SIMULATION OF NETWORK USING TRUETIME TOOLBOX

T. Chvostek, A. Kratky**, M. Foltin****

[*]Institute of Control and Industrial Informatics, Faculty of Informatics and Information Technologies, Ilkovičova 3, 812 19 Bratislava, Slovak Republic
[**]Institute of Control and Industrial Informatics, Faculty of Electrical Engineering and Information Technology, Ilkovičova 3, 812 19 Bratislava, Slovak Republic
[***] SYPRIN s.r.o., Žehrianska 10, 851 07 Bratislava, Slovak Republic

Introduction

A typical control system contains an industrial controller, controlled process and some kind input/output channel, usually a communication network. In a standard way simulation scheme often does not contain any model of network, model is simplified and we rather assume that we have enough computing power and a fast communication network. This solution does not consider any network influence during control process which can lead to at least to an unpredictable behavior. When a network is used to exchange data in a hard real-time system, we need also to include a network model in a simulation scheme to get proper results. Used network type defines specific communication rules and leads to unique behavior of data flow which can dramatically change output results in process control. Delays, data drops or network load can degrade quality of transferred data in real-time control loop. This is the main reason why we have to include network model in simulation models of complex or real-time embedded systems (for. ex. with strict timing conditions) to improve controller stability or robustness.

Network behavior can be usually simulated with state flow models, Petri nets, or in a custom made schemes in a Matlab SimEvents toolbox. In this article we use TrueTime toolbox (for Matlab) as a simple and easy way how to realize several network types. We will demonstrate you how to setup simple TrueTime network control system with an explanation of its basic settings and parameters. In the last section we briefly compare simulation results of motor control system which uses different network types.

TrueTime toolbox

TrueTime is a Matlab/Simulink-based simulator for real-time control systems. This toolbox facilitates co-simulation of controller task execution in real-time kernels, network transmissions and continuous plant dynamics. It is written in C++ MEX language, uses event-based simulation and external interrupts.

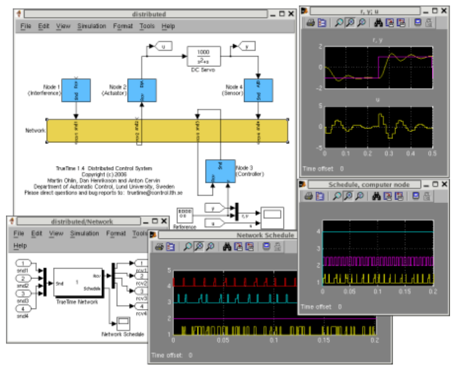


Fig. 1 Example of TrueTime toolbox simulation scheme

This toolbox provides possibility to write tasks as M-files, C++ functions or call Simulink block diagrams from within the code functions. TrueTime blocks include generally used networks as Ethernet, CAN, TDMA, FDMA, Round Robin or Switched Ethernet). It supports simulation of Wireless networks (802.11b/g WLAN and 802.15.4 ZigBee) and battery-powered devices. In a brief description we can say, that TrueTime is a small library of simulation blocks which extends usability of Matlab/Simulink to simulate discrete network process control.

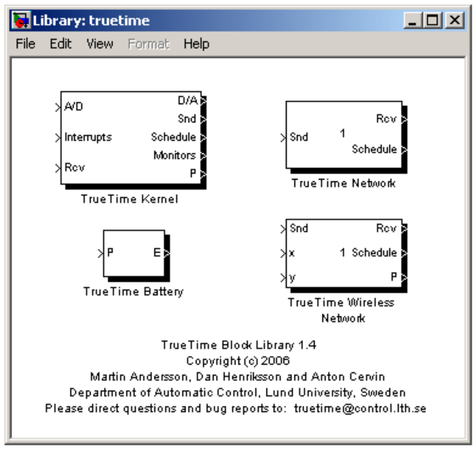
The basic principle of TrueTime simulation model

Every TrueTime toolbox simulation scheme should contain three crucial parts: TrueTime kernel (computer, I/O device or some embedded system), TrueTime network (network model) and a controlled process. There is also an optional part TrueTime battery (all blocks are shown on Fig. 2).

TrueTime kernel is responsible for I/O and network data acquisition or data processing and calculations. It can realizes a control algorithm/logic and it is the "brain" of every device. It uses several simple M-files (modified by us to satisfy our needs) which handle all mentioned operations. In the kernel can be executed several independent tasks (periodic, non-periodic) which can cooperate on the same goal.

TrueTime network or TrueTime Wireless network are blocks in which we choose desired network type and set its corresponding parameters.

If we include TrueTime battery block we can set power supply for chosen TrueTime kernels. As battery is running out of power we can implement some specific behavior to avoid collapse of our control system.



TrueTime Block Library 1.4
Copyright (c) 2006
Martin Andersson, Dan Henriksson and Anton Cervin
Department of Automatic Control, Lund University, Sweden
Please direct questions and bug reports to: truetime@control.lth.se

Fig. 2 Basic TrueTime simulation blocks

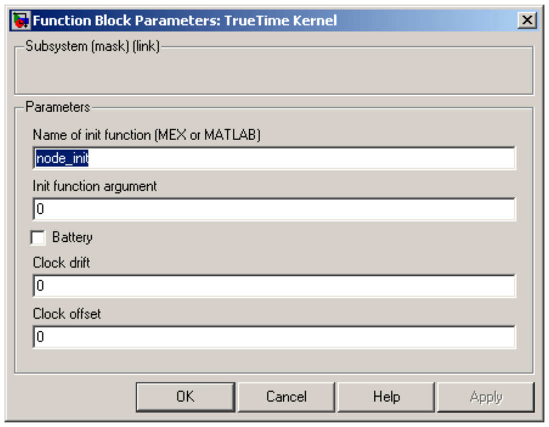
The detailed description of TrueTime blocks

TrueTime kernel block

As we mentioned before it can provide control logic and data processing. Firstly we need to set basic parameters for this block.

TrueTime kernel parameters:

- Init function – defines the name of a M-file or a MEX file where the initialization code is stored.
- Battery – is optional parameter which can be used if device is battery-powered.
- Clock drift and clock offset – can setup value time difference from standard time. When clock drift is set to 0.01 means that device local time will run 1% faster than real-time. Clock offset sets time offset from first run of simulation.


Function Block Parameters: TrueTime Kernel

Subsystem (mask) (link)

Parameters

Name of init function (MEX or MATLAB): node_init

Init function argument: 0

Battery: [ ]

Clock drift: 0

Clock offset: 0

Fig. 3 TrueTime kernel parameters

TrueTime network block

TrueTime Network block simulates transfer of a packet in a local network. When device needs to transfer a message, a “transfer started” signal is signalized to the appropriate input channel in the network block and data can pass through. When transfer is finished then a “transfer finished” signal is signalized on output channel (data leaves output) and new “transfer started” signal is send to the input channel. Transferred messages are stored in a buffer of a device. Message contains information about sender and receiver, message data (for. ex. controller output), data length and optional real-time parameters (priority, etc).

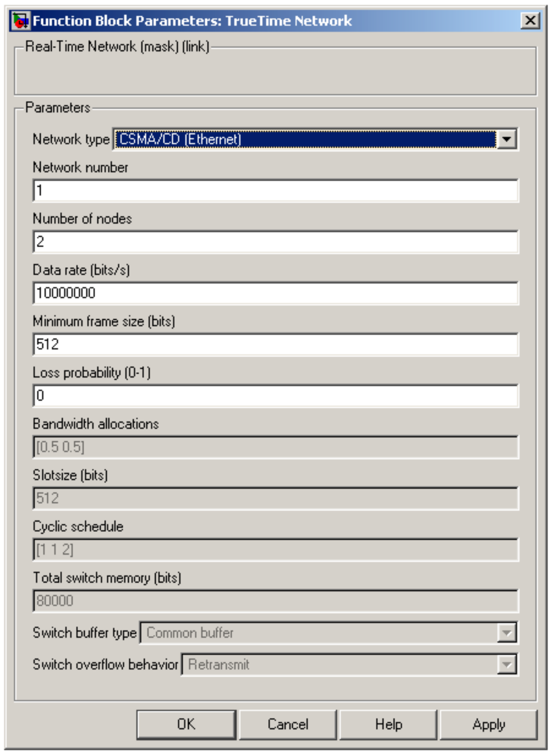
TrueTime supports six models of network:

- CSMA/CD (e.g. Ethernet),
- CSMA/ AMP (for ex. CAN),
- Round Robin (for ex. Token Bus),
- FDMA,
- TDMA (for ex. TTP)
- Switched Ethernet.

Delays caused by spreading of a network signal are not considered, because they are usually ignored in local networks. Simulation is supported only at the packet level. Higher level protocol in the kernel should split long messages to packets. Example of TrueTime network settings dialog is on the Fig. 4.

TrueTime Network Parameters:

- Network number – it is a network ID. It starts from one and every separate network should have its own number.
- Number of nodes – defines number of devices in the network.
- Data rate (bits/s) – set network speed.
- Minimum frame size (in bits) – Message shorter than this value are filled to fit this value.
- Pre-processing delay (s) – delay after message is sent
- Post-processing delay (s) – delay after message is received
- Loss probability (0-1) – probability of packet drop in the network


Function Block Parameters: TrueTime Network

Real-Time Network (mask) (link)

Parameters

Network type: CSMA/CD (Ethernet)
Network number: 1
Number of nodes: 2
Data rate (bits/s): 10000000
Minimum frame size (bits): 512
Loss probability (0-1): 0
Bandwidth allocations: [0.5 0.5]
Slotsize (bits): 512
Cyclic schedule: [1 1 2]
Total switch memory (bits): 80000
Switch buffer type: Common buffer
Switch overflow behavior: Retransmit

Fig. 4 TrueTime network parameters

TrueTime control - simple example

Now we demonstrate usage of TrueTime toolbox on this simple practical example (Fig. 5). In the simulation scheme illustrated below is designed network control system where can be connected together only five network devices (so we set number of nodes in settings dialog of network block to 5), currently only three are used and they operate on the same network. There is one A/D device (Node 1), one D/A device (Node 2) and last device acts as a controller (Node 3).

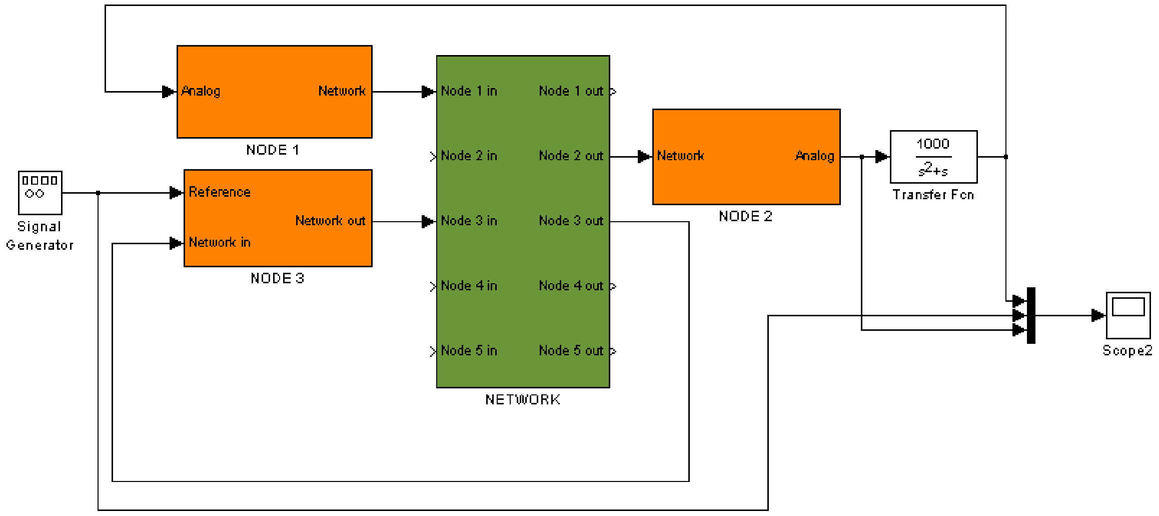


Fig. 5 Simple TrueTime network control system

Simulation model can be described as follows: Node 1 periodically converts analog signal from process output to digital value and sends it to the Node 3. Node 3 after receiving of message from Node 1 calculates controller output (according to corresponding control algorithm) and sends data to Node 2. Node 2 converts received data to the analog signal and sends it to the process.

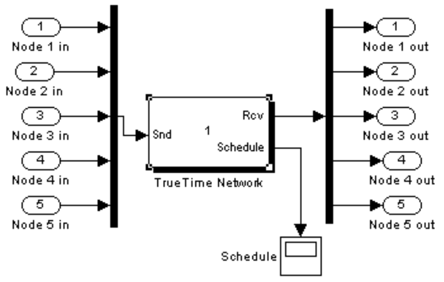


Fig. 6 Detail of subsystem with TrueTime Network block

Network devices (nodes) are simulated as subsystems with TrueTime kernel block (detail of subsystem for Node 1 is in Fig. 7). Node 1 uses one A/D channel (A/D) in the input part and one network output (Snd) on the output. Other nodes are designed similar to Node 1, with a small modification (they can use Rcv input from network, or D/A analog output, etc.).

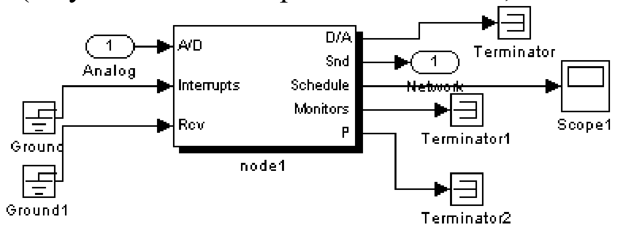


Fig. 7 Example of Node 1

Node 1 periodically converts data from analog input and sends data to the network. It uses following init function which is called once in TrueTime kernel (line by line description is shown in inline comments) when simulation starts:

```
function node1_init
  % node1 reads analog data from process (act as sensor)
  % init TrueTime kernel for node1
  ttInitKernel(1, 0, 'prioFP');% number of A/D inputs,number of D/A outputs,priority
  % init parameters of task
  data.y = 0; % first input data is 0
  offset = 0; % time offset is 0
  period = 0.010; % 10 ms, sets period for execution
  prio = 1; % basic priority, lower numbers = higher priorities

  % Periodically executed task
  % ttCreatePeriodicTask (unique name, time offset of first task, cyclic period,
  % priority, cyclically executed m-file, local memory of task - to store data)
  ttCreatePeriodicTask('sens_task', offset, period, prio, 'node1code', data);

  % Initialization of interrupt driven handler
  % ttCreateInterruptHandler(unique name, priority, name of m-file)
  % when message arrives node1msg is called
  ttCreateInterruptHandler('nw_handler', prio, 'node1msg');
  ttInitNetwork(1, 'nw_handler'); % node 1 in the network
```

Every device which contains block TrueTime kernel must be initialized by function ttInitKernel(number of inputs, number of outputs, priority). It sets number of used analog inputs and outputs and also device priority. Every device in the network must be also initialized in the network with function ttInitNetwork(id of device, name of network handler function). It sets network address of node and also defines which handler is executed when a new message arrives. If we need a periodically executed task on device/node, we can use function ttCreatePeriodicTask(unique name, time shift from simulation start, period, priority, name of m-file, local data). We can create several tasks with predefined period, priority, name of periodically executed m-file and also define some data structure which holds local data for device.

In our example M-file node1code.m will be periodically executed (every 10ms) and it will send digital signal to the network according to process output. Parameter seg stands for sequentially executed segments, firstly we start with value 1 and we process input values, next if seg has value 2 we process output values, if it is 3, we are finished and we can exit node1code.m file. Function ttAnalogIn periodically reads data from input port 1 and stores its value to the structure. Function ttSendMsg periodically sends 80 bits of converted analog signal to the device with address 3 (node 3).

```
function [exectime, data] = node1code(seg, data)
  switch seg,
   case 1,
    data.y = ttAnalogIn(1); % read analog input no 1,
    exectime = 0.00005; % we set executing time of read operation
   case 2,
    ttSendMsg(3, data.y, 80); % send 80 bits to node 3
    exectime = 0.0004; % we also set delay of this operations
   case 3,
    exectime = -1; % we are finished, don't do delay
  end
```

Init function of node 2:

```
function node2_init
  % node2 converts data from network to analog data (act as actuator)
  % Init kernel
  ttInitKernel(0, 1, 'prioFP'); % 0 A/D inputs, 1 D/A output
  % Create actuator task
  deadline = 100;  % deadline time for node2code execution
  prio = 1;
  % node2msg is called only when something is received
  ttCreateInterruptHandler('nw_handler', prio, 'node2msg');
  % node2code is executed in node2msg (see below)
  ttCreateTask('act_task', deadline, prio, 'node2code');
  % Initialize network
  ttInitNetwork(2, 'nw_handler'); % node #2 in the network
```

Init function of node 2 does not contain any declaration of a periodical task, because it does not send data to the network, it only receives data from network. If is device supposed to receive data from network we have to only declare an interrupt handler function ttCreateInterruptHandler(unique name, priority, name of executed m-file) to handle received messages from network.

Interrupt handler function:

```
function [exectime, data] = node3msg(seg, data)
  ttCreateJob('act_task') %when message arrives execute act_task
  exectime = -1; % job is finished

function [exectime, data] = node2code(seg, data)
  switch seg,
   case 1,
    data.u = ttGetMsg; % read data from network
    exectime = 0.0005;
   case 2,
    ttAnalogOut(1, data.u) % store value data.u to port 1
    exectime = -1; % finished
  end
```

Function ttGetMsg reads message from network and convert and send it to analog output 1 via function ttAnalogOut(output port number, analog value)

Node 3 is combination of both devices (node 1 and node 2), it receive messages from node 1, calculates controller output and send data back to network to the node 2.

Init function of Node 3:

```
function node3_init()
 % node3 reads data from node 1 (sensor), calculates controller output and send it
 % to node 2 (actuator)
 % Init kernel
 ttInitKernel(1, 0, 'prioFP'); % one A/D input, zero D/A outputs

 % Control parameters
 h = 0.010;
 N = 100000;
 Td = 0.035;
 K = 1.5;

 % PID controller parameters (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;

 % PID controller task
 deadline = h;
 prio = 2;
 ttCreateTask('pid_task', deadline, prio, 'node3code', data);

 % Initialization of network
 ttCreateInterruptHandler('nw_handler', prio, 'node3msg');
 ttInitNetwork(3, 'nw_handler');
```

Function ttCreateTask creates a task pid_task which calculates control output in a single pass. This is not enough for our functionality, because this task is not cyclic and it is needed to call it regularly. We will use interrupt handler nw_handler (with code defined in node3msg) created by function ttCreateInterruptHandler. Every time when a message from network arrives this interrupt is invoked (node3msg is called). In node3msg function we simply execute our task pid_task with ttCreateJob('pid_task') function.

Source code of node3msg:

```
function [exectime, data] = node3msg(seg, data)

  ttCreateJob('pid_task') %execute job pid_task
  exectime = -1;
```

Source code of node3code:

```
function [exectime, data] = node3code(seg, data)
 switch seg,
 case 1,
  y = ttGetMsg;             % read values from node 1
  r = ttAnalogIn(1);        % read reference analog value
  P = data.K*(r-y);         % calculation of PID
  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 80 bits with controller output value
  exectime = -1;
end
```

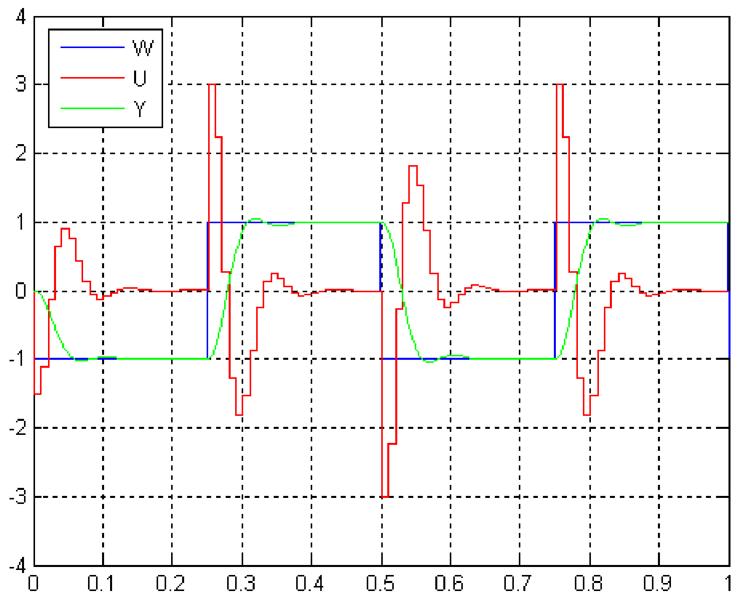


Fig. 8 Simulation result of simple TrueTime control

Simulation results for a motor control system

In last part of this article we show you how can vary simulation results on the same simulation scheme but with different network types. We only mention basic information about this control system (in Fig. 9), because details are out of scope of this article. We have to control two simple DC motors to navigate pen to the reference points (one motor sets x-axis, second motor sets y-axis). So we set two PID controllers and setup the control network. Network contains two actuators (node2, node 4, they set values to motors), two sensors (node 1, node 6, they read positions of motors) and two computers for PID calculation (node 3, node 5). Simulation results with different networks and parameters are in Fig. 10 – Fig. 17.

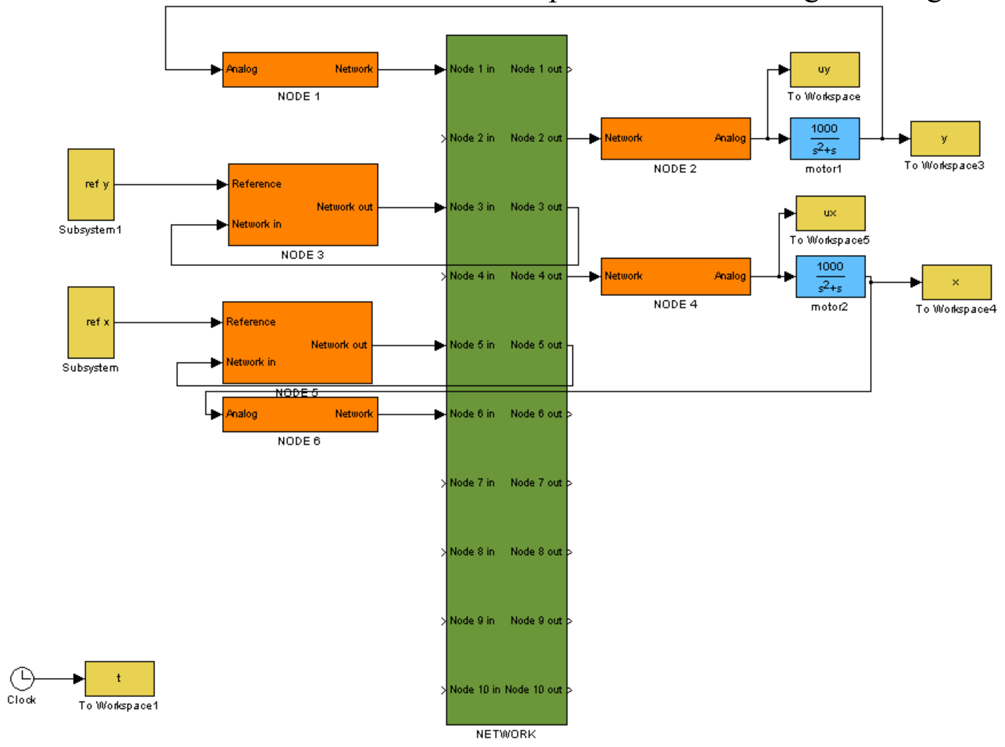


Fig. 9 Simulation scheme of simple TrueTime control

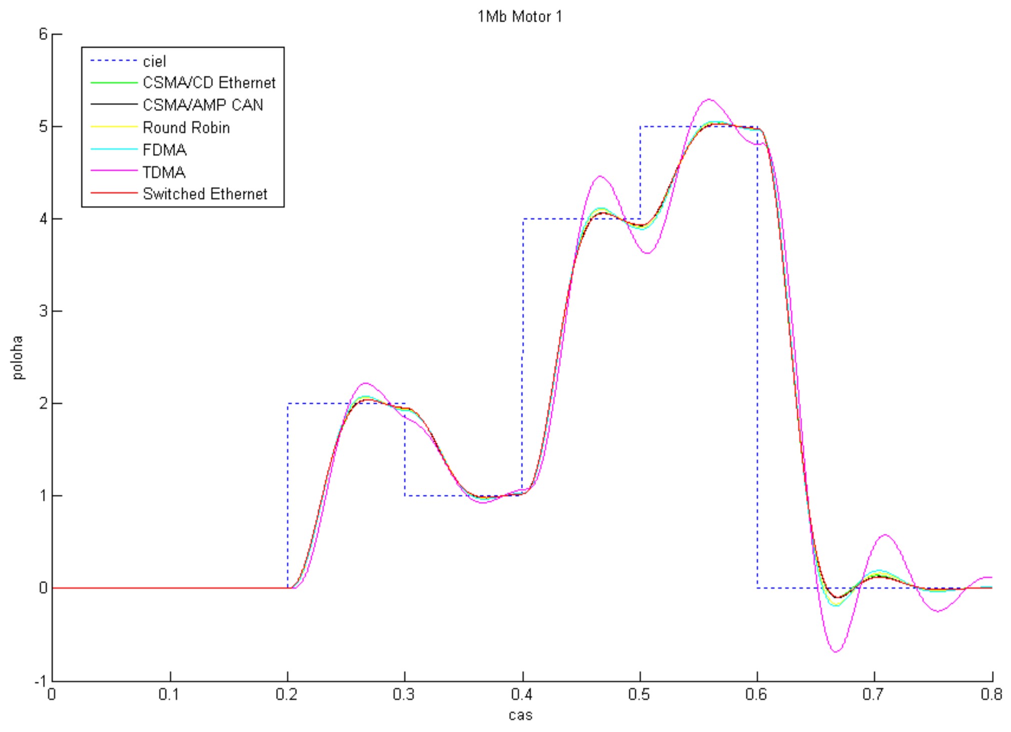


Fig. 10 Simulation result for 1MB/s network speed, no packet loss, motor 1

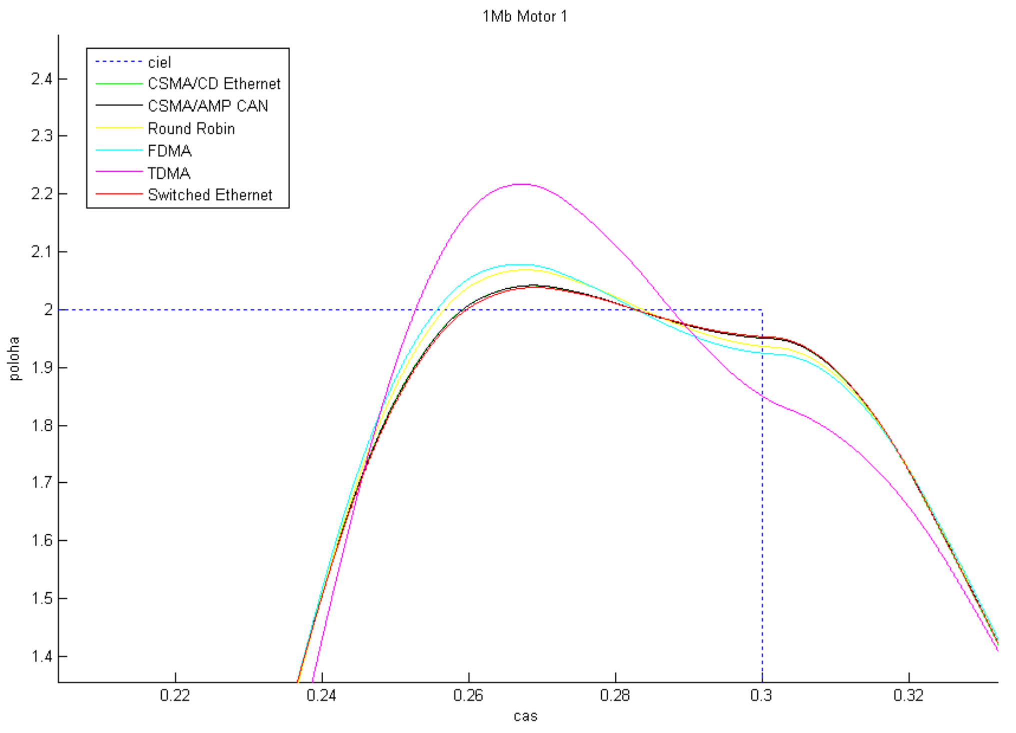


Fig. 11 Simulation result for 1MB/s network speed, no packet loss, motor 1 - detail

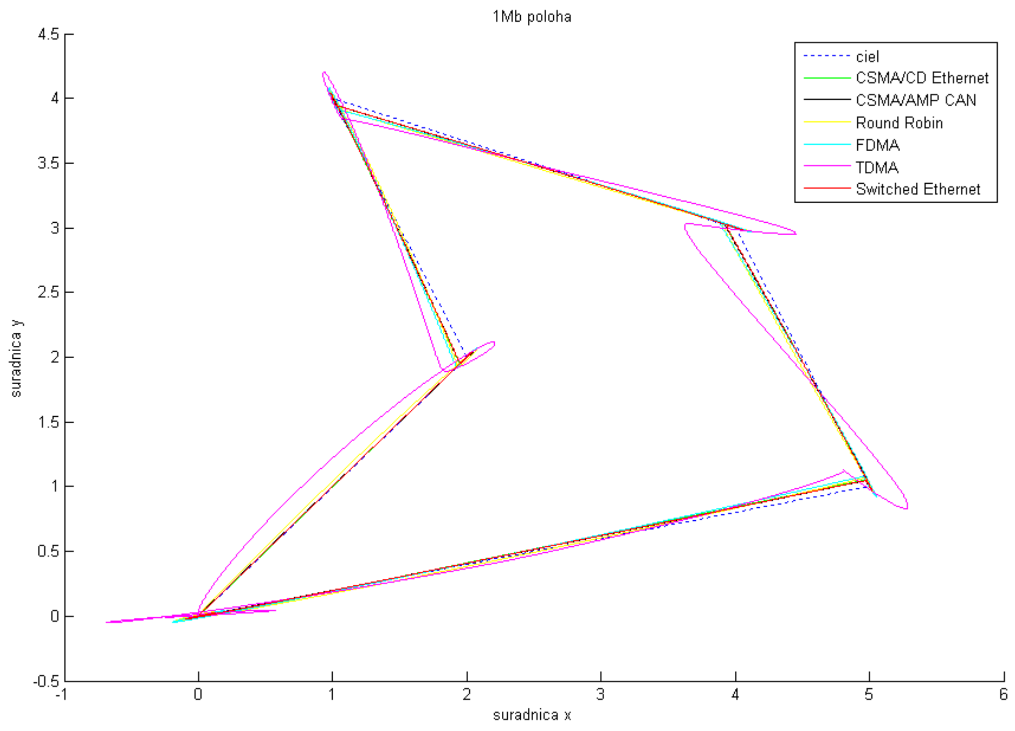


Fig. 12 Simulation result for 1MB/s network speed, no packet loss - position x,y

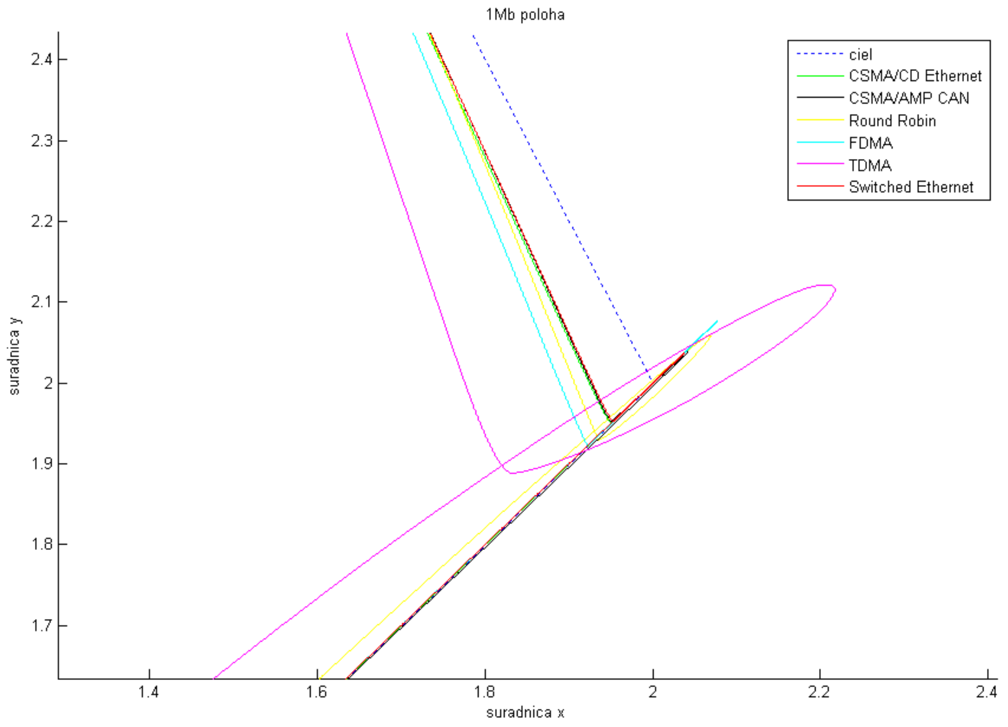


Fig. 13 Simulation result for 1MB/s network speed, no packet loss - position x,y detail

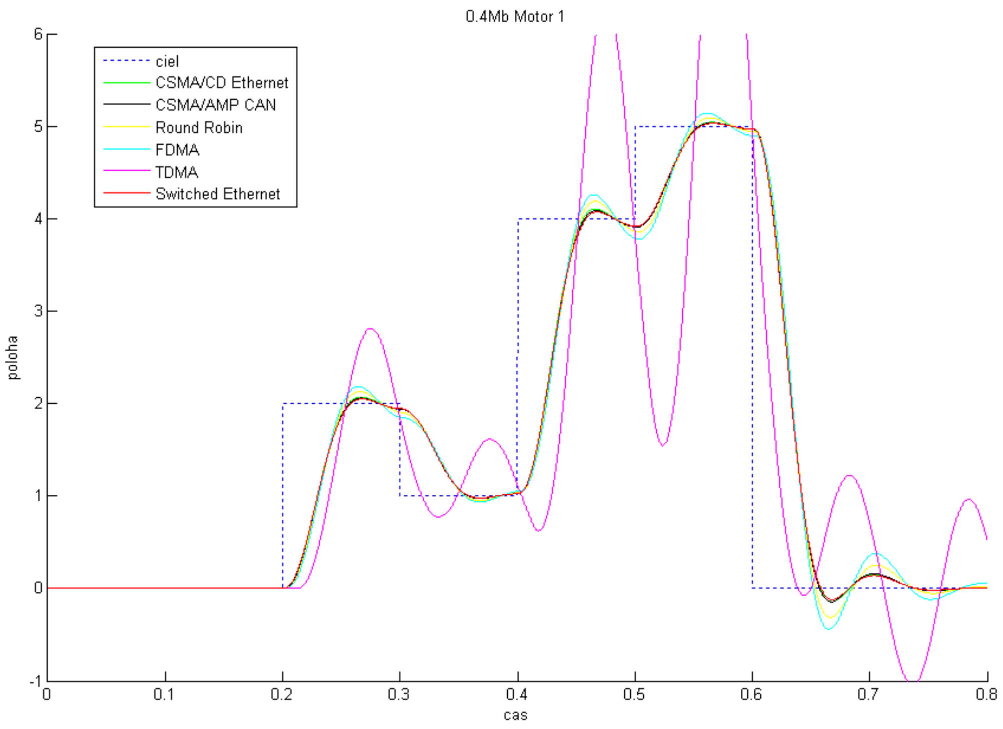


Fig. 14 Simulation result for 0,4 MB/s network speed, no packet loss, motor 1

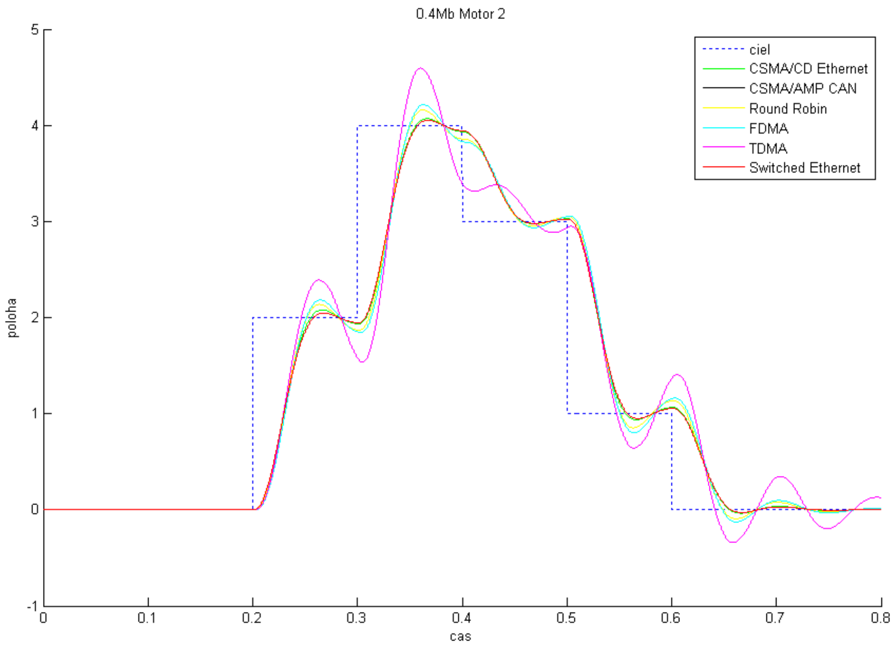


Fig. 15 Simulation result for 0,4 MB/s network speed, no packet loss, motor 2

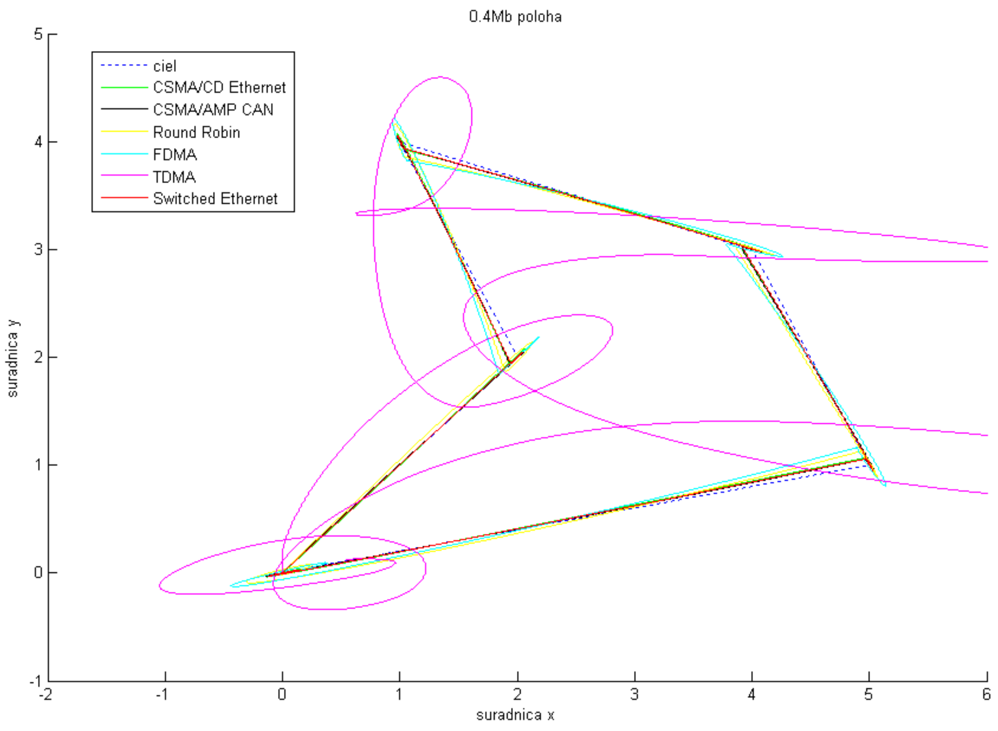


Fig. 16 Simulation result for 0,4 MB/s network speed, no packet loss – position x,y

As we can see from results, motor positioning accuracy varies from chosen network type and network speed.

Conclusion

TrueTime toolbox is simple but very powerful network simulation toolbox which can effectively simulate real-time network control systems. It is highly customizable system with rich set of network types and parameters. TrueTime is a great tool for testing and modeling real-time control systems with several running tasks and network communication. Analyzing output results of network modeled systems can improve quality and ensure stability of real-time closed loop control systems and avoid problems in practical applications.

Acknowledgement

The paper was supported by scientific project Vega 3100 and by Slovak Research and Development Agency, Grant No. APVV-20-023505.

References

[1] The MathWorks, "Getting started with Matlab, version 7.1", 2005
[2] The MathWorks, "Getting started with Simulink, version 6", 2005
[3] Lund University, TrueTime toolbox, www.control.lth.se/truetime

Ing. Tomáš Chvostek
tomas.chvostek@stuba.sk

Bc. Adrián Krátky
adrian.kratky@gmail.com

Ing. Martin Foltin, PhD.
foltin@syprin.sk