A New Simulation Architecture for Communication Network Systems with Application to HF Radio

Duncan M. G. Tait, Andrew F. R. Gillespie, and Dave Cliff

***Abstract*—This paper describes a simulation architecture developed to address the need in the defence and aerospace industry for a rapid prototyping capability to test new and existing network protocols and systems where analytical methods no longer suffice. In particular, the simulator introduced here allows study of High Frequency (HF) radio networks and other unorthodox systems that cannot be easily modelled using existing communications network simulators. This paper describes the architecture and gives an example use case – modelling an HF System using the 2G Automatic Link Establishment (ALE) linking protocol under a variety of conditions; exploring the effects of changing the number of nodes in the network, and of altering the input data rate to the system. The network performance was analysed under a wide range of combinations of conditions, and it was found that increasing the number of nodes in these networks causes specific latency increases and an overall throughput decrease, both in the unidirectional and bidirectional case, although it is better tolerated in the unidirectional case. Increasing the input data rate causes an overall throughput increase up to a threshold point, after which it saturates and then decreases as data remains unsent, due to the physical limits of the system (including networking overhead) being surpassed. This simulator, known as CommNetSim has been developed for use in evaluating the technical capabilities of, and business cases for, real-world applications to be developed by Thales UK.**

***Index Terms*—Communication networks, discrete event simulation, high frequency radio, modelling, simulation.**

I. Introduction

A challenge faced by industry is the modelling of large-scale and often heterogeneous communication network systems in a dynamic technological environment. Being able to reliably predict the performance of a system, either hypothetical or extant, adds great support to any business decision concerning the use of the system in a project or product.

Many platforms exist for simulating communication networks – these include commercially developed proprietary tools at a cost, open source frameworks and research tools [1]. The business needs of the organisation have to be taken into consideration in order to select the optimal platform for the environment it will be deployed into.

Manuscript received March 10, 2013; revised June 28, 2013. This work was supported by the EPSRC, The Systems Centre at University of Bristol and University of Bath, LSCITS and Thales UK.

D. M. G. Tait is with the University of Bristol, Bristol, UK, BS8 1UB and Thales UK (e-mail: cexdt@bristol.ac.uk)

A. F. R. Gillespie is with Thales UK, Crawley, W. Sussex, UK, RH10 9TS (e-mail: afr.gillespie@uk.thalesgroup.com).

D. Cliff is with the University of Bristol, Bristol, UK, BS8 1UB (e-mail: dc@cs.bris.ac.uk).

This paper aims to outline a simulation architecture developed to meet the needs of an industrial stakeholder, who develops and integrates communication network systems. The needs of the modelling capability were elicited through various interviews, meetings and general stakeholder engagement. These needs, and the design decisions they necessitated, are listed as follows:

- Fast prototyping of new protocols: the Python [2] programming language was used, a very high-level language that enables a large amount of functionality with concise coding to speed up development. The rationale was that coding and development time was more critical than processing time.
- Detailed analytics: a Discrete Event Simulation (DES) methodology was chosen, and a DES framework was used – a library for Python called SimPy [3]. This is the usual methodology for network simulation as it enables monitoring of the simulation at any desired level.
- Combinatorial modelling of systems – ability to test different combinations of configurations without large amounts of recoding: A modular architecture was chosen (Fig. 1), based on the Open Systems Interconnection model [4] (Fig. 2), the design goal was to allow swapping of modules at each layer of the OSI stack.
- Extensible to different forms of physical network: the architecture was made flexible enough to allow mechanisms for modelling multiple types of network bearers, for example Ethernet or wireless connections.

It was found that many of the proprietary platforms, whilst powerful, were too costly for occasional use. When developing models for novel protocols or equipment, time needs to be allowed for coding and debugging their behaviour to fit the APIs of the platforms. The language used is often relatively low level (e.g. C, C++) and thus requires more lines of code, and thus more time for development. In large companies this may not be a problem, but for smaller teams the time and resource investment can prove inefficient [5]. To this end it was decided that developing a new simulation architecture to allow for more rapid development, using an open source framework was worth the time investment as it can take into account all the stakeholder needs.

This Simulation Architecture, known as CommNetSim has since been developed as an extensible framework for use in evaluating the technical capabilities of, and business cases for, real-world applications developed by Thales UK.

The rest of this paper is structured as follows: Section II gives an overview of the simulation methodology developed for this project, including explanations of the protocol development process and the scenarios simulated; Section III

DOI: 10.7763/IJMO.2013.V3.291

describes the results of these scenarios, and Sections IV and V discuss and analyse these results. Section VI suggests further work for the expansion and improvement of this Simulation Architecture.

II. Methodology

A. Outline

A communication network system will be populated with nodes – and the connections between them will form the network. Fig. 1 shows the flow of information within each node; each container represents a layer in the node's processing system, and the structure is based upon the OSI model.

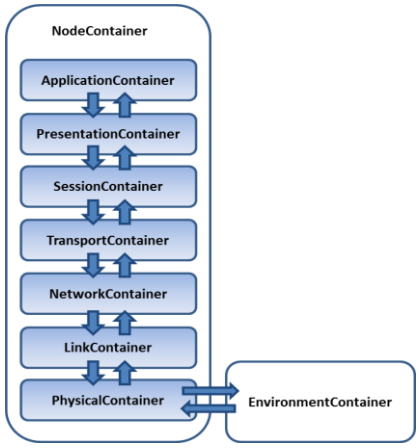

Fig. 1. Map of the information flow within each node – only adjacent Layers can communicate with one another, and only the Physical Layer can communicate with the Environment. In this map all Layers are depicted as Containers, as these are the structural mechanisms within the Simulation Architecture that contain the modules.

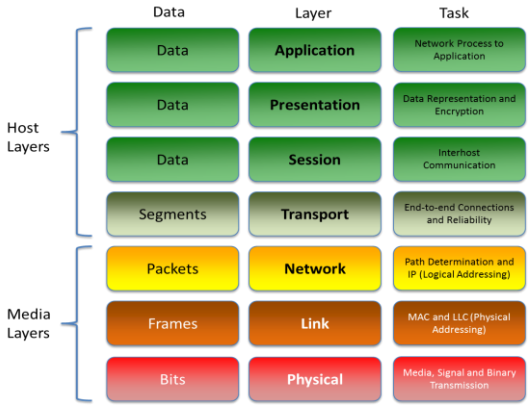

Fig. 2. Description of OSI model.

These layers interact hierarchically, passing data and internal messages up and down – the data being structured into segments, packets, and frames depending on the layer they are within. Each layer, in both the OSI Model and the software architecture, has 3 attributes:

1) Functions (what is the layer's responsibility)
2) Interface (how it is supposed to communicate with the layer directly above and below it)
3) Protocol (the rules on how to communicate with its peer layer on other network services).

The software architecture allows a separate Python module for each layer. The architecture has a container for each layer module, that all have a standard format. This can be thought of as an interface mechanism between the layers. There are two types of information that are passed throughout the node; the first is data that is ultimately to be sent to another node. The second is internal messages that remain within the node, and form the basis of communication between layers. These are both handled by the aforementioned interface mechanism, but remain separate systems.

Within each module there are processes that operate independently, each as a Finite State Machine (FSM). Together, these processes handle all the functionality of that respective layer, and form a complete module. During module design, the functions of each layer must be comprehensively catalogued, and then responsibility for those functions divided between the processes. How the functionality of a layer is divided up can almost always be done in multiple ways, and it is the author's opinion that functional blocks should be selected in order to aid understanding, rather than for efficient programming.

An event-driven internal messaging system facilitates interfacing between layers. To illustrate this, the architecture for three arbitrary layers is described in the text that follows, and is shown schematically in Fig. 3.

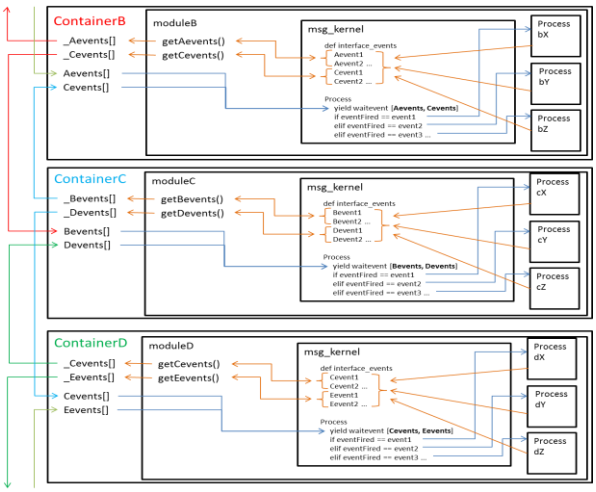

Fig. 3. Architecture of internal messaging system

Internal messaging is used in cases where no data needs to be sent or received externally to the node; an example might be a check to see whether a communication channel is vacant or not. There is a specific event for each possible message, and each Process may wait upon these events as needed. A good example to demonstrate how this works is the 'abort event' cascade, where a layer higher up in the hierarchy decides to abort what is currently occurring at the layer below it (as it has precedent to do). If a process decides to do this, it signals the corresponding event in the interface events, which is being watched by the msg kernel process in its neighbouring layer, which then passes the message down to any process within its module that is waiting upon it. Upon receiving this signal, the next behaviour for that Process should be to abort its current task.

Aside from the node structures, the other important part of the model is the environment module, which defines how the signals traverse the intermediate space between nodes. Fig. 5

shows the mechanism for this, for a generic model with multiple channels. The entire procedure is as follows: a process within a node's physical layer wants to transmit its message, so it sends it to the Environment's physical Q via an event payload. The environment then forwards it into the Environment process itself, where a new on air process is spawned that will take care of all activity to do with that message on the channel. After a propagation time, it puts the message into the list that corresponds to the channel and fires a channel taken event. This means any physical processes currently listening to that channel will be notified and able to receive the message. After the calculated time that the message is on the channel for, on air removes the message and fires the channel freed event before terminating itself.

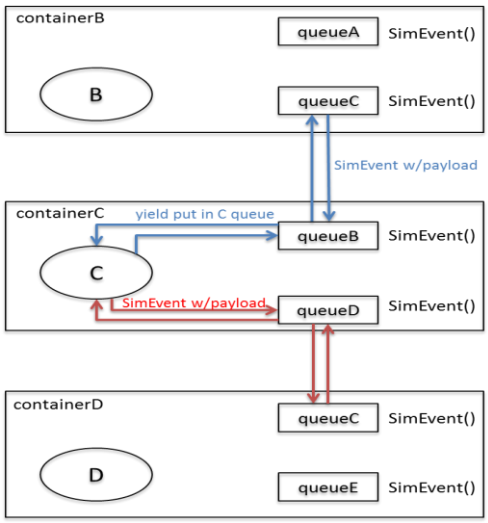

Fig. 4: Data Interface for an example container (ContainerC) and connections to its two adjacent Containers. These queues are event driven – the data is passed as a payload. A Process (e.g. C) will fire the queue event corresponding to the desired destination of the data (e.g. queueB) and attach the data to pass as the event payload. queueB will then fire the corresponding queue event (queueC here) in the neighbouring container, which will then relay the event payload to whichever Process is active within that container (e.g. perhaps B).

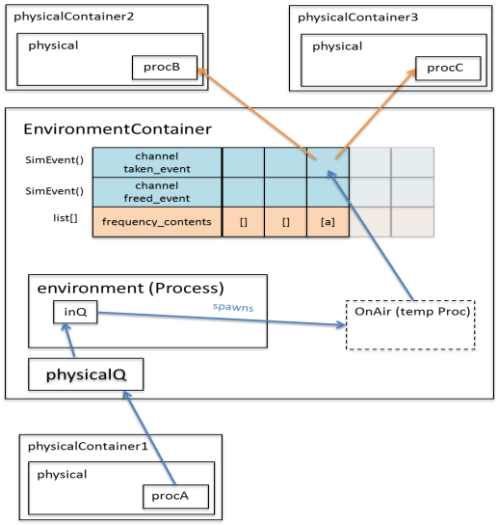

Fig. 5. A node sending a message via the environment, which is disseminated to all other nodes via their 'physical' containers. A Process (e.g. proc A) will fire the physical Q event in Environment Container that then relays the data to the environment process queue. This then creates a temporary process 'On Air' that handles the subsequent events – upon the data packet being placed onto a channel (frequency contents), a 'channel taken' event is fired that is detected by all physical containers to synchronise their behaviour. The same occurs for channel freed when the data packet comes off air, and the packet is removed from the channel.

CommNetSim has been used to model many different network scenarios. Primarily the type of systems modelled have been High Frequency (HF) radio networks. For this type of system, the Physical Layer module will be a model of the transmission equipment – antenna, modem, power amplifier etc. The Link layer will deal with the HF protocols for linking and data transfer, and the Network Layer, and all layers above, can remain generic.

B. Finite State Machines

All processes within modules can be simple or complicated, but all must adhere to the Finite State Machine (FSM) structure. Fig. 6 is an example of a process; the Linking Protocol for the Master node in a linking handshake, named ALE (Automatic Link Establishment) Master. Each progression of state is caused by some event, this may well be a Sim Event, or some other simulation occurrence such as an item being added to a queue.

This process is relatively straightforward: with each successful event in the handshake's itinerary, the process moves one step closer to being linked, and any failure event takes it back to the initial state (wait for packet). It is of critical importance that all processes are complete FSMs, and can successfully handle transitions resulting from all events at all times – even if outside the intended scope of operation, to prevent Byzantine failure modes.

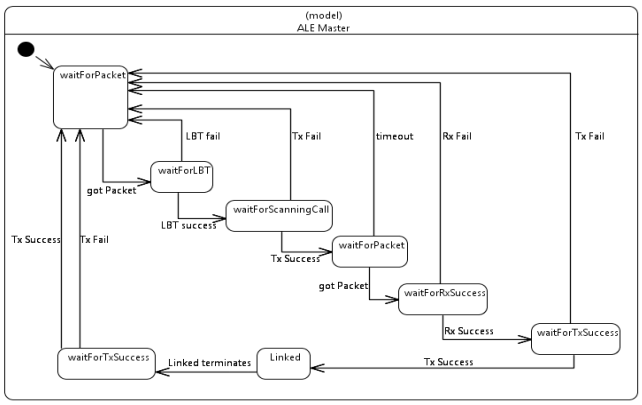

Fig. 6. Finite state machine diagram for master linking protocol.

C. Scenarios

A commonly implemented system configuration for HF radio communications is the use of a linking protocol defined by the standard MIL-STD 188-141B [6] in conjunction with a data link protocol defined by the standard STANAG 5066 [7]. This linking protocol is often referred to as 2G ALE (Automatic Link Establishment).

As an example use case of this Simulation Architecture, an investigation into the network behaviour of a 2G ALE HF system will be demonstrated. Two common variants in the configuration and operation of a radio system are the data rate it is required to run at, and the number of radio nodes in the network. Many Naval HF radio links are unidirectional point-to-point links, for example ship-to-shore communications – having only two nodes in the network. Full network behaviour is not often used as it is not required, although the protocol does support it. To reflect this, at least some of the results will be of unidirectional point-to-point scenarios, for comparison with scenarios exhibiting full

network behaviour.

To observe the effect of network operation on radio links, in particular the effect of different numbers of nodes running at different data rates, factors corresponding to these two parameters must be altered, while all others are held constant. The simulation must reflect this. Hence, the variant parameters are: input data rate, and number of nodes.

For a comparison of different data rates, sets of simulations are run – each with progressively larger input data rates. In order to implement a comparison of different numbers of nodes, all the aforementioned data rates are run for networks containing varying numbers of nodes (2, 4, 6, 8, 10 in this use case).

Due to network resources being limited, they are shared between multiple contesting nodes, which affects performance. The link-level protocols themselves will also affect performance by how well they handle a given scenario. As it will be difficult to differentiate the causes of varying performance, two tranches of simulation scenarios were undertaken. The first tranche tests full network behaviour – with all nodes attempting to transmit traffic to any other node, this can be termed a bidirectional scenario. The second tranche consists of dedicated master and slave nodes – with each master node transmitting only to a single slave node, this can be termed a unidirectional scenario. In a unidirectional scenario this means that although network resources such as number of available channels will still affect overall performance, nodes will not attempt to connect to nodes that are already busy in another link, thus testing the linking protocol separately from major network congestion effects (bidirectional scenario). Key parameters are as follows:

Number of channels: 10
Ratio of propagating channels: 0.6
Packet size: 1024 bytes
Data Rates in ARQ scheme: 75, 150, 300, 600, 1200, 3200, 4800, 6400, 8000, 9600 b/s
ARQ Frame length: 10 seconds
All values are the result of averaging over 500 runs.
Linking aborts after 300s if there is no successful link made.

III. Results

Linking latency with 2G ALE is often calculated analytically by using an algorithm and certain parameters such as scanning rate, number of channels and handshake timing values. Using multiple nodes however causes contention; both for channels and for availability of receiving nodes (as they may be busy with another link), this cannot be accounted for by algorithms. This is reflected in the results shown in Fig. 7: as the number of nodes increases, so do the median linking times. Additonally, as the input data rate increases, the latency increases further as nodes will spend longer 'On Air' per link and thus network contention increases.

In the Unidirectional case, the median latency is very similar for all scenarios – differing from 11.623 to 11.633 seconds, best to worst case respectively, which can be statistically accounted for solely by jitter. For this reason, the mean latency is plotted in order to differentiate performance based on the effects of statistical outliers on the average latency (Fig. 8). This shows the latency still increaseing with number of nodes due to channel contention issues, and also that both the *median* and *mean* latencies are lower in the unidirectional case, as the receiving node is guaranteed to be free.

All error bars are calculated from the Standard Error:

$$StErr = \frac{s}{\sqrt{n}} \quad (1)$$

As we are effectively running a Monte Carlo style simulation, the sample (500 runs) is an estimate of the true population – and the standard error is a good estimate of the standard deviation of the distribution (assuming the sample is sufficiently large).

The objective of a communication link is to transfer data – so an important metric is the amount of data successfully transferred per unit time (in this case, per simulation run which is 1000s), as a function of the amount of input data to the system. In a point-to-point link, uncontested, the amount of data transferred should be 100% of the input data (minus protocol overheads) up to the limits of the physical bearers.

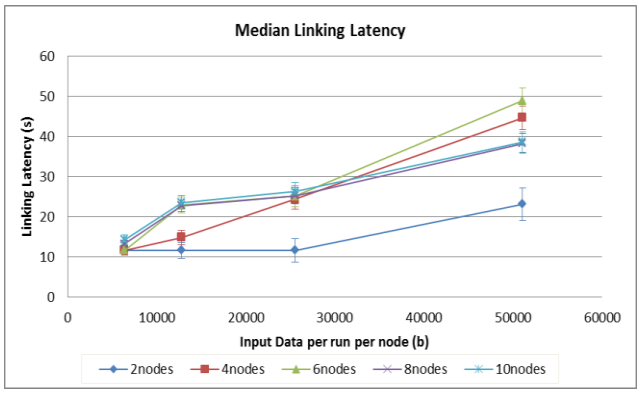

Fig. 7. Bidirectional median linking latency.

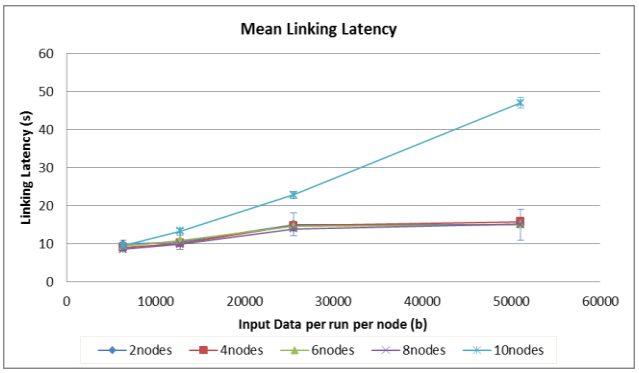

Fig. 8. Unidirectional mean linking latency.

Adding a network element, however, causes the realised data throughput to decrease – as can be seen in Figs. 9 and 10. At high input data rates, the throughput demands cannot be met and a significant percentage of data will build up, only to be left waiting in queues 'to be transmitted' at the end of each simulation. This can account for the drop off in Figs. 9 and 10 after 25kbits per run per node, as between 25k and 50kbits

per run per node, the network limit is exceeded is this case.

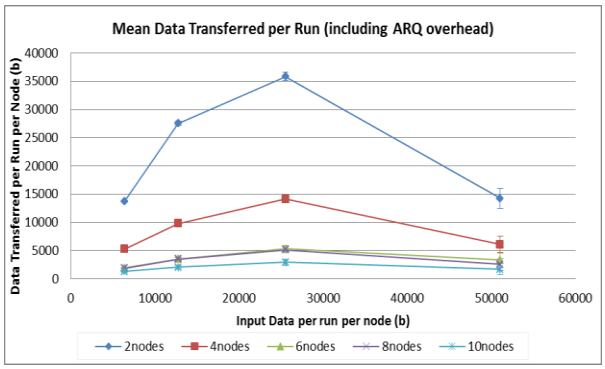

Fig. 9. Bidirectional mean data transferred per simulation run.

Unidirectional performance (Fig. 9) here is almost identical to bidirectional performance (Fig. 10), with better performance at higher numbers of nodes –a result of the lack of contention with busy nodes in the unidirectional case, which becomes more of an apparent effect with higher numbers of nodes.

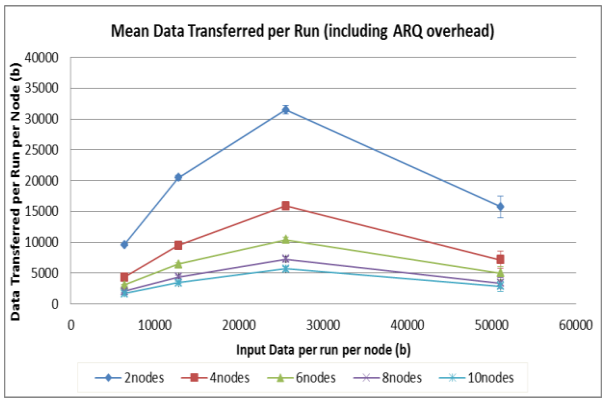

Fig. 10. Unidirectional mean data transferred per simulation run.

IV. Analysis / Discussion

Graphs like these allow the tipping-points of networks to be found - useful thresholds that can be used to better configure networks in order to optimise performance for given scenarios. For instance in Figs 9 and 10, there is an overall performance decrease in the system if the input data per node is above 25000b/s – the exact tipping point is between 25000 and 50000b/s for both bidirectional and unidirectional network systems, and can then be further investigated for each. Latency averages and ranges (Figs 7 and 8) are useful from a system configuration perspective – for instance to optimise timing parameters and minimise unecessary timeouts. As network based scenarios become more common in systems traditionally used for point-to-point links (the HF systems shown here are a prime example), examining the effect that network operation has on the performance of a system becomes increasingly important.

This Simulation Architecture can be used for a variety of different tasks. The use case demonstrated here has been for HF systems, but with a simple interchange of certain modules other systems can be modelled. In this case, the HF/ALE system was manifested in the Physical and Link layers that work in conjunction to model the HF physical and linking system. The layers above (i.e., Network up to Application) are generic and not specific to HF at all, they can be used with any other physical and link layer models. This simulation architecture represents a general purpose communication network simulation framework that has proven suitability in the HF radio network domain, and is capable of working with other communication domains.

V. Further Work

The Simulation Architecture itself is now relatively mature, and so tasks are primarily based around developing new models for the architecture, and then generating results to analyse [8]. Recent work has gone into improving the Environment module – by generating instantaneous signal-to-noise ratio (SNR) values for a given link, based on environmental parameters retrieved from data generated in VOACAP [9]. A forthcoming paper will address this in detail.

VI. Conclusion

CommNetSim is a useful tool for testing technical capabilities of communication networks within industry, and provides a lightweight, quick-to-employ alternative to network simulation incumbents, especially for simulating radio networks where other tools commonly lack the means to create and implement these protocols with ease.

References

[1] E. Weingartner, H. V. Lehn, and K. Wehrle, "A performance comparison of recent network simulators," in *Proc. IEEE International Conference on Communications*, 2009. pp.1, 5

[2] G. V. Rossum, "Python tutorial, technical report CS-R9526," in *Centrum voor Wiskunde en Informatica, Amsterdam*, May, 1995.

[3] K. Muller and T. Vignaux, "SimPy: Simulating systems in python," *ONLamp.com Python Devcenter, 2003*.

[4] H. Zimmermann, "OSI reference model--The ISO model of architecture for open systems interconnection," *IEEE Trans. on Communications*, vol. 28, no. 4, pp. 425-432, 1980.

[5] T. E. Oliphant, "Python for scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10-20, May-June 2007.

[6] MIL-STD 144-110C, *Interoperability and performance standards for medium and high frequency radio systems*, DoD. 1999.

[7] *NATO Standardisation Agreement 5066*, Profile for High Frequency (HF) Radio Data Communications, NATO.

[8] D. Tait, A. F. R. Gillespie, and S. E. Trinder, "Modelling 2G and 3G ALE: A quantitative comparison," in *Proc. Ionospheric Radio Systems and Techniques, 12th IET Conf.*, York, 2012, pp. 1-5.

[9] G. Hand, "VOACAP', ICEPAC and REC-533 Propagation Prediction Programs for Windows," *NTI / ITS.*

Duncan M. G. Tait graduated from University of Reading with a Physics BSc and a year's placement working at MBDA, and went straight on to begin an EngD (completion Dec. 2013), at University of Bristol Systems Centre, working with Thales UK on this project: Improving business decisions through modelling and simulation of communication network systems. This is on-going and based at Thales UK, Crawley, RH10 9HA.