Tool-Based Performance Evaluation of the BlackBoard Communication System

Volker Remuss
Technische Universität Berlin
Real-Time System and Robotics
Einsteinufer 17, 10587 Berlin
++49 (30) 314-79336
remuss@cs.tu-berlin.de

Armin Zimmermann
Technische Universität Berlin
Real-Time System and Robotics
Einsteinufer 17, 10587 Berlin
++49 (30) 314-73110
zimmermann@cs.tu-berlin.de

ABSTRACT
This paper presents the integration of an existing real-world, real-time, ad-hoc communication system code into a discrete event simulation (DES). This embedded simulation environment can be used to analyze the communication system's behavior in regard to throughput, delay, dynamic network reconfiguration and more. The system in question is the blackboard communication system (BBCS). It is a development of the group for real-time systems and robotics of the Technische Universität Berlin (TUB). It was used in the European Project COMETS which integrated several unmanned aerial vehicles (UAVs) in cooperative fleet and is still used in several mobile systems within European robotics groups. The DES framework of choice is OMNeT++. This paper shows how the integration of the existing BBCS code into OMNeT++ is done and how it can be used to validate the BBCS. The underlying simulation model and the specific integration solutions are pointed out. The paper presents furthermore a multi UAV scenario that was analyzed in regard to the delay distribution and reliability in a dynamic reconfigured wireless network.

Categories and Subject Descriptors
C.2.1 [**Network Architecture and Design**]: Distributed Networks, I.6.5 [**Model Development]**: Modeling methodologies.

General Terms
Algorithms, Performance, Design, Reliability, Experimentation.

Keywords
Communication system, Simulation testbed, ad-hoc networking, embedded system, real-time system

1. INTRODUCTION
The communication between mobile systems such as robots and their base stations, but also between different subsystems of a single machine is an important part. Often, it is not treated as such. Especially in academic environments it is not uncommon to use very basic communication means while starting with a new system and then cope with emerging difficulties and oddities while the system grows. To overcome this, the blackboard communication system (BBCS) was developed at the real-time systems and robotics group at the TUB.

Development started in 1999 during the project MARVIN (multi-purpose aerial robot vehicle with intelligent navigation) [8] as a specific solution to the demands of a system that has to communicate internally between embedded devices and externally with and between usual workstation. The development continued during the European project COMETS where the BBCS was used for a fleet of heterogeneous UAVs and a network of ground station computers [5].

The system incorporates a set of functionalities that suits especially robotic environments. It is designed as reliable and robust communication system typically used to transmit telemetry, sensor data, video streams and commands that are usually produced just-in-time.

The BBCS is designed as distributed system namely as multi-hop ad-hoc network with automatic routing [10]. It contains a real-time aware protocol and bandwidth management to distribute data within the network. It features flow control, out-of-order reception, lost package, and error detection. In regards to the OSI layers the BBCS implements 6 to 2 [12]. Virtually every byte transmitting link can be used as connection between two nodes.

Since long range wireless data links are slow, link aggregation can be used to combine several links transparently to a single logical connection with higher bandwidth. A link failure is detected either by transmission timeouts or by means of the underlying link if available. A connection only fails if all of its links have failed.

Due to its number of features the BBCS has also a number of parameters that can be used to optimize the system for different scenarios. For this purpose and to be able to validate the BBCS' real code the decision was taken to integrate the code into an artificial environment like a simulator.

2. The BBCS

In the following section the BBCS design will be presented at the level needed to understand the later simulation system.

2.1 Network Organization of the BBCS

As usual in a communication network there are *data sources* and *data sinks*. A *network node* is a data-accessing entity that produces data, consumes data or both. A *network* is built of a number of network nodes that are communicating via a number of data transmitting *connections*.

The name blackboard communication system stems from a *distributed shared memory* (DSM) approach (see Figure 1). Nodes are accessing this DSM called blackboard by using read and write operations on dedicated data-sets called *slots* or directly by mapping slots in their memory space. The communication is all about slots. They are the only addressable entities. Slots are addressed with a numeric identifier, have a fixed maximum size and are treated as atomic data by the BBCS. There are basically two types of slots. *Secure* slots are used to transmit series of data that is expected to be received completely. A typical example would be a progressive transfer of an image, which is sent while being produced. *Insecure* slots contain atomic data, where newer will always overwrite older data. A typical example would be a sensor measurement in a real-time system, where newer data should never be delayed in favor of unsent expired data.

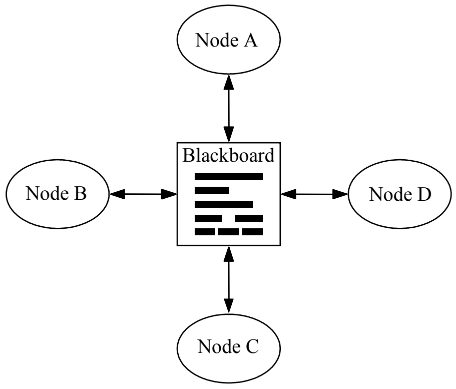

Figure 1: Blackboard as distributed shared memory

The BBCS takes care of the DSM's coherency throughout the entire (reachable) network, which is organized as a multi-hop ad-hoc network using virtually any kind of data link as connections between adjacent nodes. The system is fully distributed and free of any central entity. There is only one type of node.

A node is a process running on an operating system or as part of the single-task system in a small embedded system such as a microcontroller. It becomes a producer for a slot by writing to it and a consumer by reading it. The BBCS surveys in a distributed manner that there is only one producer to each slot. It is the idea to assign a slot to each specific purpose such as an individual sensor output. Therefore the producers are usually not changing during system operation. The design is somewhat similar to a fusion of the topic-based publish/subscriber-approach and shared spaces, but beside some other differences the BBCS has no central entity like a broker [4].

The network is organized as multi-hop ad-hoc network using distance vector routing [10][9]. Request for data as well as the data itself is sent from hop to hop. Each node can store and forward data. Every node is aware if there is a producer for a specific slot and if so, how the costs are to get it. The cost metric is usually given by the number of hops. If a source starts to produce data for a slot, it informs all its adjacent nodes that this specific slot is available at zero cost. These nodes then tell their adjacent nodes that the slot is available at one hop's cost and so on. Information about the availability will thus distribute through the network. Any data sink requests the data to be sent by its adjacent node with the lowest cost for this slot which then does the same. The path to the source is established with minimal overhead. Since there is no central entity, there must be a common knowledge about the identifiers and contents of slots in the network. It is important to note that the routing is done slot-wise and every slot has its own decentralized routing trees. The system generates one tree for every pair of source and sink for every slot.

The BBCS uses the message replication while routing (MRWR) approach for multicasting [2]. Even if there are several data sinks for a single data source in a network, data will only be transmitted once between hops as long as the routing trees are matching. Data multiplication for *n* data sinks will always occur in the node that is closest to the sink and present in each of the *n* routing trees. As a result, the same data is only transferred once on a single channel which is a necessity for the BBCS's bandwidth management approach.

The BBCS requires the assignment of a minimum bandwidth individual to each slot and a bandwidth to each data channel that is never exceeded and shall be guaranteed by the underlying hardware. Using these parameters as input, the system is able to guarantee a maximum delay for each slot and path.

The combination of user-mapped memory and bandwidth assignment is a key feature of the BBCS. It allows a user to setup a data transmission only once with an adequate bandwidth and the BBCS will automatically transmit the data with the maximum update rate that matches the bandwidth. This is a valuable feature for any real measurement like sensors data, trajectory telemetry or even series of pictures taken by a camera because usually the available bandwidth is the limiting factor.

2.2 Data Transmission

The connection between two adjacent nodes of the BBCS is organized hierarchically. The logical connection to another node is a bidirectional *port*. A port consists of at least one bidirectional channel. A *channel* consists of exactly one bidirectional or two unidirectional physical *links*. A physical link is some kind of byte-transmitting link. A port can contain virtually any number of channels to aggregate their bandwidth and become more robust. All links associated with a port have to connect the same node. Figure 2 depicts a pair of nodes connected by two channels with different bandwidths and a port with the combined bandwidth.

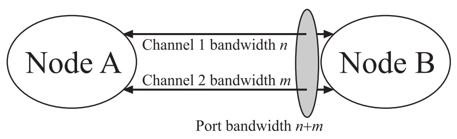

Figure 2: Channel aggregation

Data transmission over each port is done in stripes. Stripes are atomic data packages of a fixed maximum size. The maximum stripe size will typically range from 64 to 512 bytes and is a global parameter. Stripes are invisible to the user.

Error detection codes are used to assure the integrity of stripes. A sequence-numbering scheme is used to identify stripes and permit the reordering after out-of-order reception of stripes. Slot data is segmented into stripes which are multiplexed in byte stream(s) to be sent over a port's channel(s). The multiplexing uses a stripe-scheduling algorithm that takes into account the bandwidth fraction attached to every slot. The BBCS uses a very efficient binary-tree approach to calculate the schedule [7], which is beyond the scope of this document. The schedule only changes in case of a rerouting event and is then newly calculated just before it restarts.

In regards to the OSI layers [12], the BBCS implements 6 to 2. Nevertheless, during system design it became clear that it is a more practical approach in many cases to overlay the BBCS on a higher layer level to use existing infrastructure. The decision was taken to implement the BBCS also on top of existing layers such as the transportation layer.

2.3 The Platform Abstraction Layer

The adaptation between the BBCS and the underlying link uses a *platform abstraction layer* (PAL). There are PALs for different hard- and software architectures as well as link types. It is fully transparent for the user. A network node can have channels using different link types, thus allowing to bridge transparently between incompatible types of hardware.

Since the BBCS is running in user mode and is not integrated into operating systems, the communication core has to be executed periodically by the node's user code. This has to be done by calling a *synchronization operation* (sync-op). During this sync-op, the BBCS updates all received data in the node's DSM and sends the correct amount of new data according to the schedule and the assigned bandwidth using the PAL to access the underlying system. The platform has to have enough buffers in soft- or hardware to take in all data that is sent during the sync-op and received between two sync-ops. Therefore, the maximum feasible time between to sync-ops depends on these buffers' sizes and the assigned bandwidth.

In real systems, the sync-op is usually called with a fixed period of several milliseconds. The node's user code can randomly access slots at any time between two sync-ops, since the data is only changed during a synchronization. The synchronization operation and therefore the PAL have to be non-blocking.

The PAL implements functions for memory allocation, global time and read/write access.

2.4 BBCS in Current Systems

In current systems like the MARVIN UAV, the BBCS is used on bare serial wired and wireless RS232 links where it incorporates OSI layer 6 to 2 as well as over wired and wireless UDP and TCP connections where it resembles OSI layers 6 to 4. The benefit of using existing layers 3 to 1 is the opportunity to use all standard hardware and the possibility to test the communication even over long-distance internet links.

The BBCS has already been used in larger systems with multiple UAVs. A typical setup is shown in Figure 3. The BBCS would also be beneficial in the area of small sensor networks and smart dust.

2.5 BBCS in a Simulated Environment

During the use of the BBCS, the idea of having a simulation environment to test and validate rerouting capabilities, bandwidth observance and delays got tempting. These tests could be easier and with more accuracy executed in a *simulation environment* (SE). Since it was desired to test for real systems and to benefit directly from the results, the decision was taken to integrate the existing code into the SE instead of modeling on a more abstract layer. This SE needs to simulate at the underlying physical layer. The minimal simulation environment would provide links of variable attributes, a simulated time and means to establish multiple nodes and networks. Primarily, it had to be able to exchange real data. Therefore, Petri-net based TUB's TimeNet tool [4] or queuing network tools could not be used.

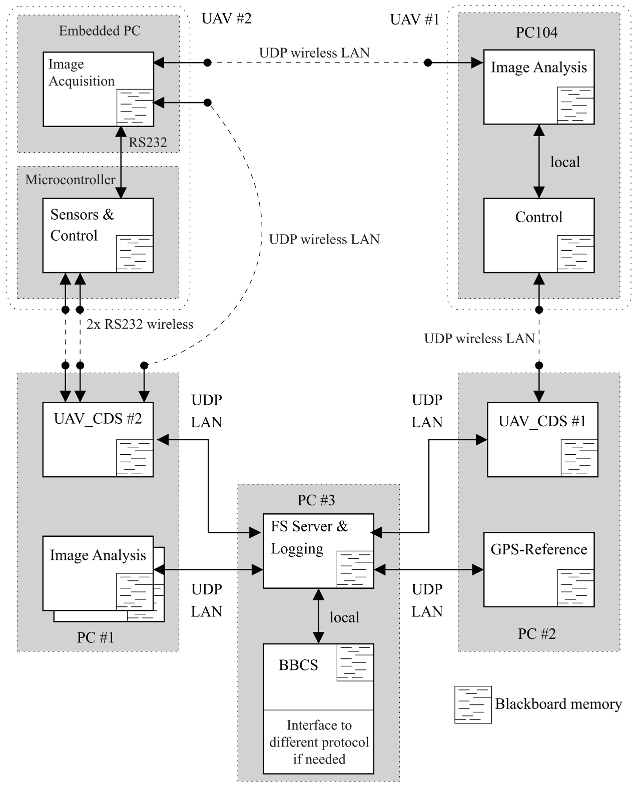

Figure 3: Typical network setup in an aerial robotic scenario

Since the BBCS at the lowest level transmits stripes and uses the store-and-forward principle, sending and reception of a stripe or several stripes can be modeled as an event without any restriction to the simulation possibilities. Therefore a discrete event network simulation is a matching simulator type as long as a simulated message can carry any real data as payload. OMNeT++ fulfils the specification. Moreover, integration of real code had been done for other applications before [3].

3. INTEGRATION OF BBCS IN OMNeT++

In the following, a brief overview of the relevant features of OMNeT++ is given and explained how the integration of the BBCS into the OMNeT++ simulation has been conducted.

3.1 The OMNeT++ DES

OMNeT++ is the acronym of object-oriented modular discrete event network simulator [11]. OMNeT++ is a DES especially designed for simulation of networks. It is written in C++, available for Unix and Windows and it comes with a runtime graphical user interface if wished. It is free for academic and non-profit use and an open-source project.

Its basic feature is the setup of hierarchical *modules* connected by unidirectional *gates*. *Messages* are sent from gates to gates. Modules can implement any kind of behavior that consumes and produces messages. Modules are implemented as classes. Hence they can be instantiated to create networks. Links between gates can have an error rate, data rate and fixed delay. Messages are sent and received between modules via gates. All message transfers are administered in a global event list and have an arrival time on which the receiving module is invoked to handle the message. Any module-internal timers have also to be handled by messages. Messages can be freely extended to carry any additional user data by means of a configuration file.

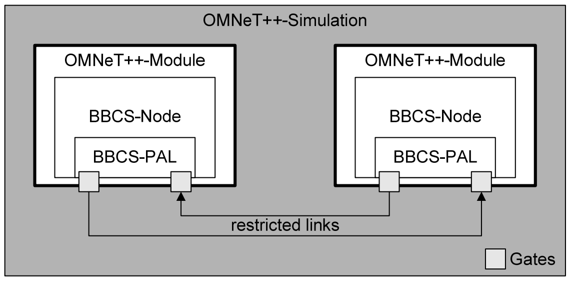

Figure 4: OMNeT++ Simulation Environment

Since OMNeT++ is using the features of C++ to generate simulation entities, the behavior of the modeled system has to be programmed in C++ as well and needs to be linked with the simulation kernel. The simulation is running as a single process in a single context. OMNeT++ adds flexibility to programming modules by having a network description language (NED) that is interpreted during the start of the simulation. NED can be used to setup a network for testing. It allows to instantiate modules and connects gates with specified attributes without the need to recompile.

Beside this, OMNeT++ offers random number generators and tools for data logging and statistical analysis. For severe statistical evaluation and parallel simulation execution it can be combined with Akaroa [6].

3.2 Real BBCS Code Integration

The existing BBCS code is written in plain C and, as stated before, uses a distributed shared memory approach. In a real network, every node of the BBCS network has a private copy of the DSM to work on. Since in OMNeT++ the simulation is running in a single process, the BBCS could not be integrated as plain C code, since then only one DSM area would exist in the simulation code. This could only be solved cleanly by converting the complete BBCS into a C++ class that can safely be instantiated by every network node module in the simulation. The DSM, all global variables and BBCS functions had to be part of that class.

Since the simulation was designed to test the real-world code in the first place it was also necessary to find a solution that could be integrated in the existing source tree without loosing compatibility with plain C. This goal was achieved by using some encapsulation code and preprocessor macros.

On the other side, the BBCS needed to interface OMNeT++ functionality. For this purpose the platform abstraction layer is used.

In Figure 4 a simple example for a simulation with two single port, single channel nodes is shown. The BBCS code is encapsulated in an OMNeT++ module and a special version of the PAL is used to send and receive messages via gates.

3.3 Adaptation to OMNeT++

A dedicated PAL was developed to interface the OMNeT++ API. To let the BBCS work, real data transfer is needed. Since the default OMNeT++ message cannot carry payload, a message extension had to be defined. Instead of encapsulating exactly one stripe in each message, the decision was taken to encapsulate the byte stream in messages of variable but restricted length. Larger messages led to fewer events at the same throughput which results in a simulation speed-up. Additionally, it can be used as simple link model. A message size of 1 byte would resemble a basic serial link, while using the upper bound of 1500 bytes would be similar to the usual maximum transmission unit (MTU) in Ethernet systems [1]. By using the link error rate model of OMNeT++, which marks messages as being broken only or by using time-outs, messages can be corrupted or dropped in the PAL to simulated erroneous connections. A BBCS channel has to be bidirectional and is set-up by using a pair of OMNeT++'s unidirectional gates.

OMNeT++ has two mutually exclusive ways to implement user code reactions to simulated events: Either by calling a handle method for every message received or by having the simulation code poll a read method. In the latter case, queuing of incoming events is supported. The design of the BBCS with the periodically called sync-op would match the second variant. However, since the first approach is superior in speed, the handle-implementation was chosen. Because BBCS' sync-op is designed to be called regularly and to pull all data from the link received since the last call, it was necessary to implement a reception queue in the simulation code. The BBCS is accessing the queue through the PAL. A benefit of this design is the possibility to instrument the queue or restrict its size.

Sending of messages is done directly to the OMNeT++ simulation kernel where all simulation events are queued.

To test the routing functionality it is necessary to simulate breaking up of links between nodes. Since such dynamic network capabilities of OMNeT++ are very limited, this has also been integrated in the PAL.

A problem when simulating a distributed network in a single simulation environment is the globally synchronized time. A fully synchronized net is too deterministic and covers only a subset of all possible states. In the BBCS simulation this is solved by invoking the sync-op operation with a random jitter or by using individual periods for each node.

3.4 Measurement and Evaluation Possibilities

In the simulation environment as described, two different kinds of observations are currently measured.

Transmission delays between pairs of nodes can be exactly measured since unlike in real a network, a global reference time is available. Since the BBCS itself is in the focus of attention, the delay times are preferably measured at the user side. For a realistic examination the BBCS itself can be used to transmit time stamp data. Measurements of the underlying OMNeT++ message transfer times are of no significance here.

Data throughput is interesting at two different levels. Firstly, the throughput for user data can be measured at the user side of the BBCS for real data transmissions. Secondly, the throughput that the BBCS generates and sends through the underlying link may be of interest. By doing so it is possible to inspect if the BBCS is acting as designed and fulfills the user's bandwidth request without exceeding the specified bandwidth capability of the underlying link.

These measurements can be taken on every network node and are sufficient to analyze the BBCS' performance in regard to routing capabilities, load scenarios and overall data transmission performance.

4. SIMULATION

In the following, simulation results for a dynamic network setup will be presented. The scenario was chosen because the authors have experience with real-life setups of this kind and the goal was to verify the correct function of the simulation and test BBCS' dynamic routing capabilities.

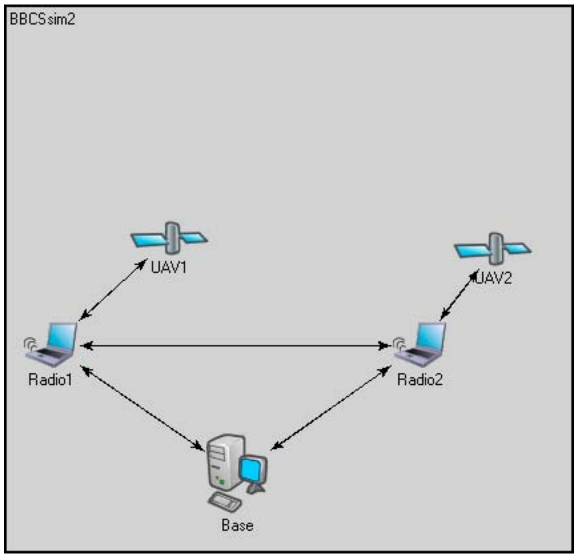

Figure 5: Initial UAV Scenario

4.1 Simulation Scenario

The simulated network as shown in Figure 5 resembles a typical UAV setup and is a slightly reduced version of the system shown in Figure 3. A ground system controls two UAVs using three computers. There is the base station *Base* for mission control and the laptops *Radio1* and *Radio2* that can be used for additional mission-specific tasks. They are also equipped with some radio links to the UAVs. The ground segment is fully connected using some wired network.

Each UAV can communicate with its ground station and with the other UAV, whichever is in reach.

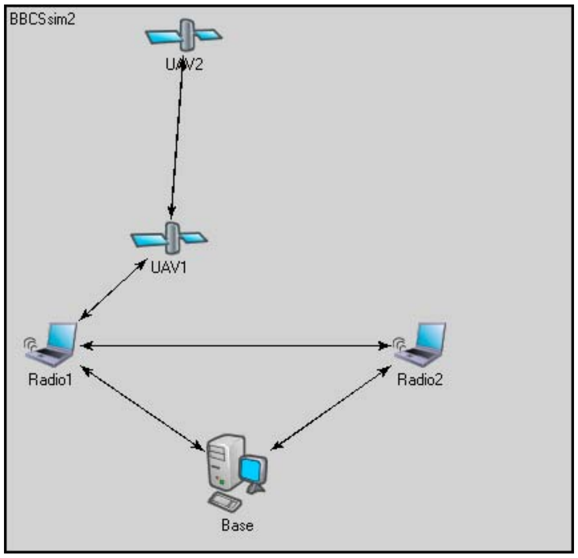

Figure 6: Scenario after 180 s of flight

The scenario simulated starts as shown in Figure 5 that displays the network as it is presented in the OMNeT++ simulator GUI. *UAV2* has a connection to Radio2 but not to *UAV1*. UAV2 then recedes from Radio1 and approaches UAV2. At 90 seconds simulated time, the connection to UAV2 is established and at 180 seconds the connection to Radio2 is lost as shown in Figure 6. The radio link will give no feedback about the connection situation. All links have the same cost and the same bandwidth of 1 MByte/s.

The data flow between UAV2 and Base is analyzed.

4.2 BBCS Behavior

The BBCS is designed to use the least expensive path between two nodes, which in the given scenario is the shortest path. The BBCS is also expected to reconfigure the route according to changes in the network structure. The shortest path between UAV2 and Base at the beginning of the scenario is two hops long. Data travels from UAV2 to Base using Radio2 as intermediate hop. At 90 seconds a new connection is established, UAV2 and UAV1 are starting to communicate. Since the newly available path for the data flow of interest is longer than the one in use, the observed transmission is not influenced. At 180 seconds the initial path breaks down and the BBCS has to reconfigure. A timeout feature is used to detect breakdown of links.

4.3 Simulation Setup and Measurements

During the above scenario, the synchronization interval of the BBCS was set to 10 ms plus a uniformly distributed jitter of +/-2 ms. The simulation environment was used to simulate establishment and destruction of communication lines. The BBCS has to sense the lost connectivity by detecting line inactivity using a timeout of 200 ms.

User data transferred from every node to every node were timestamps using an insecure slot (see section 2.1). This allows measurement of user data delay times and update rates for every possible communication path in the graph.

In Figure 7, a histogram of delay times measured in milliseconds is depicted. It presents the times for the path from UAV2 to Base during the first part of the experiment with the link between UAV2 and Radio2 still being intact.

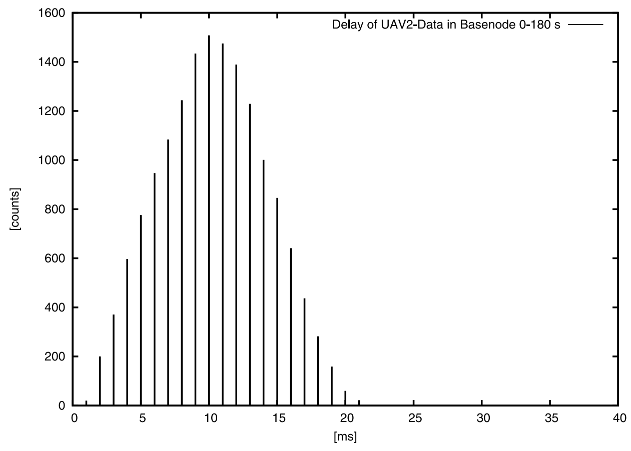

Figure 7: Delay of user data from UAV2 to Base 0-180 s

The path has a length of 2 hops. Since the data (timestamp) is small enough to be sent in a single stripe and the chosen bandwidth permits to send at least one timestamp per sync-op it is clear that the timestamp will be received with a maximum delay of 2 maximum sync-op cycle times once sent. This equals 24 ms plus some time for data transmission. The minimum delay measured is very close to 0 ms. This happens when all participating nodes are called in the optimal order and with low delay. All measured data lies in this boundary. The distribution is the result of the sync-op jitter and therefore the randomized order of invocations of the BBCS code in the three hops. The average delay in this particular experiment was 9.8 ms.

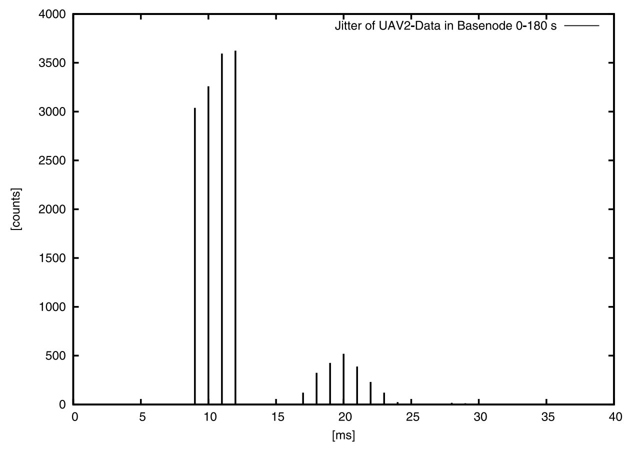

Figure 8: Time between updates of user data 0-180 s

Another interesting timing is the update rate of user data. This is measured as the time difference between two received time stamps. The histogram in Figure 8 shows the measurement at node Base. In most cases the update rate is similar to the sync-op period, meaning that in every cycle an update is received. In some cases, the update interval is about twice the sync-op period. In these cases there was no update available during a sync-op period and during the next period a newer time stamp was already available. This can happen because of the desynchronized operation of the individual nodes. In some cases the timing is such that a newer time stamp can catch up with an older time stamp. Newer time stamps will replace older ones because this scenario uses insecure slots which allow exactly this behavior to favor newer data. 16.3% of time stamps have been updated before their reception.

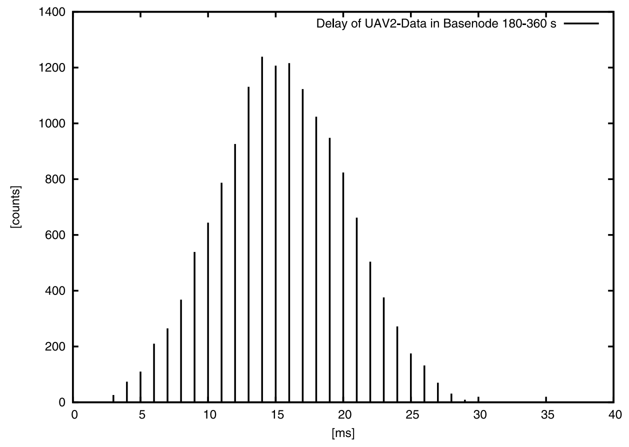

Figure 9: Delay of user data from UAV2 to Base 180-360 s

In the second part of the experiment the link between Radio2 and UAV2 breaks down after the first 180 seconds and a rerouting occurs. The new path has a length of three hops using UAV1 as relay station to UAV2. In Figure 9 again the delay time and Figure 10 the time between updates is depicted.

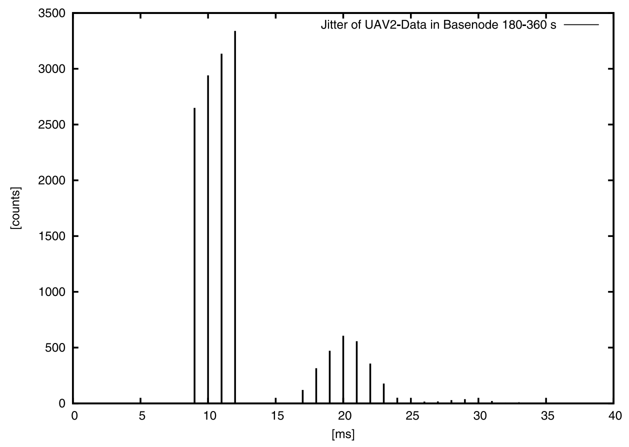

Figure 10: Time between updates of user data 180-360 s

It can bee seen that the delay time is longer due to the longer path and is still bound by the theoretical maximum of about 36 ms. The average delay for this part of the experiment is 15 ms. Figure 10 shows that the amount of messages that are being updated before reception at the base station has increased slightly to 23.5%.

Another interesting observation is the time needed by the BBCS to reconfigure the network. Figure 11 plots Base's time stamps as they are received in UAV2. At 180 second, that is when the existing connection between UAV2 and Radio2 vanishes, the update of Base's time stamps stops. It takes the system 200 ms to detect the link failure, because this is the chosen timeout parameter. 29 ms later the system has reconfigured and time stamp updates are resumed. Hence, the reconfiguration took only the time of 2 sync-ops, which is a good result.

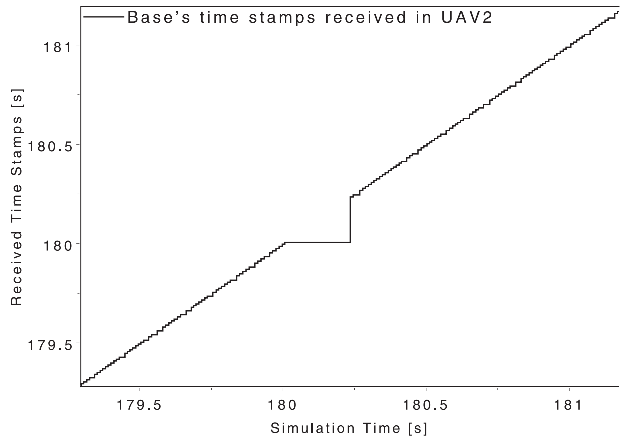

Figure 11: Rerouting Time for path UAV2 and Base

The fast reconfiguration is due to the BBCS' design. UAV2 already consumed the data from Base when it got UAV2's request for forwarding it. Therefore, the data could be made available during a single sync-op. The data origin Base is not involved in this reconfiguration process.

4.4 Simulation Conclusion

These first test results are quite promising. The original code has been integrated successfully and the chosen simulation model has already turned out to be a useful tool for the analysis of the BBCS. It is possible to analyze network scenarios in dynamic topologies. Measurement of reconfiguration and delay times is a profitable result.

The simulation in the current version can already be used to test scenarios without the need to set up a real network. It helps to make decision about useful real setups and their parameters. The fully implemented BBCS could also be used to test application code on a simulated network.

During the described simulation experiment, 397.000 OMNeT++ messages have produced and consumed. On a basic workstation (AMD Athlon64 x2 2.2 GHz, 2 GB RAM) the presented scenario was executed at about 27 times the original speed including intensive logging. The current simulation has no multi-processor support.

5. SUMMARY

In this paper, the integration of an existing communication system into a discrete-event simulator was presented. After a short introduction of the blackboard communication system and the simulator OMNeT++ it was explained what was needed to adapt the code to be able to simulate multi-node and -hop networks and how the underlying network and its dynamic reconfiguration was modeled.

The integrated simulation tool was then used to evaluate the behavior of the communication system. A real-life simulation scenario was chosen that resembles an existing aerial robotic system well known to the authors. The scenario includes a dynamic change in the network topology and was analyzed in regard to its timing performance.

The results show that the simulation is working properly and is already a useful tool to evaluate the behavior of the communication system. It is planned to be used to test future extensions of BBCS and application scenarios.

6. OUTLOOK

From the engineering point of view, the BBCS simulation will be used to design and analyze network setups for upcoming robotic systems. The simulation will also be used to analyze – and if necessary – to improve the behavior of the BBCS during different load conditions, larger networks, isochronous transfers and multimedia traffic. It is also possible to test the performance on all underlying layers that are available for OMNeT++.

The main point of interest is the robustness of the system.

7. REFERENCES

[1] *The Ethernet: a local area network: data link layer and physical layer specifications*. SIGCOMM Computer Communication Review 11, USA, 1981, 20-66

[2] Bhattacharya, S., Elsesser, G., Tsai, W., and Du, D. 1994. Multicasting in generalized multistage interconnection networks. *J. Parallel Distrib. Comput.* 22, 1 (Jul. 1994), P. 80-95

[3] Bless R., Doll M., *Integration of the freebsd TCP/IP-stack into the discrete event simulator OMNET++* Simulation Conference, 2004. Proceedings of the 2004 Winter Volume 2, 2004, P. 1556 - 1561

[4] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A., *The many faces of publish/subscribe*. ACM Computing Surveys 35, 2 (Jun. 2003), P. 114-131

[5] Ollero, A. et al. *Multiple eyes in the skies: architecture and perception issues in the COMETS unmanned air vehicles project,* Robotics & Automation Magazine, IEEE Volume 12, Issue 2, June 2005, P 46 – 57

[6] Pawlikowski, K., Yau, V. W. C., and McNickle, D., *Distributed stochastic discrete-event simulation in parallel time streams*, Proceedings of the 1994 Winter Simulation Conference; 1994, P. 723-730

[7] Remuss, V., Musial, M., *Communication System for Cooperative Mobile Robots using Ad-hoc Networks,* IFAC Symposium on Intelligent Autonomous Vehicles, Portugal, 2004

[8] Remuss, V., *MARVIN mark II*, pdv.cs.tu-berlin.de/MARVIN/index.html, Germany, 2005

[9] Tanenbaum, A. S., *Computer Networks*, 3rd ed., Prentice Hall, Upper Saddle River, New Jersey, 1996, P. 355-358,

[10] Tutsch, D., Performance Analysis of Network Architectures, Springer 2006, Germany, P. 55

[11] Varga, A., *OMNeT++ Community Site*, www.omnetpp.org, 2007

[12] Zimmermann, A., Knoke, M., Huck, A., Hommel, G., *Towards Version 4.0 of TimeNET*. Proc.13th Conf. on Measurement, Modeling, and Evaluation of Computer and Communication Systems, 2006, P. 477 – 480.

[13] Zimmermann, H., *OSI Reference Model — The ISO Model of Architecture for Open Systems*, IEEE Transactions on Communications, vol. 28, no. 4, April 1980, P. 425 – 434