Deterministic Transport Protocol Verified by a Real-Time Actuator and Sensor Network Simulation for Distributed Active Turbulent Flow Control

Marcel Dueck, Mario Schloesser, Stefan van Waasen and Michael Schiek

Abstract Total drag of common transport systems such as aircrafts or railways is primarily determined by friction drag. Reducing this drag at high Reynolds numbers ($<10^4$) is currently investigated using flow control based on transversal surface waves. For application in transportation systems with large surfaces a distributed real-time actuator and sensor network is in demand. To fulfill the requirement of real-time capability a deterministic transport protocol with a master slave strategy is introduced. With our network model implemented in Simulink using TrueTime toolbox the deterministic transport protocol could be verified. In the model the *Master-Token-Slave* (*MTS*) protocol is implemented between the application layer following the IEEE 1451.1 smart transducer interface standards and the Ethernet medium access protocol. The model obeys interfaces to the flow control and the DAQ-hardware allowing additional testing in model in the loop simulations.

Keywords TrueTime · Real-Time transport protocol · Distributed actuator and sensor network · Network model

M. Dueck (✉) · M. Schloesser · S. van Waasen · M. Schiek
Central Institute of Engineering, Electronics and Analytics ZEA-2: Electronic Systems, Forschungszentrum Juelich GmbH, Wilhelm-Johnen-Straße, 52428 Juelich, Germany
e-mail: m.dueck@fz-juelich.de
URL: http://www.fz-juelich.de

M. Schloesser
e-mail: m.schloesser@fz-juelich.de

S. van Waasen
e-mail: s.van.waasen@fz-juelich.de
URL: https://www.uni-due.de/

M. Schiek
e-mail: m.schiek@fz-juelich.de

S. van Waasen
University of Duisburg-Essen, Faculty of Engineering, 47057 Duisburg, Germany

A. Nagar et al. (eds.), *Proceedings of 3rd International Conference on Advanced Computing, Networking and Informatics*, Smart Innovation, Systems and Technologies 44, DOI 10.1007/978-81-322-2529-4_3

1 Introduction

In the FOR1779 research group "Active drag reduction by transversal surface waves" [1] fundamental research in the field of active drag reduction in high Reynolds numbers is done based on both wind tunnel experiments and numerical studies. The final aim is to realize turbulent flow control based on transversal surface waves by a distributed real-time actuator and sensor network.

The flow control will be embedded into a cascade control loop. Where the task of the outer loop is to optimize drag reduction by adapting wave parameters such as amplitude, frequency and wavelength. The inner loop is responsible for the wave control and thus minimizes deviations of the surface from the desired wave form.

The development of the distributed actuator and sensor network is based on modelling the network using Simulink and the TrueTime toolbox [2]. The model is used to investigate the most efficient topology and communication flow for the distributed turbulent flow control. The required real-time behavior will be assured by deterministic network communication. The network simulation is designed as a model in the loop approach to be used either as stand-alone network simulation or to utilize flow control experiments in the wind tunnel. For this the model includes interfaces to the flow control and the DAQ-hardware driving the actuators and sensors [3].

In Sect. 2 the model based on IEEE 1451.1 smart transducer interface standards is described. In Sect. 3 the three layer model and its origin is explained. Section 4 deals with the developed real-time *Master-Token-Slave* protocol to cover the missing features in the communication model. In Sect. 5 evaluation of the network behavior is presented as the verification of the proposed transport protocol. In Sect. 6 the work is summarized and future steps are presented.

2 Model

Data exchange in the actuator and sensor network is based on the IEEE 1451.1 smart transducer interface standards [4] and the node names are also chosen referring to this standard as network capable application processor (NCAP) and smart transducer module (STM).

The different node types are representing the interface node ($NCAP_I$), controller nodes ($NCAP_C$), nodes for actuation and local control (STM_A) and sensor nodes (STM_S) to measure data for the flow control (see Table 1). The strategy for a standard communication can be described as follows:

1. Flow control provides wave parameters as actuating variables
2. $NCAP_I$ takes data and forwards it to $NCAP_C$
3. $NCAP_C$ distributes data to STM_A
4. STM_A generates closed loop controlled surface wave
5. STM_S measures friction coefficient and supplies it to $NCAP_C$

Table 1 Description of the modelled nodes

Name	Behavior
$NCAP_I$	Interface for external flow control
$NCAP_C$	Controlling, distributing and gathering data
STM_A	Actuator node with internal closed loop wave control
STM_S	Sensor node for flow control

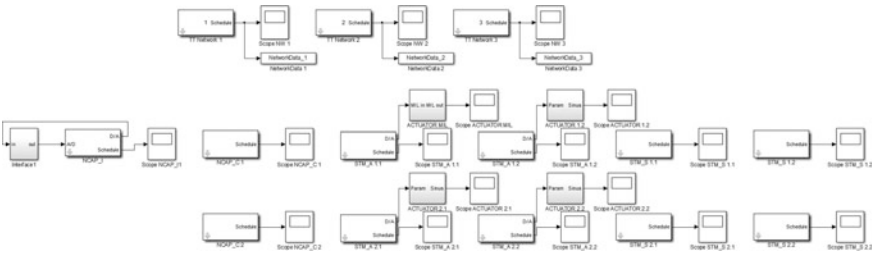

Fig. 1 Simulink network model consisting of eleven TrueTime network nodes and splitted into three different networks. One to connect $NCAP_I$ and two $NCAP_C$ nodes. Each $NCAP_C$ is connected to its assigned STM_A and STM_S nodes in a separate network. There are also interfaces to common Simulink subsystems on $NCAP_I$ and STM_A for providing model in the loop behavior

6. $NCAP_C$ gathers drag reduction information and forwards it to $NCAP_I$
7. $NCAP_I$ provides drag reduction information to flow control

To set up a communication between the nodes the TrueTime toolbox provides a low level network communication which is used here. The parameters of the low level Ethernet network have been verified using a Raspberry Pi testbed [5]. The described approach additionally needs to cover the real-time capability for a predictable message arrival time on the destination node. This *Master-Token-Slave* protocol represents the transport layer between the particular IEEE 1451.1 communication and Ethernet representing the physical layer. The current TrueTime simulation chart is shown in Fig. 1.

3 Three Layer Communication Concept

Referring to the ISO-OSI communication model [6], common real-time protocols are settled either above the transport layer such as RTP [7] or below the transport layer. This is realized either within medium access control protocols including hardware components [8] or industrial bus technology, e.g. fieldbusses [9]. Inside the layer architecture there is a logical communication between equal layers on each

communication partner. Physical communication only takes place on the bottom layer.

Following a simplified OSI-model, a three layer architecture is used in this paper as outlined by Deleuze [10]. It consists of application layer, transport layer and network access layer (see Fig. 2). For the presented purpose every layer uses defined interfaces to communicate with top and bottom layer. No cross-layer communication is allowed. One advantage using a layer strategy is to achieve replaceable protocols.

This approach can be nested in heterogeneous networks. The layers have to be implemented in every node (see Fig. 3). The $NCAP_I$ and *STM* nodes are connected with predefined protocols [3] to real hardware modules. The application layer protocol is divided into three different parts, the node behavior, the parameter format and the IEEE 1451.1 communication layer (see Fig. 3).

The simulation includes the "*Master-Token-Slave*" real-time transport protocol. Transport layer communication to the external hardware is interfaced by UDP to the central control and by TCP/IP to the actuation control [3]. The bottom layer in every node is defined by Ethernet and it is simulated in TrueTime which we recently investigated using a hardware testbed [5].

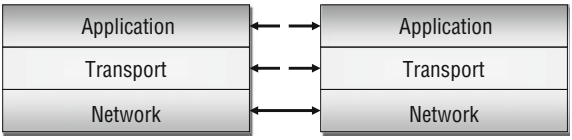

Fig. 2 Simplified protocol stack based on internet protocol architecture. This simplification enables easier modelling of network behavior

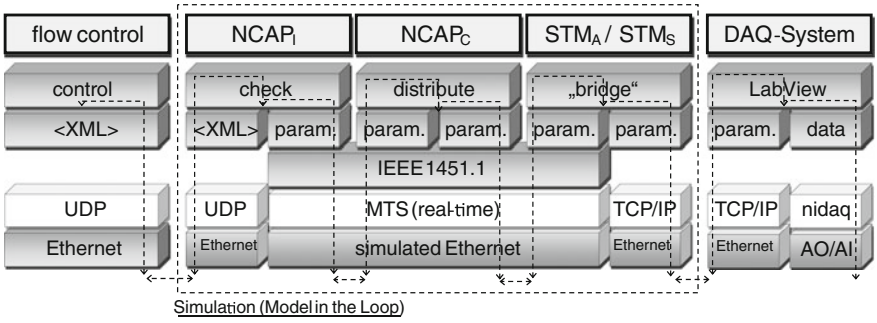

Fig. 3 Package delivery through communication layers from central control algorithm to actuation hardware for analog in- and output. The nodes in the *middle* are parts of the Simulation with UDP and TCP/IP interfaces to the hardware

4 Real-Time *Master-Token-Slave* Protocol

As proposed by Verissimo [11], it is possible to gain real-time behavior in networks by deterministic package distribution. Because application layer and medium access layer have already been defined by IEEE 1451.1 and Ethernet the transport protocol is apparently designated to deliver the real-time capability. The described network model is sealed to the outside and has not to be secured against external errors.

The defined transport protocol is named *Master-Token-Slave*, short *MTS*. The general behavior of interactions is similar to common protocols. An interface is provided to the upper layer using queues to buffer data. The layer below is used by the send- and receive interfaces available in TrueTime. Usually packages are embedded into frames running through network layers. There are two *C*-structures defined in this model: the `mts_msg` structure provides necessary information as outer frame added by the *MTS* transport layer. `I1451_msg` represents the IEEE 1451.1 message and is a dummy structure for the data to send (e.g. commands, functions, parameters, values) by the application itself. So an `I1451_msg` is handed over to the *MTS* interface and can also be expected by receiving messages through the *MTS* transport layer. An example for message exchange can be found in Fig. 4.

The communication is strictly ordered to assure determinism. It is initiated and driven by one master. Slave nodes just have to react on incoming messages. In the

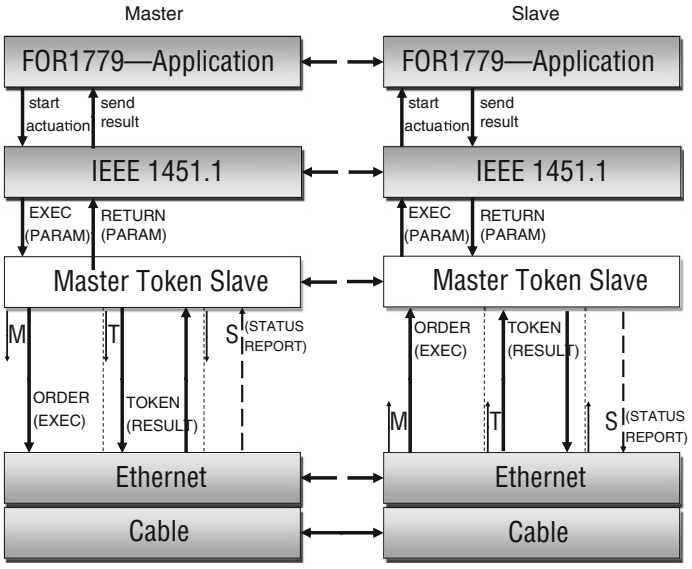

Fig. 4 Example to clarify the *MTS* interactions with the *upper* and *lower* layers. According to the IEEE 1451.1 standard, messages are sent to invoke actions on the other nodes and wait for return values. The *MTS*-protocol sorts it into two phases to send the order and carry the answer. The third phase is optionally provided to the slave node for expressing any errors or giving status reports

described example (see Fig. 1) the communication flow is divided into three different communication phases, nested in two separated network sections. This ensures that every $NCAP_I$ is master for all $NCAP_C$ and every $NCAP_C$ controls his own set of STM_A and STM_S nodes separately. In contrast to a homogeneous network our heterogeneous approach allows higher efficiency in communication since data exchange takes place in different network sections in parallel. To identify messages, every message is marked with a message identification number. Requests, answers and acknowledgements carry the same identification number as corresponding messages.

As shown in Fig. 4 every phase is initiated with a beacon-broadcast by the master. The beacon is marked with a phase number, so if any node misses a beacon it will catch up with the next one and the communication has not to be reset completely. In the first phase, the *Master*-phase, the master node (see Fig. 4, left side) can send information to his slaves. Sent messages are saved for potentially resends.

The second phase is the *Token*-phase. The master node sends a token to the slaves which received a message in the first phase. This is a short message which indicates some free time for the slave node to talk. The token is addressed to a specific slave node, it is not possible to use broadcasts here. If a message is enqueued the slave node responds to the received identification number with a corresponding message. If the queue for answers contains no message, the queue for incoming messages is investigated. In case there is no unprocessed message with corresponding identification number an error is sent back which initiates a resend of the master message in the next appropriate phase.

In the last part, the *Slave*-phase there is a time slot for status messages from slave nodes. Messages are accepted by master and an acknowledgement is sent in the *Master*-phase.

Acknowledgements and beacons as well as the error messages to invoke a resend on *Token*-phase are internal messages and are not noticed in any protocol layer above the transport layer. This is comparable to three-way-handshake or message acknowledgements in TCP/IP [12].

A message broadcast from $NCAP_I$ is possible, whereat broadcasts from $NCAP_C$ are not reasonable because of different node types (STM_A, STM_S). A good solution to save network time would be to send to groups consisting of one type by multicast. This has to be realized in the future.

5 Evaluation of *MTS* by Network Model Simulations

The described *MTS*-protocol is implemented in the model as TrueTime code functions. Several processes have to be started to run the protocol. Most of the time the processes sleep or wait to execute new communications. Interface functions can be used to send and receive messages from the main code function of a node.

The behavior can be analyzed by Simulink scopes which record datasets from the TrueTime kernel blocks. For evaluation and analysis of timing and behavior an example has been implemented using the *MTS*-protocol in TrueTime. The deterministic simulation working on application layer is structured as follows:

1. $NCAP_I$ sends 16 messages to each connected $NCAP_C$
2. $NCAP_I$ waits for answers
3. $NCAP_{C1}$ and $NCAP_{C2}$ answer to every message
4. $NCAP_{C1}$ sends one status messages for every incoming message

The network model used is shown in a Simulink chart in Fig. 1. The messages are sent from and to *MTS*-layer by predefined interfaces. In this experiment there are no additional disturbances like interrupts, dying nodes or lost packages introduced. The message size is in general 1400 Byte which is below the maximum transfer unit of Ethernet at about 1500 Byte. Beacons and token have the size of Ethernet minimum frame size of 64 Byte. The timing windows for the communication phases are equally spaced by $\frac{100}{3}$ ms. The experiment takes 0.3 s which equals three full *MTS*-cycles.

Figure 5 shows the result of this experiment. The network events are initiated by the transport layer protocol in the simulated Ethernet layer. The events are indicated by a peak on the scope-output. Also the limitation in time for the different communication states occurs in the right moment.

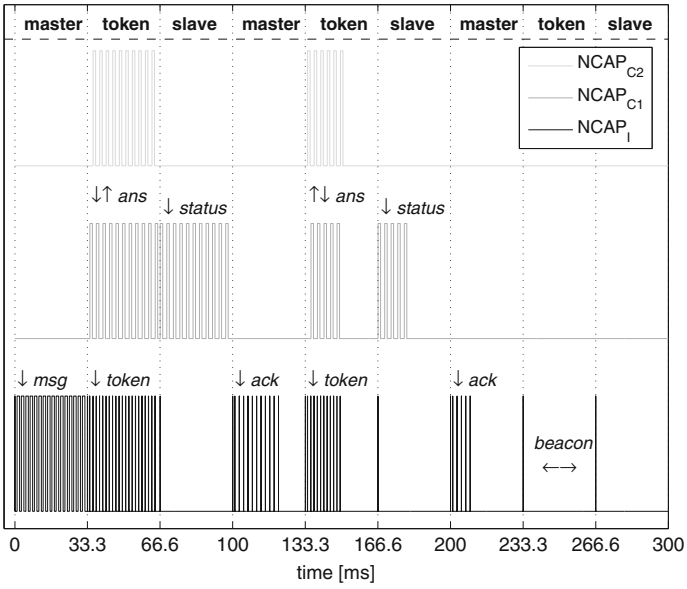

Fig. 5 The result of an experiment for communication between one master ($NCAP_I$) and two slave nodes ($NCAP_{C1}$, $NCAP_{C2}$) shows a full communication which is initiated once by enqueueing some messages on master node and then waiting for reactions on slave nodes

The master permanently controls all slaves. At the start a beacon is sent to indicate the start of *Master*-timeslot. 16 messages are enqueued by $NCAP_I$. The messages are broadcasted to both $NCAP_{C1}$ and $NCAP_{C2}$ (see Fig. 5) within the *Master*-phase. An answer is enqueued by the controller nodes after receiving a message. Then the *Token*-phase is initiated by a second beacon. Tokens are sent to all nodes which should have received a message in the first timeslot. The corresponding answer is read from the queue and sent directly. The end of the *Token*-phase is indicated by another beacon. The following *Slave*-phase is open for status messages by the slave nodes. It is proposed to be used for registration to the network at start of a slave node or for error notifications to the master node. The detailed behavior of the previous described communication is depicted in Fig. 6.

In this example the whole communication was not finished during the first cycle and it is continued within the next cycle. Hereby the master starts with acknowledging the status messages from $NCAP_{C1}$, then the token is sent to both slaves and requested answers are received. Again status messages are sent by $NCAP_{C1}$ and acknowledged by the master. After no further messages occur, the flow is just continued by sending beacons for every new phase.

The result of this experiment proofs the correctness of the deterministic transport layer protocol embedded in the network model. The reaction time in the *Token*-phase from token send until the message is fully received is 1.25 ms. The time of

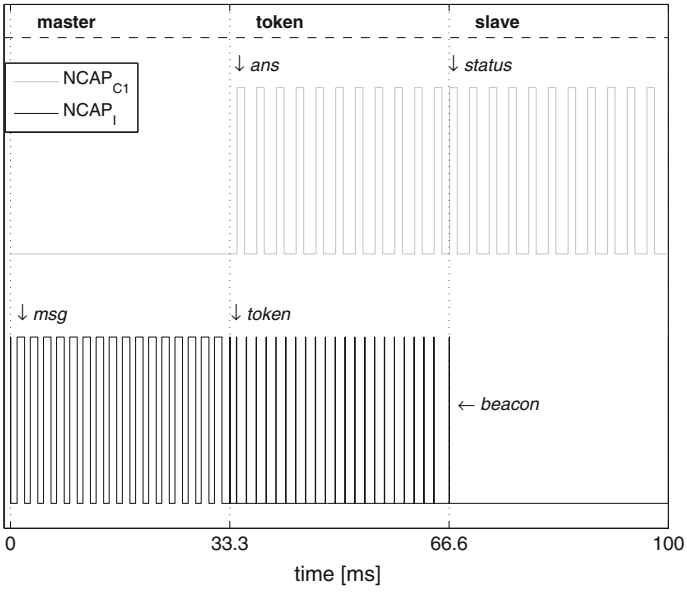

Fig. 6 One communication cycle consisting of three different communication phases is shown. The detailed communication between $NCAP_I$ and $NCAP_{C1}$ highlights the token answer mechanism and the status message part

network occupation and buffer time by beacon, token and acknowledgements is below 1 ms. Messages with the size of 1400 Byte like master messages or token answers need a propagation time of 1.14 ms.

6 Conclusion and Outlook

In this paper we presented a deterministic transport protocol for a real-time actuator and sensor network based on a three layer network architecture. The *Master-Token-Slave* (*MTS*) protocol is implemented between the application layer defined by IEEE 1451.1 smart transducer interface standards and the Ethernet medium access protocol delivered by the TrueTime toolbox. Real-time capability of the protocol is checked using a reduced network model where the response time depends on the length of the three different communication windows. The in-house development allows to adapt the phase-lengths to the sample rate of the flow control.

Error influences like packet loss, node malfunction, restart or blackout have to be covered in a future version of the model. Also collisions which can occur in the last phase of a communication cycle only have to be investigated. To cover e.g. a $NCAP_C$ blackout the network has to be changed from heterogeneous to a full-mesh topology. Multicasts from $NCAP_C$ to STM_A and STM_S for *Master*-phase should be considered to save time supplying the same data to a specific node type.

To summarize a flexible real-time transport protocol has been developed which can now be used to investigate model in the loop simulations to develop distributed real-time actuator and sensor networks for turbulent flow control.

Acknowledgments The authors would like to thank all partners in the research group FOR1779 and acknowledge the funding by the DFG (German Research Foundation).

References

1. DFG. GEPRIS. [12.06.2014]. http://gepris.dfg.de/gepris/projekt/202175528
2. Cervin, A., Henriksson, D., Lincoln, B., Eker, J., Årzén, K.-E.: How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime. IEEE Control Syst. Mag. **23**(3), 16–30 (2003)
3. Dueck, M., Kaparaki, M., Srivastava, S., van Waasen, S., Schiek ,M.: Development of a real time actuation control in a network-simulation framework for active drag reduction in turbulent flow. In: Automatic Control Conference (CACS), 2013 CACS International, pp. 256–261, Dec 2013
4. IEEE std 1451.1-1999, standard for a smart transducer interface for sensors and actuators—network capable application processor (ncap) information model, June 1999. The Institute of Electrical and Electronics Engineers, Inc. 3 Park Avenue, New York, NY 10016-5997, USA
5. Dueck, M., Schloesser, M., Kaparaki, M., Srivastava, S., van Waasen, S., Schiek ,M.: Raspberry pi based testbed verifying truetime network model parameters for application in distributed active turbulent flow control. In: Proceedings of the SICE Annual Conference (SICE), pp. 1970–1975, Sept 2014

6. Day, J.D., Zimmermann, H.: The OSI reference model. Proc. IEEE **71**(12), 1334–1340 (1983)
7. Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V.: RFC 3550: RTP: A Transport Protocol for Real-Time Applications. Technical report, IETF, 2003
8. Carvajal, G., Wu, C.W., Fischmeister, S.: Evaluation of communication architectures for switched real-time ethernet. IEEE Trans. Comput. **63**(1), 218–229 (2014)
9. Tovar, E., Vasques, F.: Real-time fieldbus communications using profibus networks. IEEE Trans. Ind. Electron. **46**(6), 1241–1251 (1999)
10. Deleuze, C.: Content networks. Internet Protoc. J. **7**(2), 2–11 (2004)
11. Verissimom P.: Real-time communication. In: Mullender, S. (ed.) Distributed Systems, pp. 447–490. Addison-Wesley (1993)
12. Cerf, V., Dalal, Y., Sunshine, C.: Specification of Internet Transmission Control Program. RFC 675, December 1974