEC achieves an arbitrary level of statistical reliability. This

Table 1. State transition definitions

State	Description
CLOSED	State in which no connection exists and no data is transferred
LISTEN	State in which a destination is ready to listen to any incoming data
ASYN_SENT	ASYN message sent by the host initiating connection
ESTABLISHED	Steady state in which data transfer takes place
AFIN_SENT	AFIN message sent to indicate no new data being sent
AFIN_RECEIVED	AFIN message received and AFINACK is sent as an acknowledgement
APP_CONN	Request issued by the application to initiate connection by sending ASYN
APP_LISTEN	Request issued to the receiving host to move to the LISTEN state
APP_CLOSE	Request to initiate closing a connection by sending AFIN
ASYN_RX	ASYN received, indicating a connection has been requested
ASYNACK_RX	ASYNACK received, indicating connection request has been granted
MACK_RX	Single or multiple packet ACK received
AFIN_RX	AFIN received, indicating end of any new data, initiating connection close
CLOSE_TO	A timeout allowing outstanding retransmissions before going to CLOSED state
LISTEN_TO	A timeout to go to LISTEN state so that the destination can receive all data packets

also means it does not guarantee absolute delivery of data, while eliminating the need of ACKs and ARQ completely. We use the RS (Reed-Solomon) class of error correcting codes that are predominantly used in applications such as RAID-like systems, optical disk storage devices, and optical communication technologies. Quasi-reliable mode also manages a subset of the state transitions discussed earlier. RS error correcting codes are linear block codes that can detect and correct multiple bursts of errors. They represents data as polynomials and oversample them. An (n, k) code encodes k source data bytes into n codewords, with the ability to correct at most $(n-k)/2$ errors. We use the publicly available RS error correcting Python library[34] for our FEC mechanism.

A widely used RS code uses 8-bit symbols as it facilitates byte-oriented systems and is computationally less intensive. This corresponds to the number of symbols in the encoded block to be $n = 2^8 - 1 = 255$. To provide different error correcting capabilities, k can be varied. To account for a higher payload size in the TPDU, a mechanism to iteratively encode and decode codewords is incorporated.

- During encoding at source, chunks of codewords are computed with specified FEC strength and the default n being 255. Two conditions are possible: the sum of the lengths of codewords computed exceeds the length of the maximum payload size, or the length of the codewords is less than the block size of 255. In the former, n is calculated based on the strength whereas k is computed for blocks of smaller size in the latter case.

- During decoding at the destination, each chunk of the payload is decoded individually which involves computation of k. To simulate different noise levels in the network during analysis stage, a user-specified BER based `RateBasedErrModel` is developed. This introduces burst errors into the payload calculated using its length and BER.

An object of RS based error correcting class is created using a constructor with `n` and `k` arguments computed as discussed previously. FEC adds redundancy for error correcting capability that overcomes the necessity of retransmissions.

3. Supplemental modules

While the centralized controller provides the main functionality, supplemental modules serve as utilities to simplify operations and provide reusable modules across different AeroTP modes. These modules make use of the object-oriented features of Python. They act as independent libraries, instantiated by the centralized controller.

Segmentation and header processing modules are a prerequisite for processing related to size and alignment of control and payload data during network communication. The segmentation module accepts application data and segments into chunks for transmission. In contrast, the destination side decapsulates the segment from the receiver buffer `RECV_BUFF`. Segmentation also provides the sequence numbering mechanism, which numbers each TPDU to permit resequencing. Header processing at the source helps in constructing the header in network byte order using the binary packing library method `struct.pack(format, v1, v2, ...)`, which can easily be decomposed at the destination. The arguments `v1,v2,..` representing header fields are packed according to `format`. This module provides extensive support in terms of handling header field attributes.

Various other modules serve as tools to validate and analyze the overall implementation. A utility module converts values between different numerical base formats and provides time related functions using the `datetime` library. A traffic generator module generates CBR (constant bit rate) application traffic at a specified rate, which emulates end-to-end application layer protocol behavior. The CRC protects the integrity of data end-to-end across the network. It can be used in conjunction with FEC to detect errors. The HEC (header error check) is a 16-byte strong CRC whereas the payload has a 32-byte CRC performed by the Python library module `binascii`.

C. AeroRP Implementation

The AeroRP class uses three 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 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.

D. 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 `ProcessUpdates` function that is located in the AeroRP class and the routing table is updated based on the location information present in the extended header.

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.

E. 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.

F. 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,[35] random waypoint, or random direction to generate the location and velocity of a given node.[36] 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.

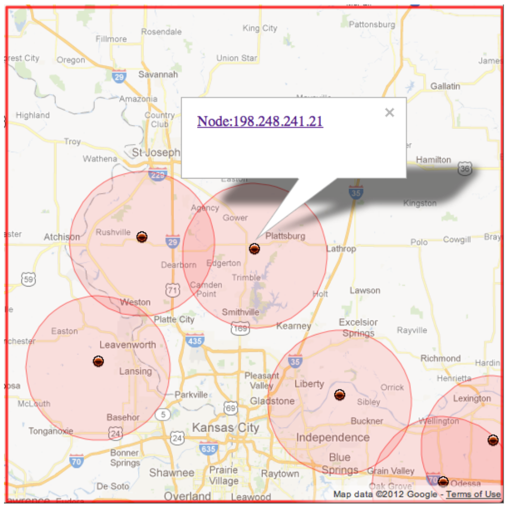
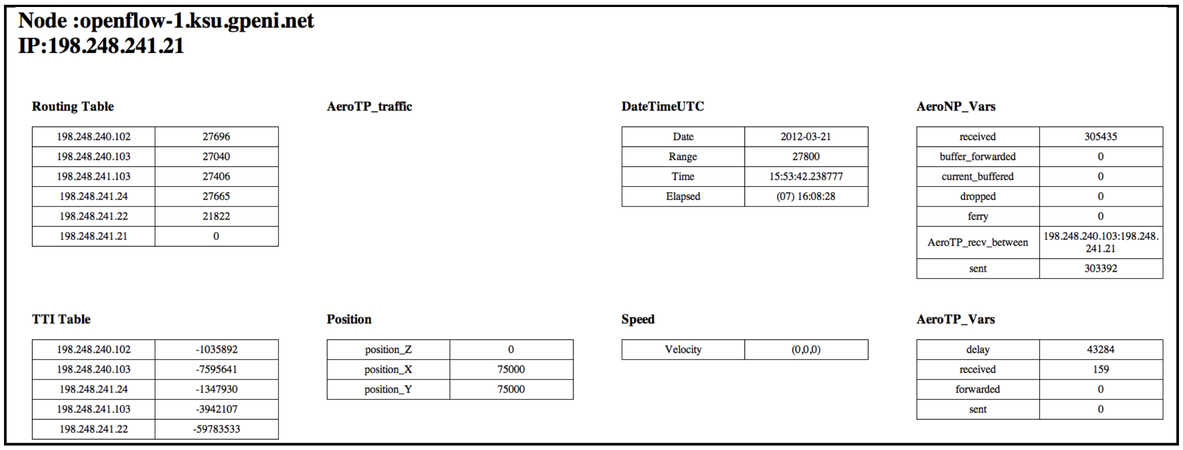


Figure 9. The visualization system: Nodes monitoring page

G. 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 9. 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 that 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 10. 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. A link to the second page of the web-interface shows a detailed page listing all the logged entries for this node as shown in Figure 10.

V. Experiments and Results

In this section, we first present our testbed environment that consists of local-emulation, PlanetLab-emulation, and real-time modes. Then, we present the results of out experiments that validates our implementation.

A. Test Environment

As we explained in Section IV, 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 11 and 12.

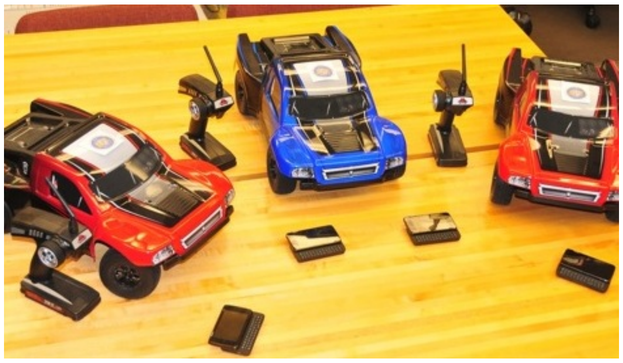


Figure 11. Remote-controlled cars are used for mobility

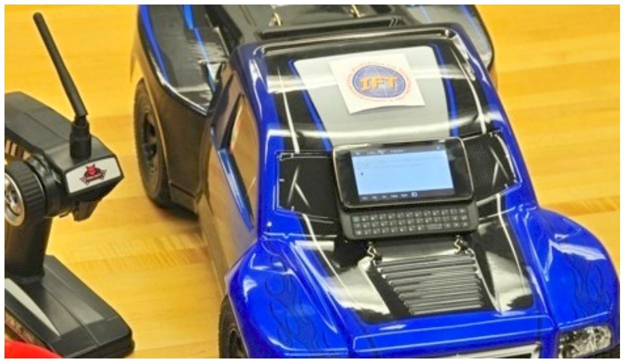


Figure 12. A Nokia N900 phone is attached to each car

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 interface.[37] The phone runs a Linux distribution called Maemo 5,[38] 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.

B. Local-emulation Experiments

In the local-emulation experiments, the objective is to run implementation locally to verify the correctness of the code and the basic functions of the implementation. We performed the local-emulation over an area of 150 $\times$ 150 km^2. All the emulations are averaged over 10 runs with each 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 27800 m (15 nautical mi). 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 PDR (packet delivery ratio) 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.

In our previous work,[18] we ran several simulations to check the performance of AeroRP using ns-3 .[39] 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.

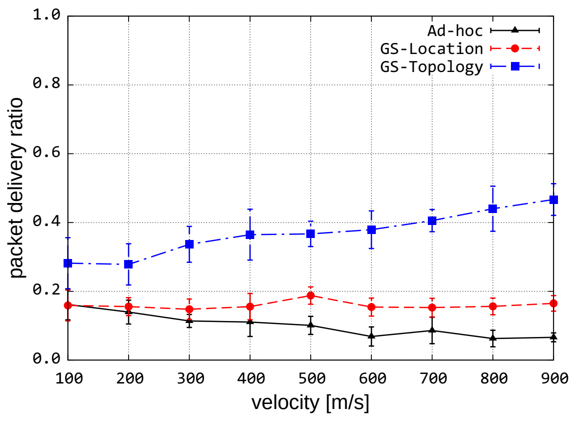


Figure 13. Effect of node velocity on PDR

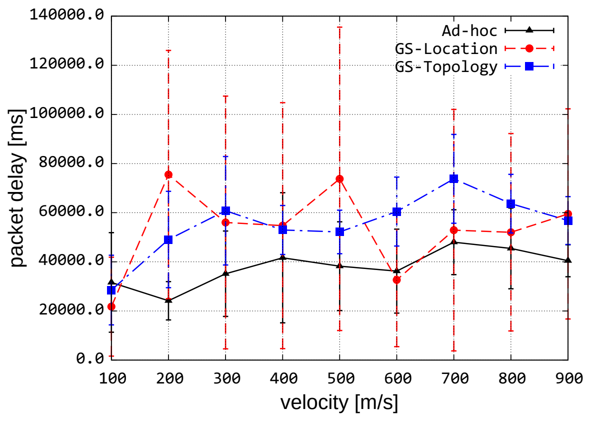


Figure 14. Effect of node velocity on end-to-end delay

As shown in Figure 13, the PDR for the local-emulation mode is about 20% for all the routing modes. Compared to our previous simulation results,[18] we can see that AeroRP simulation and local-emulation results are similar (shown in Figure 13 and Figure 7, cf.[18]). On the other hand, the packet delay is about 40 seconds for all of the modes as shown in Figure 14. 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.

C. PlanetLab-emulation Experiments

Functional testing of all AeroTP modes is initially conducted on the GpENI testbed.[30,31] Here, we analyze the performance of AeroTP quasi-reliable mode in a lossy environment. In order to test the accuracy of the FEC mechanism, we have simplified the testing on two systems operating with the Maemo Linux distribution.

1 MB application data is transmitted from one of the systems to another during a single scenario. FEC strengths provided by RS codes ranging from (255, 254) to (255,74) are tested. Note that the higher the difference between n and k, the higher the strength. We measure the cumulative goodput and overhead of using different FEC strengths that provides statistical reliability. Emulation of errors with varying rates is done at the destination end receiving interface.

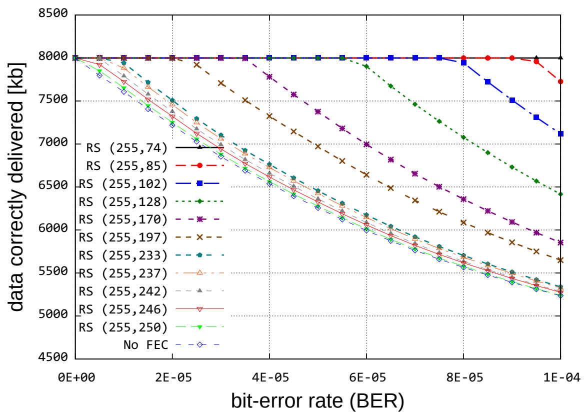


Figure 15. Cumulative goodput

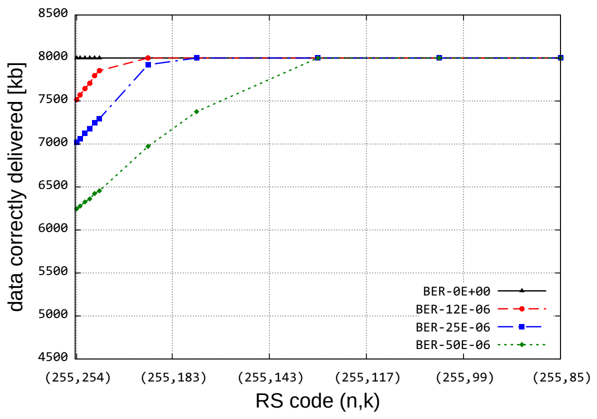


Figure 16. Cumulative goodput

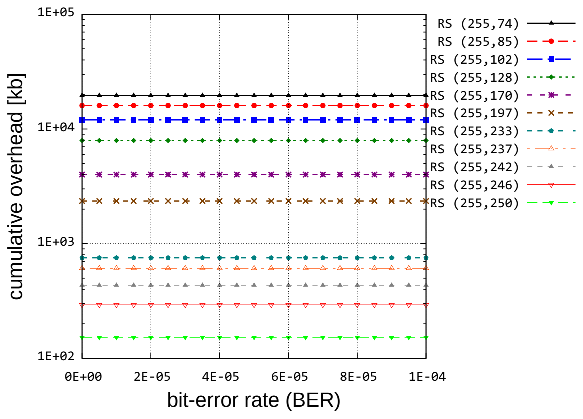


Figure 17. Cumulative overhead

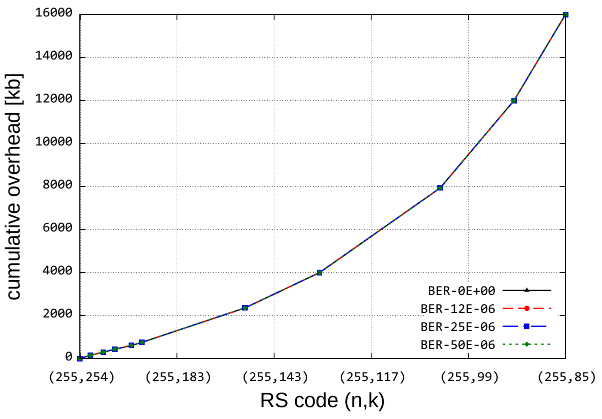


Figure 18. Cumulative overhead

Figure 15 shows the amount of data correctly delivered as BER increases with varying FEC strength. At the error rates tested, except for the highest strength RS (255,74) code, the total amount of correctly delivered data decreases as the error rate increases. The AeroTP quasi-reliable mode eliminates the need for ACKs and retransmissions in the case of lossy links that cause bit errors at the cost of not *guaranteeing* reliability. Due to increased noise and interference in the wireless channel, more bit errors reduce the amount of correctly delivered data. This can be compensated by error correction with sufficient FEC strength. Figure 16 shows the amount of data correctly delivered for different FEC strength with lower error rates. Each FEC strength has a threshold BER that can deliver entire data correctly. An FEC strength of RS (255,128) code is able to correct errors occurring at rate $\leq 5.0 \times 10^{-5}$.

The next set of plots indicate the overhead added by the FEC mechanism. Figure 17 shows that as the FEC strength is increased, the overhead added by the RS codes increases (note the log y-axis scale). This means more packets are required to carry the application data. The reliability induced in quasi-reliable mode comes at the cost of additional FEC codewords added to the packet, reducing the application data in each packet. Note that error rate does not affect the overhead for a given strength. Figure 18 shows the increase in the overhead with increased FEC strength, quantified by the number of packets sent. The results are comparable to the simulation results.[15] Significant overhead is added that can influence the FEC strength value to be used in AeroTP. The strength value can be adapted based on the error-rates, by cross-layering link characteristics between link and MAC (media access control) layer.

D. Real-time Experiments

In this section, we present several scenarios of real-time experiments. The testbed for this experiments is a number of Nokia N810 and N900 phones with real GPS devices and remote-controlled cars for mobility. All of the experiments in this section use the AeroRP ad-hoc mode. The data traffic is generated from one node as AeroTP PDU (protocol data unit) and destined to the receiver. We start with two nodes to show the basic functionality of AeroRP and AeroNP. Then, we add relay nodes to show the buffering and forwarding functionality. The nodes use real GPS to determine their position and velocity. The sender sends an AeroTP PDU every second. This scenario is to verify functionality of the GPS with AeroRP neighbor discovery and data forwarding functions. All the nodes start within range of each other. Then, the sender and the receiver get out of range while staying in range with the relay nodes. The range of nodes used by the AeroRP is set to 80 meters, which is a threshold for significant packet losses. For mobility, we used RC cars shown in Figure 11 to move the phones around during the experiments.

Table 2. Performance results in real-time experiments

Scenario	Sent	Forwarded	Received	PDR
One sender and one receiver	200	0	100	90%
One sender, one receiver, and one relay	100	50	82	82%
One sender, one receiver, and two relays	200	119	140	70%

In the one sender and one receiver experiment, the sender sends 200 AeroTP PDUs and results in an average PDR of 90%. This experiment shows the AeroNP and AeroRP neighbor discovery. In the one sender, one receiver, and one relay experiment, the number of sent AeroTP PDUs is 100 and the number of received AeroTP PDUs is 82. The number of forwarded packets at the relay node is 50. This experiment shows a packet delivery ratio of 82%. In the one sender, one receiver, and two relays experiment, the number of sent AeroTP PDUs is 200 and the average number of received AeroTP PDUs is 140. The number of forwarded packets at the relay node is 119. This experiment shows a packet delivery ratio of 70%. These experiments show that the multihop functionality AeroNP and AeroRP work correctly.

VI. Conclusion

In this paper, we present a Python implementation of AeroTP, AeroRP, and AeroNP protocols. The implementation has been tested with several basic scenarios. In addition, we implement 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. We use Nokia N810 and N900 with real GPS devices using remote-controlled cars for our testing phase. The results show correct functional implementation of the protocols. For future work, we plan to run more experiments and several runs for each scenario to gain more confidence on the results.

Acknowledgments

We note that some texts, presented in this paper, are adapted from previously published papers.[9–11] 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]Rohrer, J. P., Jabbar, A., Çetinkaya, E. K., and Sterbenz, J. P., "Airborne Telemetry Networks: Challenges and Solutions in the ANTP Suite," *Proceedings of the IEEE Military Communications Conference (MILCOM)*, San Jose, CA, November 2010, pp. 74–79.

[2]Rohrer, J. P., Jabbar, A., Çetinkaya, E. K., Perrins, E., and Sterbenz, J. P., "Highly-Dynamic Cross-Layered Aeronautical Network Architecture," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 47, No. 4, October 2011, pp. 2742–2765.

[3]Rohrer, J. P., Jabbar, A., Perrins, E., and Sterbenz, J. P. G., "Cross-Layer Architectural Framework for Highly-Mobile Multihop Airborne Telemetry Networks," *Proceedings of the IEEE Military Communications Conference (MILCOM)*, San Diego, CA, November 2008, pp. 1–9.

[4]Peters, K., Jabbar, A., Çetinkaya, E. K., and Sterbenz, J. P., "A Geographical Routing Protocol for Highly-Dynamic Aeronautical Networks," *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, Cancun, Mexico, March 2011, pp. 492–497.

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

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

[7]Rohrer, J. P., Pathapati, K. S., Nguyen, T. A. N., and Sterbenz, J. P. G., "Opportunistic Transport for Disrupted Airborne Networks," *Proceedings of the IEEE Military Communications Conference (MILCOM)*, Orland, FL, November 2012, pp. 737–745.

[8]Çetinkaya, E. K., Rohrer, J. P., Jabbar, A., Alenazi, M. J., Zhang, D., Broyles, D. S., Pathapati, K. S., Narra, H., Peters, K., Gogi, S. A., and Sterbenz, J. P. G., "Protocols for Highly-Dynamic Airborne Networks," *Proceedings of the 18th ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, Istanbul, August 2012, pp. 411–413, Extended Abstract.

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

[10]Alenazi, M. J., Çetinkaya, E. K., Rohrer, J. P., and Sterbenz, J. P. G., "Implementation of the AeroRP and AeroNP Protocols in Python," *Proceedings of the International Telemetering Conference (ITC)*, San Diego, CA, October 2012.

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

[12]Rohrer, J. P., Perrins, E., and Sterbenz, J. P. G., "End-to-End Disruption-Tolerant Transport Protocol Issues and Design for Airborne Telemetry Networks," *Proceedings of the International Telemetering Conference (ITC)*, San Diego, CA, October 2008.

[13]Jabbar, A., Perrins, E., and Sterbenz, J. P. G., "A Cross-Layered Protocol Architecture for Highly-Dynamic Multihop Airborne Telemetry Networks," *Proceedings of the International Telemetering Conference (ITC)*, San Diego, CA, October 2008.

[14]Çetinkaya, E. K. and Sterbenz, J. P. G., "Aeronautical Gateways: Supporting TCP/IP-based Devices and Applications over Modern Telemetry Networks," *Proceedings of the International Telemetering Conference (ITC)*, Las Vegas, NV, October 2009.

[15]Pathapati, K. S., Nguyen, T. A. N., Rohrer, J. P., and Sterbenz, J. P., "Performance Analysis of the AeroTP Transport Protocol for Highly-Dynamic Airborne Telemetry Networks," *Proceedings of the International Telemetering Conference (ITC)*, Las Vegas, NV, October 2011.

[16]Peters, K., Çetinkaya, E. K., and Sterbenz, J. P. G., "Analysis of a Geolocation-Assisted Routing Protocol for Airborne Telemetry Networks," *Proceedings of the International Telemetering Conference (ITC)*, San Diego, CA, October 2010.

[17]Pathapati, K. S., Rohrer, J. P., and Sterbenz, J. P. G., "Edge-to-Edge ARQ: Transport-Layer Reliability for Airborne Telemetry Networks," *Proceedings of the International Telemetering Conference (ITC)*, San Diego, CA, October 2010.

[18]Narra, H., Çetinkaya, E. K., and Sterbenz, J. P., "Performance Analysis of AeroRP with Ground Station Updates in Highly-Dynamic Airborne Telemetry Networks," *Proceedings of the International Telemetering Conference (ITC)*, Las Vegas, NV, October 2011.

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

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

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

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

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

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

[25]Krasnyansky, M. and Yevmenkin, M., "Virtual point-to-point (TUN) and ethernet (TAP) devices," 2006.

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

[27]Fu, Z., Greenstein, B., Meng, X., and Lu, S., "Design and Implementation of a TCP-Friendly Transport Protocol for Ad Hoc Wireless Networks," *Proceedings of the 10th IEEE International Conference on Network Protocols*, November 2002, pp. 216–225.

[28]Fu, Z., Zerfos, P., Luo, H., Lu, S., Zhang, L., and Gerla, M., "The Impact of Multihop Wireless Channel on TCP Throughput and Loss," *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2003, pp. 1744–1753.

[29]Prechelt, L., "An Empirical Comparison of Seven Programming Languages," *IEEE Computer*, Vol. 33, No. 10, 2000, pp. 23–29.

[30]Sterbenz, J. P. G., Medhi, D., Monaco, G., Ramamurthy, B., Scoglio, C., Choi, B.-Y., Evans, J. B., Gruenbacher, D., Hui, R., Kaplow, W., Minden, G., and Verrant, J., "GpENI: Great Plains Environment for Network Innovation," `http://wiki.ittc.ku.edu/gpeni`, November 2009.

[31]Sterbenz, J. P. G., Medhi, D., Ramamurthy, B., Scoglio, C., Hutchison, D., Plattner, B., Anjali, T., Scott, A., Buffington, C., Monaco, G. E., Gruenbacher, D., McMullen, R., Rohrer, J. P., Sherrell, J., Angu, P., Cherukuri, R., Qian, H., and Tare, N., "The Great Plains Environment for Network Innovation (GpENI): A Programmable Testbed for Future Internet Architecture Research," *Proceedings of the 6th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom)*, Berlin, Germany, May 2010, pp. 428–441.

[32]"PlanetLab," `http://www.planet-lab.org/`, November 2009.

[33]Spurrier, N., "FSM in Python," `http://www.noah.org/python/FSM`, 2002.

[34]Brown, A., "Reed-Solomon class of error correcting codes in Python," `https://github.com/brownan/Reed-Solomon`, 2009.

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

[36]Camp, T., Boleng, J., and Davies, V., "A Survey of Mobility Models for Ad Hoc Network Research," *Wireless Communications and Mobile Computing*, Vol. 2, No. 5, 2002, pp. 483–502.

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

[38]"Maemo Linux Platform," `http://maemo.org`, 2012.

[39]"The ns-3 Network Simulator," `http://www.nsnam.org`, July 2009.