Simulation of Networked Control Systems Using TrueTime

Cervin, Anton; Ohlin, Martin; Henriksson, Dan

2007

SIMULATION OF NETWORKED CONTROL SYSTEMS USING TRUETIME

Anton Cervin, Martin Ohlin, Dan Henriksson

Department of Automatic Control LTH
Lund University, Sweden
anton@control.lth.se

Abstract: This paper gives a brief introduction to the TrueTime simulator and then gives several examples on how TrueTime can be used to simulate networked control systems. Among the examples are time-triggered and event-based networked control and AODV routing in wireless ad-hoc networks.

1. INTRODUCTION

TrueTime (Cervin *et al.*, 2003; Andersson *et al.*, 2005) is a Matlab/Simulink-based simulator for networked and embedded control systems that has been developed at Lund University since 1999. The simulator software consists of a Simulink block library (see Fig. 1) and a collection of MEX files. The kernel block simulates a real-time kernel executing user-defined tasks and interrupt handlers. The various network blocks allow nodes (kernel blocks) to communicate over simulated wired or wireless networks. The latest release, TrueTime 1.5, also features a couple of stand-alone network interface blocks that makes it simpler to develop networked control simulations.

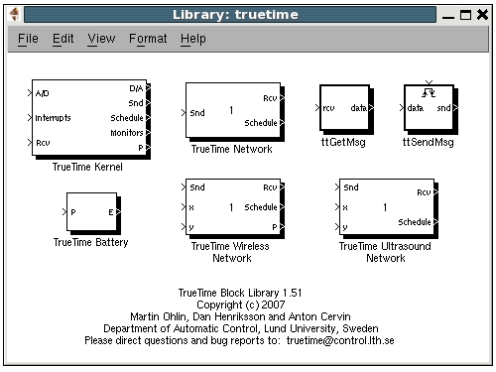

Fig. 1. The TrueTime 1.5 block library. TrueTime is freeware and can be downloaded from `www.control.lth.se/truetime`.

In contrast to other co-simulation tools such as Stateflow/Simulink or Ptolemy II (Baldwin *et al.*, 2004), TrueTime is not based on a mathematical modeling formalism. Rather, a TrueTime simulation is programmed in much the same way as a real embedded system. The application is written in Matlab code or in C++. The main difference from real programming is that the execution/transmission times must be specified by the developer. This approach makes TrueTime a very flexible co-simulation tool. Also, the step from simulation code to production code is not that large. The main drawback is that the simulation models are not amenable to analysis.

In this paper, we will focus on how networked control applications can be simulated using TrueTime. Following a brief overview of TrueTime in Section 2, a number of networked control examples are given in Section 3. An overview of related simulation tools are given in Section 4, and the conclusions are given in Section 5.

2. A BRIEF OVERVIEW OF TRUETIME

2.1 *The Kernel Block*

The TrueTime Kernel block simulates a computer node with a generic real-time kernel, A/D and D/A converters, and network interfaces. The block is configured via an initialization script. The script may be

parametrized, enabling the same script to be used for several nodes.

In the initialization script, the programmer may create objects such as tasks, timers, interrupt handlers, semaphores, etc., representing the software executing in the computer node. During a simulation, the kernel repeatedly calls the code functions of the tasks and interrupt handlers. The code functions may in turn contain arbitrary function calls, but they must be written in a special format that specifies the execution time of each code segment. Another restriction is that local variables do not retain their values between segments.

The initialization script and the code functions may be written in either Matlab code or in C++. In the C++ case, the initialization script and the code functions are compiled using Matlab's MEX facility, rendering a much faster simulation.

The TrueTime Kernel block supports various preemptive scheduling algorithms such as fixed-priority scheduling and earliest-deadline-first scheduling. It is also possible to specify a custom scheduling policy.

2.2 *The Network Blocks*

The TrueTime Network block and the TrueTime Wireless Network block simulate the physical layer and the medium-access layer of various local-area networks. The types of networks supported are CSMA/CD (Ethernet), CSMA/AMP (CAN), Round Robin (Token Bus), FDMA, TDMA (TTP), Switched Ethernet, WLAN (802.11b), and ZigBee (802.15.4). The blocks only simulate the medium access (the scheduling), possible collisions or interference, and the point-to-point/broadcast transmissions. Higher-layer protocols such as TCP/IP are not simulated (but may be implemented as applications in the nodes).

The network blocks are mainly configured via their block dialogues. Common parameters to all types of networks are the bit rate, the minimum frame size, and the network interface delay. For each type of network there are a number of further parameters that can be specified. For instance, for the wireless networks it is possible to specify the transmit power, the receiver signal threshold, the pathloss exponent (or a special pathloss function), the ACK timeout, the retry limit, and the error coding threshold.

A TrueTime model may contain several network blocks, and each kernel block may be connected to more than one network. Each network is identified by a number, and each node connected to a network is addressed by a number that is unique to that network.

The network blocks may be used in two different ways. The first way is to have one kernel block for each node in the network. The tasks inside the kernels

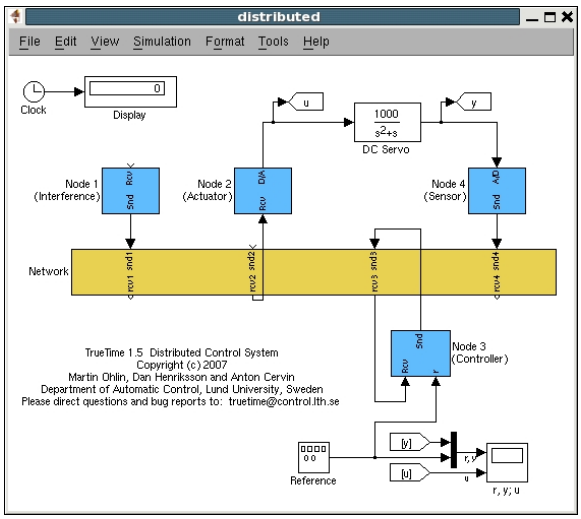

Fig. 2. Networked control system with a plant, four computer nodes, and a network.

can then send and receive arbitrary Matlab structure arrays over the network using certain kernel primitives. This approach is very flexible but requires some amount of programming to configure the system. The second way is to use the stand-alone network interface blocks. These blocks eliminate the need of kernel blocks, but they restrict the network packets to contain scalar or vector signal values. Finally, it is possible to mix kernel blocks and network interface blocks in the same network.

3. EXAMPLES

3.1 *Configuring a Networked Control Application*

The first example illustrates the steps needed to configure a networked control application in TrueTime. We will consider the model in Fig. 2 (example `distributed.mdl` in the TrueTime distribution), in which four nodes are connected to a network. The Sensor node samples the process output periodically and sends the measurement values to the Controller node over the network. The arrival of a sensor packet to the Controller node triggers a task that computes a new control signal. The control signal is then sent to the Actuator node, where it is applied to the process.

In this example, each node contains a TrueTime Kernel block. Focusing on the Controller node, the initialization of the kernel involves specifying the number of analog inputs and outputs, selecting the scheduling policy, creating the control task, and configuring the network interface. The complete Matlab code for this is given below:

```
function controller_init

% Initialize the kernel
```

```
nbrInputs = 1;
nbrOutputs = 0;
sched = 'prioFP';  % Fixed-priority sched.
ttInitKernel(nbrInputs, nbrOutputs, sched);

% Controller parameters
h = 0.010;  % Sampling period
N = 100;
Td = 0.035;
K = 1.5;

% Create task data (local memory)
data.u = 0.0;
data.K = K;
data.ad = Td/(N*h+Td);
data.bd = N*K*Td/(N*h+Td);
data.Dold = 0.0;
data.yold = 0.0;

% Create control task
deadline = h;
prio = 2;
ttCreateTask('ctrltask', deadline, prio, ...
             'ctrlcode', data);

% Initialize network interface
ttCreateInterruptHandler('nw_handler', ...
                         prio, 'msgRcv');
ttInitNetwork(3, 'nw_handler'); % node #3
```

The local memory of the task (in this case controller parameters and states) is represented by a Matlab structure array (`data`). Each time the task is invoked, the code function `ctrlcode.m` should be executed. This function is given below:

```
function [exectime,data] = ctrlcode(seg,data)
switch seg,
 case 1,
  y = ttGetMsg;        % Get sensor message
  r = ttAnalogIn(1);   % Read reference value
  P = data.K*(r-y);
  D = data.ad*data.Dold + ...
      data.bd*(data.yold-y);
  data.u = P + D;
  data.Dold = D;
  data.yold = y;
  exectime = 0.0005;
 case 2,
  ttSendMsg(2,data.u,80);  % Send to actuator
  exectime = -1; % finished
end
```

The code is this case consists of two segments, corresponding to the two cases in the `switch` statement. In the first segment, the sensor value is read from the network interface and the reference is read from an analog input. The control signal in then computed. This takes 0.5 ms simulated time, as specified by the `exectime` return value. In the second segment, the control signal is sent to the actuator.

In the last line of the initialization script, an interrupt handler is connected to the network interface. Each time a packet arrives, the function `msgRcv` should be executed. This minimal code function simply creates an instance (a job) of the controller task:

```
function [exectime,data] = msgRcv(seg, data)
ttCreateJob('ctrltask')
exectime = -1; % finished
```

To complete the configuration, similar functions (an initialization function, a task code function, and an interrupt handler code function) must be written for each of the nodes. Finally, the network must be configured (via its block dialogue) to have 4 nodes, and the other relevant network parameters must be chosen.

3.2 *Using the Stand-Alone Network Interface Blocks*

The second example illustrates how the new, stand-alone network interface blocks can be used to simulate time-triggered or event-triggered networked control loops. In this case, because there are no kernel blocks, no initialization scripts or code functions must be written.

The networked control system in this example consists a plant (an integrator), a network, and two nodes: an IO device (handling AD and DA conversion) and a controller node. At the IO node, the process is sampled by a ttSendMsg network interface block, which transmits the value to the controller node. There, the packet is received by a ttGetMsg network interface block. The control signal is computed and the control is transmitted back to the IO node by another ttSendMsg block. Finally, the signal is received by a ttGetMsg block at the IO and actuated to the process.

Two versions of the control loop will be studied. In Fig. 3, both ttSendMsg blocks are time-triggered. The process output is sampled every 0.1 s, and a new control signal is computed with the same interval but with a phase shift of 0.05 s. The resulting control performance and network schedule is shown in Fig. 4. The process output is kept close to zero despite the process noise. The schedule shows that the network load is quite high.

In the second version of the control loop, the ttSendMsg blocks are event-triggered instead, see Fig. 5. A sample is generated whenever the magnitude of the process output passes 0.25. The arrival of a measurement sample at the controller node triggers—after a delay—the computation and sending of the control signal back to the IO node. The resulting control performance and network schedule is shown in Fig. 6. It can be seen that the process is still stabilized, although much fewer network messages are sent.

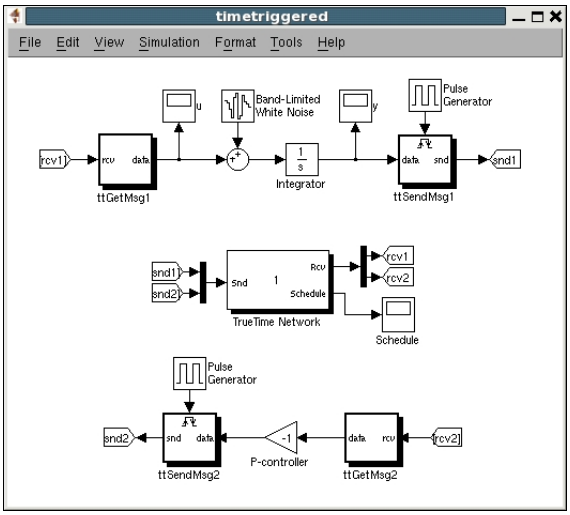

Fig. 3. Time-triggered networked control system using the stand-alone network interface blocks. The ttSendMsg blocks are driven by periodic pulse generators.

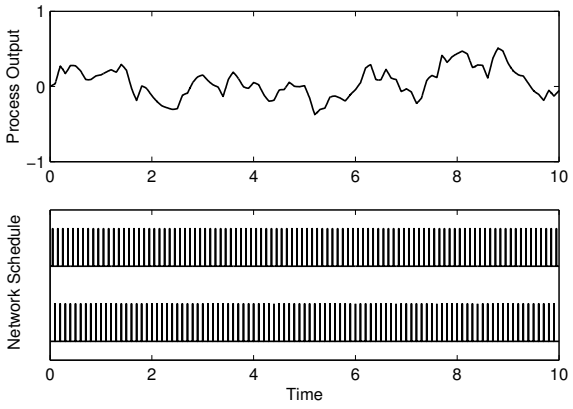

Fig. 4. Plant output and network schedule for the time-triggered control system.

3.3 *AODV Routing in Wireless Networks*

The TrueTime Wireless Network block simulates communication in an ad-hoc network, i.e., no centralized access point or infrastructure exists to coordinate the traffic across the network. In such networks it is necessary to implement decentralized functionality to be able to route the traffic over the network. This example describes a TrueTime implementation of one such ad-hoc wireless routing protocol.

AODV (Perkins and Royer, 1999) stands for Ad-hoc On-Demand Distance Vector routing, and contrary to most routing mechanisms, it does not rely on periodic transmission of routing messages between the nodes. Instead, routes are created on-demand, i.e., only when actually needed to send traffic between a source and a destination node. This leads to a substantial decrease in the amount of network bandwidth consumed to

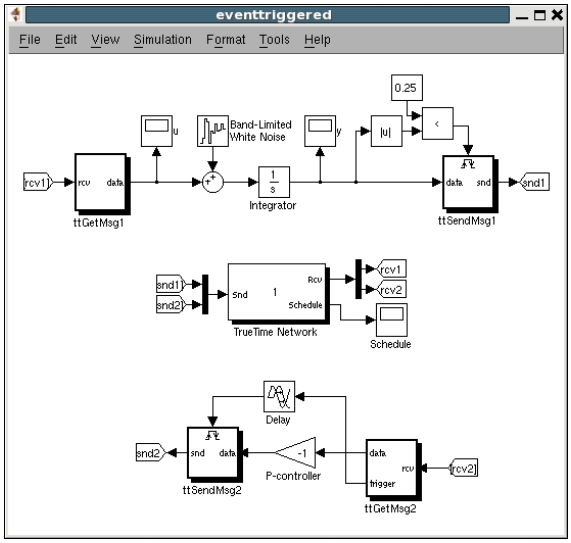

Fig. 5. Event-triggered networked control system using the stand-alone network interface blocks. The process output is sampled by the ttSendMsg block when the magnitude exceeds a certain threshold.

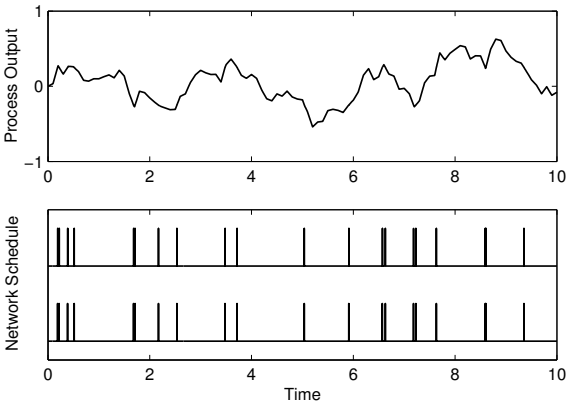

Fig. 6. Plant output and network schedule for the event-triggered control system.

establish routes. Below follows a brief description of the functionality of AODV. For a complete definition of the AODV protocol, see (Perkins and Royer, 2003).

AODV uses three basic types of control messages in order to build and invalidate routes: route request (RREQ), route reply (RREP), and route error (RERR) messages. These control messages contain source and destination sequence numbers, which are used to ensure fresh and loop-free routes.

A node that requires a route to a destination node initiates route discovery by broadcasting an RREQ message to its neighbors. A node receiving an RREQ starts by updating its routing information backwards towards the source. If the same RREQ has not been received before, the node then checks its routing table for a route to the destination. If a route exists with a

sequence number greater than or equal to that contained in the RREQ, an RREP message is sent back towards the source. Otherwise, the node rebroadcasts the RREQ. When an RREP has propagated back to the original source node, the established route may be used to send data. Periodic hello messages are used to maintain local connectivity information between neighboring nodes. A node that detects a link break will check its routing table to find all routes which use the broken link as the next hop. In order to propagate the information about the broken link, an RERR message is then sent to each node that constitute a previous hop on any of these routes.

To implement AODV in TrueTime, two dedicated tasks are created in each node to handle the AODV send and receive actions, respectively. The AODV Send task is activated from the application code as a data message should be sent to another node in the network. The AODV Receive task handles incoming AODV control messages and forwarding of data messages. Communication between the application layer and the AODV layer is handled using mailboxes.

The AODV Send task operates according to the following pseudo-code:

```
if data message received from application then
    check the routing table for a route to the destination;
    if a valid route exists then
        forward data message to next hop on route;
        update expiry time of route entry;
    else
        initiate route discovery by broadcasting RREQ message;
        buffer data message until route has been established;
    end if
else if notified of established new route then
    send all buffered data messages to destination
end if
```

The AODV Receive task performs the following:

```
if receiving data message then
    update expiry timer for reverse route entry to source;
    if this node is the destination then
        Pass data message to application;
    else
        forward data message to next hop on route;
        update expiry timer of route entry;
    end if
else
    if message_type == RREQ then
        if first time this RREQ is received then
            enter RREQ in cache;
            create or update route entry to source;
            check the routing table for a route to the destination;
            if a route exists then
                send RREP message back towards source;
            else
                update and rebroadcast the RREQ;
            end if
        end if
    else if message_type == RREP then
        check the routing table for a route to the destination;
        if no route exists then
            create route entry to destination;
        else if route entry exists but should be updated then
            update route entry to destination;
        end if
        if this node is the original source then
            notify the AODV send task about the new route;
        else if route to destination was created or updated then
            update reverse route entry towards source;
            propagate RREP to next hop towards source;
        end if
    else if message_type == RERR then
        find and invalidate all affected route entries;
        propagate the RERR to all previous hops on the routes;
    end if
end if
```

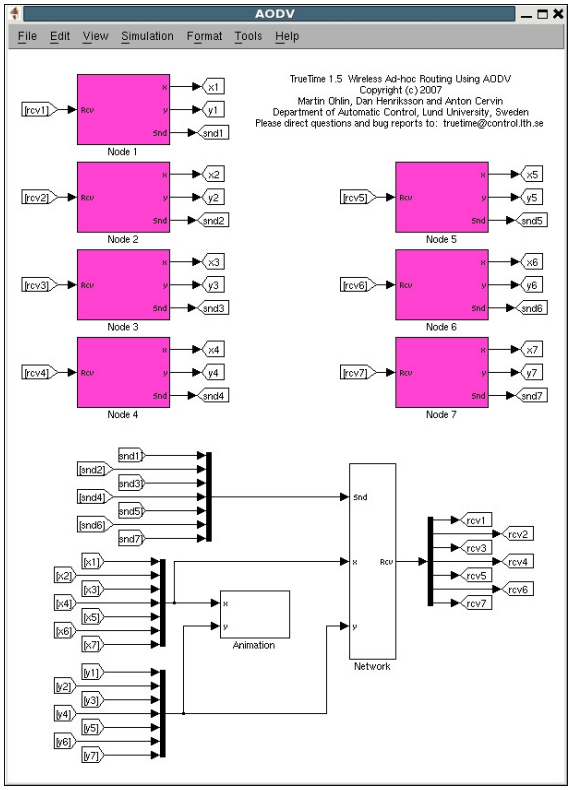

Fig. 7. Large AODV routing example involving seven mobile nodes and a wireless network.

Each node also contains a periodic task, responsible for broadcasting hello messages and determine local connectivity based on hello messages received from neighboring nodes. Finally, each node has a task to handle timer expiry of route entries.

A TrueTime model with seven nodes connected to a wireless network is shown in Fig. 7. Each node contains a kernel block that runs the AODV algorithm and integrators that represent the x and y positions of the node, see Fig. 8. The analog outputs of the kernel are used to move the node, and the analog inputs can be used to sense the current location. In the model, there is also a block that animates the position of the nodes as the simulation progresses, see Fig. 9. In the example, node 1 wants to periodically send data to node 7. The initial route that is established is $1 \rightarrow 3 \rightarrow 5 \rightarrow 7$. However, at one point in time,

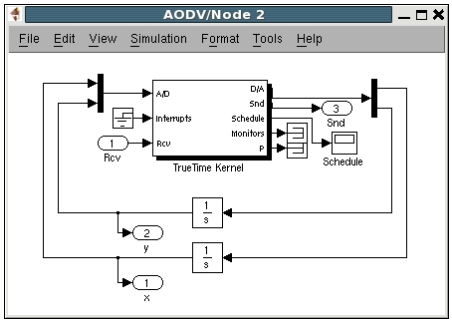

Fig. 8. Mobile node containing a TrueTime kernel and two integrators representing the x and y positions of the node.

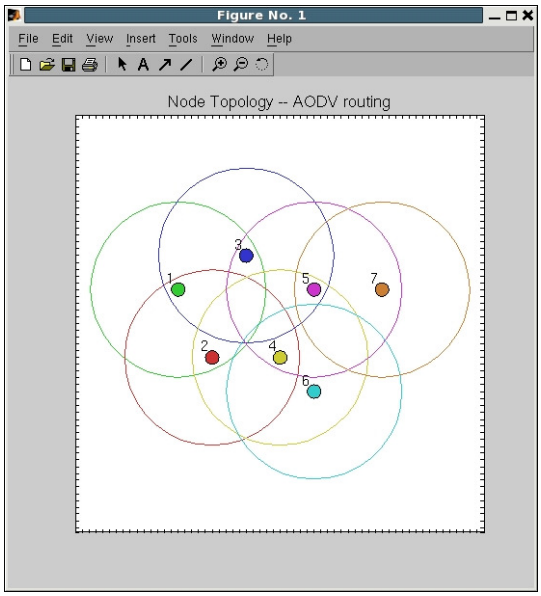

Fig. 9. Animation in the AODV example showing the positions and communication radius of the mobile nodes.

node 5 starts to move, which leads to the route being broken. Eventually, node 6 repairs the route by moving in between node 4 and 7.

4. RELATED WORK

Today there exists a number of general network simulators. One of the most well-known is ns-2 (The VINT Project, 2004), which is a discrete-event simulator for both wired and wireless networks with support for, e.g., TCP, UDP, routing, and multi-cast protocols. It also supports simple movement models for mobile applications. The channel model in ns-2 is quite simple (Dricot and Doncker, 2004). ns-2 makes the assumption that messages are received without errors if the power level is above a certain threshold. Packets with power levels below the same threshold are simply dropped. The packet with the largest power level is received if two transmissions occur at the same time, and the difference in power level between them is larger than 10 dB. Otherwise both packets are dropped. Three different path-loss models are available, two of them are deterministic and form ideal circles where the messages are received perfectly inside and dropped outside. The third model is called the shadowing model and adds some probabilistic changes to the path-loss by using a zero mean Gaussian variable.

Another discrete-event computer network simulator is OMNeT++ (OMNeT++ Community, 2004). It contains detailed IP, TCP, and FDDI protocol models and several other simulation models (file system simulator, Ethernet, framework for simulation of mobility, etc.). It uses the same path-loss function as the TrueTime wireless block, errors are however treated in a more detailed manner. It distinguishes between header and data part of packages and also between different modulation techniques. Compared to these simulators, the network simulation part in TrueTime is in some cases more simplistic. However, the strength of TrueTime is the co-simulation facilities that make it possible to simulate the latency-related aspects of the network communication in combination with the node computations and the dynamics of the physical environment. Rather than basing the co-simulation tool on a general network simulator and then try to extend this with additional co-simulation facilities, the approach has been to base the co-simulation tool on a powerful simulator for general dynamical systems, i.e., Simulink, and then add support for simulation of real-time kernels and the latency aspects of network communication to this. An additional advantage of this approach is the possibility to make use of the wide range of toolboxes that are available for MATLAB/Simulink. For example, support for virtual reality animation.

There are also some network simulators geared towards the sensor network domain. TOSSIM (Levis *et al.*, 2003) compiles directly from TinyOS code and scales very well. Its radio and interference model is however very simplistic, with either perfect transmissions or predefined error rates which can be changed at runtime. COOJA (Österlind, 2006) is similar to TOSSIM but simulates the Contiki OS instead. Network in a box (NAB) (NAB, 2004) is another simulator for large-scale sensor networks. Another example is J-Sim, a general compositional simulation environment that includes a generalized packet switched network model that may be used to simulate wireless LANs and sensor network (Tyan, 2002). Again, these types of simulators generally lack the possibility to simulate continuous-time dynamics and to simulate the inner workings of the nodes at the thread and interrupt handler level, features that have been present in TrueTime since the early versions.

A few other tools have been developed that support co-simulation of real-time computing systems and control systems. RTSIM (Palopoli *et al.*, 2000) has a module

that allows system dynamics to be simulated in parallel with scheduling algorithms. XILO (El-Khoury and Törngren, 2001) supports the simulation of system dynamics, CAN networks, and priority-preemptive scheduling. Ptolemy II is a general purpose multi-domain modeling and simulation environment that includes a continuous-time domain, and a simple RTOS domain. Recently it has been extended in the sensor network direction (Baldwin *et al.*, 2004). In (Branicky *et al.*, 2003) a co-simulation environment based on ns-2 is presented. The ns-2 simulator has been extended with an ODE-solver for dynamical simulations of the controller units and the environment. However, this tool lacks support for real-time kernel simulation.

5. CONCLUSION

TrueTime is a flexible tool for simulation of networked control systems. Using TrueTime, it is possible to do co-simulation of

- the computations inside computer nodes, including tasks and interrupt handlers,
- the scheduling algorithm in the nodes,
- the wired/wireless communication between nodes,
- the dynamics of the physical plant under control,
- the sensor and actuator dynamics,
- the dynamics of mobile robots/nodes,
- the dynamics of the environment, and
- the energy consumption in the nodes.

One general limitation of TrueTime is that it has been developed as a research tool rather than as a tool for system developers. The special code function format is another limitation, which makes it hard to directly simulate production code. Finally, the tool is based on commercial software. We are currently developing a version of TrueTime for Scilab/Scicos. We would also like to develop a multi-threaded version in which it would be simpler to directly simulate production code.

6. REFERENCES

Andersson, Martin, Dan Henriksson, Anton Cervin and Karl-Erik Årzén (2005). Simulation of wireless networked control systems. In: *Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference ECC 2005*. Seville, Spain.

Baldwin, Philip, Sanjeev Kohli, Edward A. Lee, Xiaojun Liu and Yang Zhao (2004). Modeling of Sensor Nets in Ptolemy II. In: *IPSN'04: Proceedings of the Third International Symposium on Information Processing in Sensor Setworks*. ACM Press. pp. 359–368.

Branicky, Michael, Vincenzo Liberatore and Stephen M. Phillips (2003). Networked Control Systems Co-Simulation for Co-Design. In: *Proceedings of the 2003 American Control Conference*. Vol. 4. Denver, USA. pp. 3341–3346.

Cervin, Anton, Dan Henriksson, Bo Lincoln, Johan Eker and Karl-Erik Årzén (2003). How does control timing affect performance?. *IEEE Control Systems Magazine* **23**(3), 16–30.

Dricot, J.-M. and Ph. De Doncker (2004). High-accuracy physical layer model for wireless network simulations in NS-2. In: *Proceedings of the International Workshop on Wireless Ad-hoc Networks, IWWAN'04*. Oulu, Finland.

El-Khoury, Jad and M. Törngren (2001). Towards a Toolset for Architectural Design of Distributed Real-Time Control Systems. In: *Proceedings of the 22nd IEEE Real-Time Systems Symposium*. London, England.

Levis, Philip, Nelson Lee, Matt Welsh and David Culler (2003). TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In: *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*. Los Angeles, CA, USA. pp. 126–137.

NAB (2004). NAB (Network in A Box). Home page: `http://nab.epfl.ch`.

OMNeT++ Community (2004). OMNeT++ Discrete Event Simulation System. Home page: `http://www.omnetpp.org`.

Österlind, Fredrik (2006). A Sensor Network Simulator for the Contiki OS. Technical Report T2006-05. Swedish Institute of Computer Science.

Palopoli, L., L. Abeni and G. Buttazzo (2000). Real-time control system analysis: An integrated approach. In: *Proceedings of the 21st IEEE Real-Time Systems Symposium*. Orlando, Florida.

Perkins, Charles E. and Elizabeth M. Royer (1999). Ad-hoc on-demand distance vector routing. In: *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*. New Orleans, LA. pp. 90–100.

Perkins, Charles E. and Elizabeth M. Royer (2003). Ad-hoc on-demand distance vector (AODV) routing. Request for Comments: http://www.ietf.org/rfc/rfc3561.txt.

The VINT Project (2004). The Network Simulator ns-2. Home page: `http://www.isi.edu/nsnam/ns/index.html`.

Tyan, H.-Y. (2002). Design, realization and evaluation of a component-based compositional software architecture for network simulation. PhD thesis. Ohio State University.