MOBILE AD HOC NETWORKS (MANET) PROTOCOLS EVALUATION FRAMEWORK

Vadim A. Slavin
Michael Polyakov
Mark Quilling
Lockheed Martin Space Systems
Sunnyvale, CA

Mike Wittie
University of California
Santa Barbara, CA

Matthew Andrews
Lucent Technologies
Murray Hill, NJ

ABSTRACT

We propose a novel, robust MANET protocols evaluation framework which enables researchers to track performance metrics and evaluate theoretical predictions. This framework speeds up the research and development spirals, provides faster feedback to algorithm developers and closes the loop between theory and qualitative analysis of the protocols' performance. Our test and evaluation effort is divided into two parts. Rapid prototyping and evaluation of proposed algorithms is performed in the MATLAB environment. These tools enable us to numerically analyze performance, capabilities, convergence, and robustness of new algorithms. The second higher fidelity approach is the test and evaluation framework developed in OPNET simulation environment. Its unique features are the novel application and evaluation process including sophisticated statistics collection and an event logging architecture.

INTRODUCTION

We propose a novel, robust MANET protocols evaluation framework as part of our Mobility-Aware Resource Coordination for Optimization of Network Infrastructure (MARCONI) effort to research, develop and evaluate a revolutionary Mobile Ad Hoc Network (MANET) prototype. The project requires radical rethinking of a wireless networking stack and has already led to prototyping and evaluation of new protocols. This collective effort spans a distributed team of researchers working together to translate groundbreaking theoretical research into significant performance gain over existing state of the art MANET.

In order to track our performance metrics and evaluate theoretical predictions, we have created an evaluation framework that speeds up the research and development spirals, provides faster feedback to algorithm developers and closes the loop between theory and qualitative analysis of the protocols' performance. Our test and evaluation effort is divided into two parts. Rapid prototyping and evaluation of proposed algorithms is performed in the MATLAB environment. Our tools developed in MATLAB enable us to numerically analyze performance, capabilities, convergence, and robustness of new algorithms.

The second higher fidelity approach is the test and evaluation framework developed in OPNET simulation environment. Its unique features are the novel application and evaluation processes we developed. These tools are independent of the type of networking stack being tested and thus allow for a direct comparison of various protocol iterations. The application module harness uses a scenario document easily created and imported into the simulation to allow for a flexible way of describing application scenarios from the tactical user's perspective. The statistics collection and logging framework we developed speed up the debugging cycle and help in evaluating the performance and the behavior of the new protocols.

PREVIOUS APPROACH

Network protocol development is a complex process riddled with design and implementation challenges. Assuring protocol correctness in all cases requires the programmer to not only understand the complexities of different parts of a protocol, but also gain insight into the interaction of the protocol within the network stack [1].

Furthermore, a distributed protocol development effort, while already challenging in its design stage, can be even harder during implementation and debug stages.

Traditional software development methods, while successful at bringing the networking community a number of popular protocols, are not uniformly efficient in all possible types of development projects [2].

Typically, once a new protocol has been developed it needs to be simulated in a network simulation tool to evaluate its correctness and efficiency. OPNET Modeler is the industry standard for network modeling and

simulation. It is based on a series of hierarchical editors that directly parallel the structure of real networks [3].

The standard method for processing performance data in OPNET involves graphical depictions of numerous statistics collected during the running of a simulation. Although statistics can give us coarse information about the performance of a given set of protocols, they cannot say anything about *why* a given protocol performed as such or where the problems are.

One other complication is the distributed nature of attributes defining a simulation. A scenario is described by attributes scattered over process, node, scenario and global settings all acting together.

The application process module used in most standard models for packet-level traffic simulation. It is inflexible and strongly coupled with other standard process modules that many users choose to replace.

Finally, OPNET's basic statistics gathering architecture is insufficient to evaluate and explain large scale behavior. Though it offers a powerful event driven debugger, as the number of simulated nodes rises, it becomes very hard to keep track of events. It is difficult to store and compare events from different runs, or to customize their format [4, 5]. Although OPNET does allow one to generate a visual plot of various events, we found the OPNET capabilities insufficient.

The alternative analytic tool available to programmers is a protocol behavior log implemented as a series of console or file printouts. While useful for quick implementation checks, this approach is not viable for solving more complex problems due to size limitation on most platforms. Additionally, console output is difficult to search, and cannot be reused.

Our project started out like many other advanced network research projects. A network simulation tool (we chose OPNET) was an essential piece in our design. We assembled other tools for future debugging and evaluation that are mentioned above. However, having found the traditional way inefficient for our goals we invested our efforts in designing a more developed and streamlined process for simulation, evaluation, and debugging our protocols and algorithms. We believe that our approach has lead to a shorter development and evaluation cycle with a smaller team.

RAPID PROTOTYPING

The MATLAB environment provides an interface to easily script a prototype algorithm. Using this tool we can quickly code the approximation of algorithms derived from the theory and evaluate how well they perform [6]. The main purpose of this effort is to provide a numerical basis of confidence by solving the underlying optimization problem which guides further protocol development.

When creating the theory that underlies a network stack design we start with the statement of NUM (Network Utility Maximization) optimization problem which encapsulates the network utility and constraints. Our objective is to maximize the user perceived utility subject to the constraints on resources. We can specify, through optimization decomposition (OD) [7], an optimization problem for each component throughout the network stack. These optimization problems define coupling and information sharing requirements between different elements of the network. The downfall is that most of these problems are either NP hard or need to be solved in a centralized manner: thus our need for decentralized approximate solutions. There are many ways (heuristics) to find a solution to these problems; our aim is to find the one that does it the best. We can easily test and modify existing and new algorithms until we find one that suits our needs. This ability is very useful and helps us find algorithms that perform quite well.

The framework we developed consists of a main loop that steps though events. Each event can consist of position change, flow arrival or departure, or change in QoS. For each of these events we run an inner loop, which is on a small time scale, to simulate the packet exchange across the network. Throughout this inner loop we assume the node positions and applications remain fixed.

For every event there are several modules that get executed. The first one is responsible for the network scenario, spatial distribution of nodes and their mobility. It also describes the types of flows that enter the network and their destinations. These are scriptable parameters which can be adjusted.

The rest of the modules are responsible for implementing four major components: source rate control, routing, power control, medium (channel) access and flow scheduling. The choice of schemes that implement the above processes determines the type of network stack we simulate.

The flows are simulated not as discrete packet flows but as continuous streams. This approximation allows us to model the system quicker and test the concepts which are the foundations of the new algorithms.

We also compare the performance of a new algorithm such as priority based random access. We can show numerically the potential gains we are likely to get by implementing such an algorithm in OPNET. Because of the low fidelity of MATLAB we will most likely see less improvement when implemented in OPNET but it is a reasonably good predictor.

We visualize our results as graphs of various parameters of the network as compared to the control set of components. We implemented a basic 802.11 scheme for each of the variable components in our framework to gauge the improvements derived from our theory. For every simulation, we can run our both sets of components for the same flow and node distribution and mobility scenario. [Fig 1]

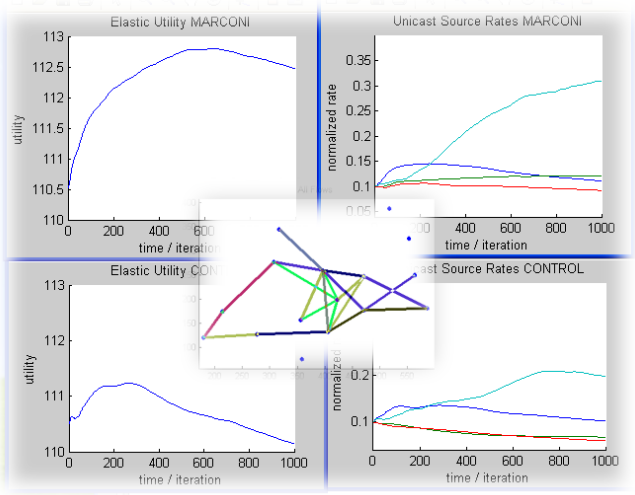

Fig. 1. MATLAB Framework Simulation Results.
Here, total network utility (left column) and unicast source rates (right column) network parameters are compared for a simulation using our set of components (top row) vs a control stack (bottom row). Clearly, we see improvement over the control stack. The center graph is the routing topology visualization at some time step. The blue dots are nodes and each colored path marks a route for a different flow.

Another important capability, not related to simulating the network stack, is the visualization of topology, routing, and mobility. Our MATLAB framework is able to interface with OPNET and present routing and link visualization in a user friendly manner. We are able to import this data from OPNET simulation and visualize connectivity and how the OPNET routing protocol implementation behaves in the context of node mobility. This capability has been very useful in verifying our routing algorithms. [Fig 5]

OPNET SIMULATION ENVIRONMENT

As we built our framework, we added several components to the simulation's suite of input parameterization and output visualization tools. We augmented OPNET data collection mechanism with a logging infrastructure to systematically collect non-numeric data and introduced global statistics specific to the wireless application flow tracking. While OPNET provides a sophisticated application process that can be used with its standard node models, we were compelled to implement our own lightweight xml-driven application process that targets the setup of application flows from the perspective of the tactical user.

A.Logging and event data collection

The OPNET framework did not provide a method of collecting non numeric data in an organized fashion. Out of the box it supports a combination of statistics (recording some value through time) and a variety of debug print functions. The OPNET debugger in concert with the Microsoft Visual Studio environment allowed detailed tracing of behavior, potentially stepping through code line by line, a method very effective for debugging *code*. However, these capabilities are less useful in analyzing algorithm and protocol behavior as a whole, particularly in large scenarios, for which a larger-scope view is desirable.

Our logging infrastructure provides a powerful way for any module in the simulation to report important events – whether this is route table changes, traffic flow beginning or something more fine grained. Unlike statistic collection it is able to collect arbitrarily complex event objects. Each event combines relevant information described by the developer: time stamp, initiator node id, packet id, or any other atomic piece of information relevant to the event.

From software development perspective, the logger class is a fully standalone module which can be defined by a test and evaluation team independent of the code to be analyzed. The responsibility of logging events is left up to the protocol developers who use the logger features via a single static function throughout their code.

Once created and logged (via a static initialization function), the events can be outputted into a variety of formats: a command window output stream, a plain text file or an XML file format, or an excel spreadsheet in tabular form. There is a robust inheritance hierarchy in place which allows sub-classing of logging events. For example, a general routing event can have other children: route discovery initiation event, a node's routing table

update event, routing packet receipt event, etc. This allows complex filtering to allow the experimenter to focus on particular types of events. This can be done in one central location regardless of the complexity of the rest of the system.

The power of such flexible output is apparent. Once the events are placed into spreadsheets, they can be further filtered and sorted by time, or by any other field. With the output of just one simulation the data can be analyzed chronologically then per wireless node, etc.

B.Application process with xml scripting of scenarios

To test network behavior at the packet level of granularity, OPNET provides a rich application module, able to simulate a dozen known application traffic patterns, as well as custom applications. While we initially pursued this line, several problems emerged.

Our project relies on flexibility of simulating various traffic flow patterns to test many specific features of the new protocols from the perspective of the end user. For example, 'a voice flow at 8kbps should be sent with high urgency and quality of service demands to a group of receivers'.

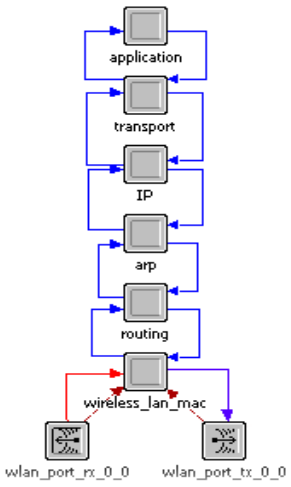

Fig. 2. MARCONI Application Harness.
The pictured networking stack is modeled in OPNET and includes our own application process. The plug-an-play interface of the application harness enables integration with different stacks to be evaluated against each other's performance.

In OPNET native application module, this requires translating application behavior into traffic patterns, changing application parameters accordingly and saving these profiles for use by other nodes – a cumbersome process. Modifying these patterns might be a time consuming task even for a small behavior change.

In addition, OPNET's application module is tightly integrated with other modules and processes in the stack model which makes it virtually inextensible. Because our project required a complete overhaul and rewrite of the existing network stack pieces, the standard application module was not suitable.

More importantly, however, is OPNET's native application's inability to describe application flows from the perspective of the network end user. OPNET requires the developers to introduce an extra step of translating application flow behavior into a traffic pattern description. The theory and the design of our project rely heavily on the specifications of the needs of the tactical user. Therefore, the evaluation of the project requires the same approach.

As a solution, we designed a more light-weight application suite as an OPNET module. An application dispatcher interfaces with the protocol lower in the stack, the transport protocol [Fig. 2], and starts child processes for applications when necessary. We currently support the following application types:

1. File Transfer: transferring files of specific size
   There are no constraints on the service, and throughput and delay are allowed to vary arbitrarily, as long as the file takes to be delivered. The initiator side chooses a file size to transmit and schedules itself to transmit packets periodically until done. The receiver simply records them.
2. Chat: sending text bursts
   The chat application transmits two-way low data rate bursty traffic. We script the initiator task to start at a particular time and the receiver starts responding with its own flow of data once it receives the first packets thus initiating a chat conversation.
3. Voice: sending VoIP
   This inelastic application implements a non-trivial rate constraint and specifies a tight delay constraint. Its operation and traffic patterns are similar to chat, though of higher bandwidth.

Application profile description has been moved into an xml script read by the dispatcher at runtime. Because, the XML script conforms to an xml schema we designed, the process of composing an xml script is thus very interactive when an xml editing tool is used. Most common xml editing tools provide predictive contextual attribute suggestion, so when a user starts typing the xml script, the

xml tool will suggest the allowed attributes depending on the context.

The script schema allows us to design application profiles to mimic the behavior of each application that a tactical user (i.e. warfighter, soldier) would be using minute by minute. In the future, we plan to also model video streaming applications, short command and situation awareness messages. Together with the above application types we already have implemented these are the applications typically used by a warfighter [9]. For each simulated user (node) in the network each application type loads a corresponding (by type) application profile. This can be the same application profile for all users or an individual one. Our application scenario description can thus be very general or very granular depending on the requirements of the test cases. In addition, this approach satisfies our desire for a single location where multiple applications could be easily scripted and their profiles saved for distribution to others [Fig. 3].

```
</ApplicationProfile>
<ApplicationProfile id="first_initiator" type="chat">
    <name>chat app</name>
    <description>profile description</description>
    <task id="task001" priority="6"  mode="initiator">
<!-- comments: this is the first chat initiator -->
        <destination>224.0.0.1</destination>
        <start_time>15</start_time>
        <end_time>55</end_time>
        <behavior>
            <packet_size distribution_name="normal"
                                    mean_value="600" sigma="0.4"/>
            <packet_frequency distribution_name="normal"
                            mean_value="100" sigma="0.4"/>
        </behavior>
    </task>
</ApplicationProfile>
<ApplicationProfile id="u2" type="voice">
```

Fig. 3. XML Script example.
Each Application profile describes a particular application's sequence of tasks and their behavior as it would be observed by the user. It can be scripted using a predefined schema to define the behavior of the simulation as well as provide a clear story to a human reader. Here the chat application is defined to start a flow at 15 secs and finish it at 55 secs while sending approximately (defined by the normal distribution) 100 packets per second each of size about 600 bits.

C. Global Statistics

Existing OPNET capabilities include a powerful statistic collection engine. This allows logging of data and its subsequent statistical analysis. Once we implemented our novel protocols as process models of the node stack in OPNET environment we also defined statistics specific to these protocols.

One of these process models was our application process model. To support modeling of inelastic (with-constraints) flows, we have created local and global statistics that keep track of QoS requirements satisfaction by application flows of interest.

a. Flow Periods Valid/Invalid: for every custom defined time period (default is 1 second) the system records the total number of flows, per flow type, that met (or did not meet) their requirements

b. Number of Receivers per Flow Type: number of receivers recorded per flow type at each time step. This statistics shows when flows were added, refused, removed or preempted.

c. Goodput Per Flow Type: bit rate of the application data delivered to the destination as opposed to the bit rate of the total data transferred.

d. Received Bit Rate that is higher than min throughput rate.

While by itself these features do not constitute an innovation they augment our application profile simulation. The goal of these statistics is to present a clear picture of the behavior of the application flows within a system.

Because the statistics are applied to the application process and describe end-to-end behavior of the traffic flows, these statistics are not dependent on the underlying networking stack modeled. Therefore, the same application harness together with the defined statistics can be applied to many different networking stacks for a fair comparison of their efficiency.

THE NEW PROCESS

The tools described above have introduced a considerable improvement to our research process. We have optimized the key components of our process: theory validation, unit testing and debugging, system testing and analysis. Here we give an example of the new process we have instituted.

The research effort is broken up into several Spirals. Each spiral includes the development of a new piece of theory, its analysis and subsequent implementation and testing. At the end of each spiral we release a code base which we evaluate to isolate the performance improvements.

The starting point of each Spiral is the draft of theoretical innovations that would be needed to improve the performance of the MANET from the perspective of a tactical user. The performance improvement we consider would be the improvements in network throughput, reduced latency, and greater reliability as perceived by the end user – the war fighter.

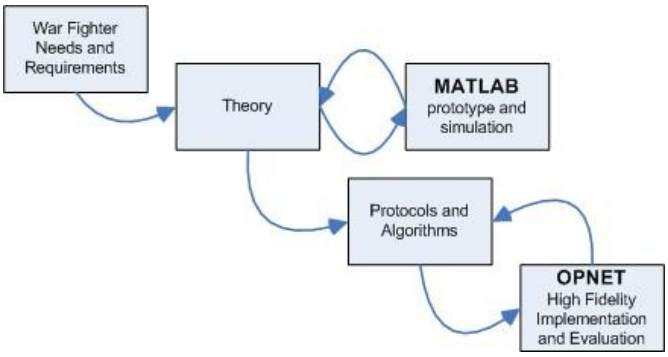

Fig. 4. The 'waterfall' process which describes each spiral of our research project. End user requirements drive the theory behind our protocols which is then quickly evaluated using the MATLAB tools we developed. The Protocols and new features result. A higher fidelity simulation completes the process.

Based on the draft of the new features the theory team comes back with the high level preliminary description of the protocols and algorithms as they apply to the general theory of network optimization as well as our project's theoretical assumptions. These algorithms are then implemented in our MATLAB environment. Our tools developed in MATLAB allow us to quickly evaluate, at a low fidelity, the sensitivity, robustness, optimality, and convergence of these algorithms which potentially drive the protocol development.

The results of the quick simulation are communicated to the theory teams and the loop is closed. At this point the theory team produces more detailed design for the new protocols or protocol improvements. A more thorough, higher fidelity protocol implementation then takes place in our OPNET simulation environment.

We use the tools described above to test each new atomic piece of functionality before we release the code for evaluation. We use our application module to quickly build a few simple test application scenarios using specifically geared towards showcasing the functionality we have just implemented. We then run the scenarios in OPNET and utilize a detailed logging structure that helps analyzing and debugging new protocols.

The output files of the loggers help share our findings with other members of our dispersed team to explain a bug in the code or even a design flaw.

This component test effort not only helps us flush out code bugs but also discover protocol behavior inconsistencies. We categorize the problems as implementation bugs or as design bugs. The logging and tracing outputs can be shared with our teammates responsible for design. These materials present a clear picture of protocol events and serve as bug reporting materials as well.

Another example is our MATLAB tools built specifically for visualization of Routing topology established as a result of the simulation run. We are able to replay the simulation as a movie watching for the available connectivity and route establishment as a result of this connectivity [Fig 5].

Once the bugs are fixed we are ready for the final stage of the spiral – evaluations.

Again we employ our application module to design more complicated and more realistic scenarios. We design the node mobility and each node's application behavior as it would be seen by each user. Each of the applications define their own behavior as specific as "send a voice message at 10 seconds to multicast IP 224.0.0.1" or "reply to all incoming voice traffic for IP 224.0.0.2".

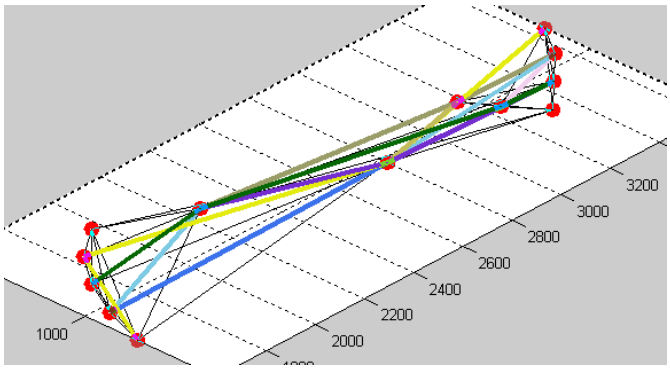

Fig. 5. MATLAB Routing Topology Visualization.
Here the links between nodes are shown as thin grey lines. The thicker color coded lines are routes at a particular time instant. The tool can be played out as an animation or stepped through chronologically.

We then apply these application attributes to both the new network stack we are evaluating and the baseline network stack we chose at the beginning of the project as our stating point. We thus run two simulations for each setup. We then gather local (per node), and global (per network) statistics outlined above to see what performance gains we notice as a result of our innovations. Furthermore, specific features can be turned on and off to pinpoint the improvement results and tie them to specific innovations. At our discretion, we also run a third simulation for the same setup on the networking stack resulting form the previous spiral. This serves as regression testing and aids our analysis of features contributing to the performance gains.

We are confident in the fair comparison to the baseline stack because we employ the same application harness with exactly the same setup attributes to run the simulation for the current stack and the baseline stack.

The simulation results guide the prioritization of features for the next spiral because the most effective theoretical innovations are given higher priority.

CASE STUDY

In this section we will take a representative siege scenario and evaluate it in MATLAB and then in OPNET. As part of the MARCONI program objective we are required to provide "equivalent performance at 10% of the bandwidth" so our evaluation method has to reflect the effect of bandwidth on performance. For each scenario we evaluate the bandwidth is reduced until the stack can no longer support the load, we call this the saturation bandwidth. We apply this method to both the MARCONI stack and a representative Control stack; the ratio is the percent of bandwidth which we achieved equivalent performance.

We will consider a scenario consisting of 49 nodes in 7 groups surrounding a target [Fig 6]. We use a nominal bandwidth of 20 Mhz, a data rate of 54 Mbps, and a transmit power of 50 mW. This equates to a reach of about 1.7 km, for each node, at the nominal bandwidth. The groups are positioned in such a way so that they can only communicate with adjacent groups.

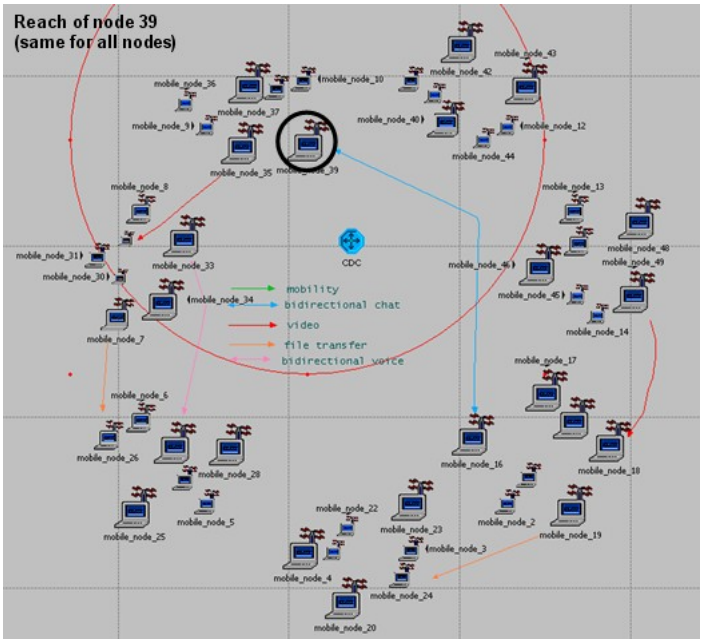

Fig. 6 Scenario layout

We have set up 6 flows in the network; 2 inelastic video with a min rate of 500 kbps, 1 inelastic chat flow with a min rate of 28 kbps, 1 inelastic voice flow with a min rate of 80 kbps, and 2 elastic file transfers. The network is simulated for 60 seconds with the flows starting at different intervals (last flow starts at 30 seconds).

When we evaluate the performance of the stacks on the above scenario we have two metrics we are concerned with. The elastic utility is the sum of the logs of each of the elastic flows (in bps). For inelastic flows the utility is the sum of the valid flow periods (or brownie points); a flow receives a brownie point if it reaches it destination at or above 90% of its min rate.

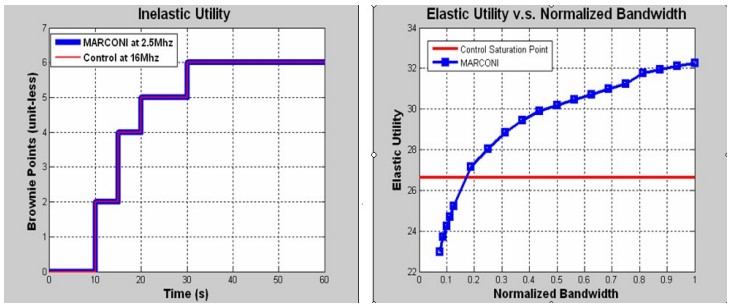

Fig. 7 MATLAB simulation results, (left) inelastic utility, (right) elastic utility.

The MATLAB results are shown in [Fig 7]. The inelastic utility (left) for the MARCONI and Control Stacks are plotted for each of the respective saturation bandwidths. The elastic utility (right) shows MARCONI's elastic utility plotted against the fraction of the Control stacks bandwidth. The threshold line (red) indicates the utility the Control stack achieved at saturation. What these two plots tell us is that MARCONI was able to carry the offered load at 2.5 Mhz where the Control needed 16 Mhz. This equates to equivalent performance at about 16% of the bandwidth. These are great results but have a few caveats due to the implicit low fidelity of the MATLAB simulation tool.

Our next step, once we have verified that the proposed algorithms perform well, is to determine the changes that need to be made for a real world implementation. The performance in MATLAB gives us somewhat of a best case of how good the proposed algorithms can perform. Once we move to OPNET modifications and approximations must be made in order to implement them as protocols.

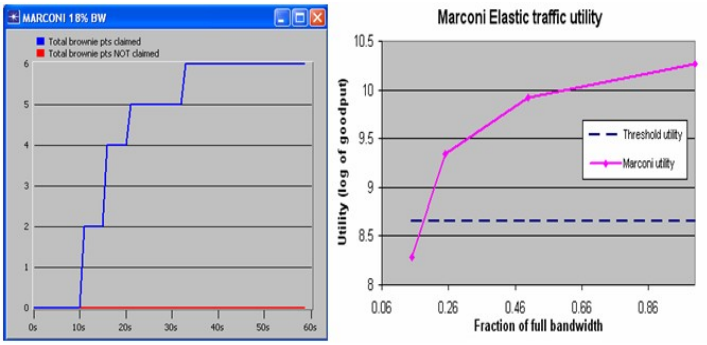

Fig. 8 OPNET simulation results, (left) inelastic utility, (right) elastic utility.

The OPNET results shown in [Fig 8] are very similar to those shown in MATLAB. The MARONI and Control stack were saturated at 4.8 Mhz and 19 Mhz, respectively. This equates to equivalent performance at about 24% of the bandwidth. The discrepancies between the elastic utility plots is due to the fact that in OPNET we used a log base 10 and in MATLAB we used a natural log.

The MATLAB prediction was better than what we found in OPNET for a few reason. The first is we don't model signaling delay in MATLAB, secondly we are changing from a flow based model to a packet based model. There are other, less influential, factors but the above two are the main contributors to the discrepancies. Because, in some cases, we are making approximations to the theory derived algorithms in OPNET it may be beneficial to return to the MATLAB platform and evaluate the different possible ways of implementing a specific algorithm. This feedback loop is much less prominent than the one between the theory and MATLAB because of the difficulty of implementation in OPNET.

This case study clearly shows how we use MATLAB to find and evaluate potential protocol algorithms. These algorithms are molded in MATLAB till they have the desired properties to present a feasible real world implementation. This implementation can then be coded in OPNET and eventually move to a real radio.

FUTURE WORK

We plan to add a few more capabilities in the near future. To begin with, we intend to implement a few other application types: video streaming, short messaging, and situation awareness messages. One other idea we have been nurturing is to implement a central run-time application profile distributor to allow batch mode execution of multiple simulations with different (e.g. randomized) traffic profiles. We hope to create a central modeling process that can allocate application profiles to nodes at runtime based on a single configuration. This process will read a master script that describes probabilistic distributions specifying which nodes may run which application profiles and with which parameters. This will greatly enhance our ability to run sensitivity and confidence tests.

Today's testbeds for distributed autonomous systems tend to be limited to simulation validation at one extreme or hardware platform demonstration at the other end. Neither of these approaches does justice to validation and testing of the decentralized control and sensing software systems that lie at the heart of the distributed systems such as MANETs. A network emulation testbed would provide the right infrastructure for development and evaluation of the networked control and sensing software which is at the core of our MANET stack implementation.

Once we finish our initial research effort, we plan to implement such a network emulation testbed using a Linux cluster. One central machine would emulate the wireless channel and the rest would serve as the virtual nodes in the network. This would allow us to perform even higher fidelity evaluation of our concepts, protocols, and algorithms before moving to a field demonstration. Particularly, we would be able to test the hardening of the system: i.e. the robustness to network failures, delays, instabilities, outages, etc.

CONCLUSION

We have presented our novel, robust MANET protocols evaluation framework which has enabled us to dramatically speed up the research and development cycle of our effort, improve the efficiency of the theory to protocol cycle iteration, and otherwise increase the productivity of our research team spanning over 6 public and private research institutions.

Our rapid prototyping framework in MATLAB has enabled us to numerically analyze performance, capabilities, convergence, and robustness of a new network stack before a more thorough implementation effort is required.

Our higher fidelity simulation and evaluation framework has enabled us to test the network stack programmatically and with higher accuracy. We believe it has enables us to discover the design and implementation flaws much faster than otherwise would be possible. This has contributed to the overall efficiency of our research effort.

While the true performance of our new algorithms remains to be proven in emulation and field test environment, we are confident that we will see significant improvement to the existing state-of-the-art MANET systems. This outlook is based on the evaluations of our protocols from the point of view of the end user – an approach not possible with the same ease and efficiency before our tools were developed.

ACKNOWLEDGEMENT

This material is based upon work supported by the Defense Advanced Research Projects Agency, and the Space and Naval Warfare Systems Center, San Diego, under Contract No. N66001-06-C-2021. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency for the Space and Naval Warfare Systems Center, San Diego.

REFERENCES

[1] D. Lapsley, M. Bergamo, "*An integrated approach to the development of wireless network protocols*", in *Proc. of the 1st international Workshop on Wireless Network Testbeds, Experimental Evaluation Characterization* ,Los Angeles, CA, USA, 2006). WiNTECH '06. ACM Press, New York, NY, 10-17.

[2] David Cavin, Yoav Sasson, Andre Schiper, "*On the accuracy of MANET simulators*", in *Proc. of the second ACM international workshop on Principles of mobile computing*, New York, United States 2002, pp. 38–43

[3] *Modeler Wireless Suite for Defense* [web page]. 2006 OPNET Technologies, Inc. Available: http://www.opnet.com/solutions/network_rd/modeler_wireless_defense.html

[4] X. Chang, "*Network simulations with OPNET*", in *Proc. of the 31st Conference on Winter Simulation: Simulation---A Bridge To the Future - Volume 1*, Phoenix, Arizona, United States, 1999.

[5] Varshney, M., Xu, D., Srivastava, M., and Bagrodia, R., "*SenQ: a scalable simulation and emulation environment for sensor networks*", in *Proc. of the 6th international Conference on information Processing in Sensor Networks*, Cambridge, Massachusetts, USA, 2007.

[6] Elphick, D., Leuschel, M., and Cox, S., "*Partial evaluation of MATLAB*", in *Proc. of the 2nd international Conference on Generative Programming and Component Engineering,* Erfurt, Germany, 2003.

[7] L. Bui, R. Srikant, A. Stolyar, "Optimal Resource Allocation for Multicast Flows in Multihop Wireless Network," unpublished.

[8] P. Gupta, Y. Sankarasubramaniam, and A. Stolyar, "*Random-access scheduling with service differentiation in wireless networks*," Bell Labs., Tech. Rep., May 2004

[9] Joe Leland**,** *Future Army Bandwidth Needs and Capabilities,* Rand Corporation, 2004.

[10] Mark Quilling, "Evaluating NUM and Optimization Theory Driven Next Generation MANET Architectures," Military Communications Conference (MILCOM), submitted for publication.