2010 IEEE-RAS International Conference on Humanoid Robots
Nashville, TN, USA, December 6-8, 2010

Design of an Embedded Networking Infrastructure for whole-Body Tactile Sensing in Humanoid Robots

Emanuele Baglini, Giorgio Cannata, Fulvio Mastrogiovanni

Abstract— **Whole-body tactile sensing, especially when aimed at implementing tactile-based motion control, requires well-defined infrastructures. Requirements related to Real-Time operation, bandwidth and robustness must be carefully addressed. With respect to a tactile technology described in [2], this paper discusses a number of infrastructural solutions based on a Ethernet Real-Time protocol. Associated benefits and drawbacks are discussed. The identified models and solutions are compared both analytically and with extensive numerical simulations.**

I. Introduction

During the past few years, research on humanoid robots devoted much effort on the integration between heterogeneous aspects, such as electromechanical design, software architectures, sensing and control. Humanoids can be considered *distributed* systems where many embedded devices must communicate efficiently and dependably. A typical example includes networked devices (e.g., for low-level control and sensory information processing) communicating with a *master* node, which hosts intensive algorithmic processes, full Operating Systems or gateways towards even more powerful computational machinery.

The problem of *infrastructure* is fundamental. The embedded network can be implemented as a *fieldbus*, with desired characteristics in terms of Real-Time requirements, cycle and response time, as well as robustness. In the context of distributed control, many contributions appeared in the literature. In [6] a first approach at the system level is introduced, where different parts of a robot body are managed by *sub*networks through a *serial* line. In [15], [9], the infrastructure of the HOAP-1 robot is described: Real-Time communication between control nodes and the on-board PC is managed through a simple *USB* cable. *Firewire* (i.e., IEEE 1394 standard) has been used in [12] to model a Real-Time distributed control system for humanoid robot control. CAN bus is used in [16], [19], [17] to implement distributed controllers. Finally, [4] proposes a hybrid network based on Firewire for the upper layer and on CAN for *sub*networks control. These solutions can not be easily adapted when dealing with sensing, and specifically with whole-body tactile sensing: on one hand, bandwidth and Real-Time requirements can be hardly met by adapting already existing infrastructures; on the other hand, although a number of Ethernet-based solutions has been reported [1], [8], a principled discussion about possible implementations for distributed sensing has not been considered yet. The major contribution of this paper is a systematic analysis of Real-Time Ethernet protocols, specifically based on a tactile system presented in previous work [2]. In particular, we analyse the behaviour of Ethernet *Powerlink*[1] and *EtherCAT*[2].

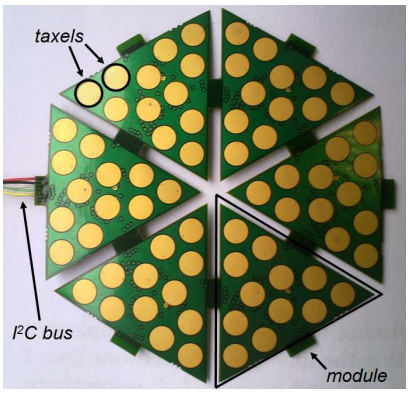

Fig. 1. A hexagonal patch made up of six triangular modules (in *green*).

The paper is organized as follows. Section II introduces the reference architecture, specifically detailing weak points and bottlenecks, as well as a number of Ethernet-based solutions, with a specific emphasis on the most promising protocols: the analysis is performed adopting as indicator the *cycle time*, i.e., the time needed for a master node to update a representation of the status of each slave in the network. Finally, realistic numerical analyses are carried out, which assume to cover entirely with tactile sensors a typically-sized humanoid robot. Conclusions follow.

II. Architecture Design and Analysis

A. The Reference Architecture for Tactile Sensing

The considered *robot skin* [2] is composed of a large number of spatially distributed tactile elements (i.e., *taxels*), organized in *patches*, which are surface compliant structures covering large parts of a robot body[3] (see Figure 1). Since complex contact phenomena are distributed over large robot surfaces (i.e., the *palm*, the *forearm* or the *torso*), each patch

All the authors are with the Department of Communication, Computer and System Sciences, University of Genova, Via Opera Pia 13, 16145, Genova, Italy. Contact author email: fulvio@dist.unige.it.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n. 231500-ROBOSKIN.

[1]Please refer to the official Ethernet *Powerlink* website available at: http://www.ethernet-powerlink.org.

[2]Please refer to the official EtherCAT consortium website available at: http://www.ethercat.org.

[3]The first prototype has been realized as a joint effort between University of Genova and Italian Institute of Technology.

consists of many triangular modules of $3cm$ side, each one comprising a 2D tactile sensing array as well as embedded electronics. Each patch is associated with a microcontroller responsible for data acquisition, early processing and data transfer to higher level control and cognitive processing modules. Patches can be part of a network, thereby implementing physically and logically independent sensing surfaces. Each taxel is a capacitive transducer (golden pads in Figure 1). Up to 12 taxels can be arranged in a triangular module, made up of flexible PCB. Up to 16 triangles can be managed by one microcontroller through 4 I^2C buses (each one controlling 4 triangular modules). Each microcontroller is connected to a CAN bus line, so that the network of microcontrollers in *daisy chain* is connected to a master node (e.g., a PC104) where higher level processing occurs.

Although this design exhibits many interesting features, such as high surface conformance and high modularity, it still lacks a number of desirable features: (i) the skin infrastructure must be adaptive to account for unexpected structure changes, such as in the case of failures; (ii) given the huge amount of distributed tactile information, a huge number of microcontrollers is needed, whereas higher bandwidth and Real-Time requirements for closing tactile-based control loops must be achieved; (iii) primary as well as ancillary electronics (e.g., cables, ports, etc.) must be strongly embedded. The goal of this paper is to address the second and (only in part) the first issues by investigating a networking infrastructure based on $100Mbit$ Real-Time extensions of the Ethernet protocol, aimed at replacing the existing CAN bus between microcontrollers. In our case, each taxel encodes pressure information using a $8bit$ packet. The amount of data $D_{\mu c}$ each microcontroller manages can be computed as:

$$D_{\mu c} = 8bit \times 12taxel \times 16modules = 192byte. \quad (1)$$

Assuming that a whole robot surface can sum up to $1m^2$ (the skin for an adult human is roughly $2m^2$), using current technology around 160 microcontrollers are needed for a complete robot coverage. In fact:

$$\#_{\mu c} = \frac{\overbrace{1m^2}^{area}}{\underbrace{3.89cm^2}_{module\ area} \times \underbrace{16}_{modules\ per\ node}} \simeq 160. \quad (2)$$

Given these estimates, the infrastructure must guarantee that at least a *reactive* tactile-based control loop can be closed in a network of 160 microcontrollers in the span of few milliseconds, e.g., to implement reactive behaviours. One could argue that this requirement can be relaxed by considering subnetworks, e.g., separate for arms, legs and the torso. According to [18], the percentage of skin present in an arm, a leg or in the torso is around – respectively – 10%, 20% and 33%: a possible subnetwork for the whole arm still comprises up to 16 microcontrollers, for a torso up to 55: closing such tactile-based control loops is still a difficult problem.

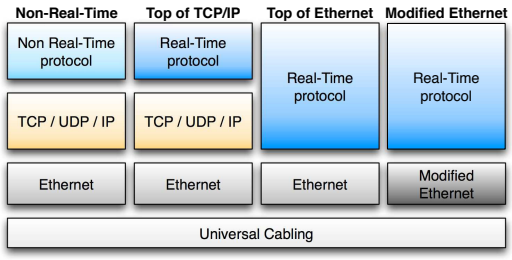

Fig. 2. Different Real-Time Ethernet Typologies.

B. Real-Time Ethernet Extensions

In this Section we focus on two suitable Real-Time Ethernet protocols, namely *Powerlink* and *EtherCAT*, which prove to best fit the discussed design requirements.

Ethernet-based Real-Time (RT in short) distributed infrastructures differentiate between non RT and RT services. Non RT applications exploit TCP/IP-based HTTP or FTP for low-priority services whereas, to build RT Ethernet solutions (RTE in short), three main approaches are usually considered [5] (see Figure 2). The first one (referred to as "on top of TCP-IP") is to maintain the TCP/IP-based protocols unchanged and to concentrate all RT modifications in the top layer, such as in *Ethernet/IP*. The second one (i.e., "on top of Ethernet") assumes that TCP/IP protocols are bypassed and Ethernet functionalities are directly accessed: a significant example is *Powerlink*. In the third one the Ethernet infrastructure is modified to boost RT performance (and therefore it is called "modified Ethernet"). Examples include *ProfiNET/IO*, *Sercos III* and *EtherCAT*.

Ethernet/IP, *ProfiNET IO* and *Sercos III* [4] do not meet the design requirements, for a number of reasons related to non-deterministic communication delays, low performance, need for special hardware and forced topologies. We deepen our analysis for two possible candidate solutions, namely Ethernet *Powerlink* and *EtherCAT*, because of their suitability to meet the requirements defined in Section II-A. *Powerlink* is based on software implementations and exploits only standard COTS hardware, whereas *EtherCAT* requires dedicated hardware at the node interface, but of easy embeddability. According to a well-established trend in literature [10], [11], [13], [7], [14], [3], [5], the two RTE protocols are compared with respect to *cycle time*.

1) Ethernet Powerlink: Ethernet *Powerlink* (EPL in short) is a master-slave standard classified in the category "on top of Ethernet", which implements RTE services by specifying a special protocol type, referred to as *EtherType*, in the usual Ethernet frame. A master (i.e., the *managing node* MN) polls the *controlled nodes* CNs sending and/or requesting data. EPL supports different network topologies, i.e., *tree*, *star*, *ring* and *daisy chain* with a maximum of 240 nodes. Every cycle (see Figure 3), MN sends a `SoC` (i.e., Start of Cycle) frame in broadcast: after a `IFG` (i.e., InterFrame Gap) period

[4]Please refer to the related websites: `http://www.odva.org`, `http://www.profibus.com` and `http://www.sercos.com`.

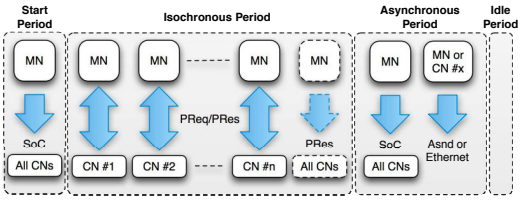

Fig. 3. Ethernet Powerlink Cycle.

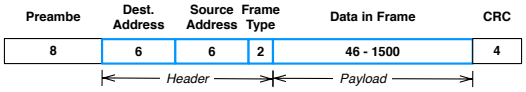

Fig. 4. Ethernet Frame.

as specified by the Ethernet standard, the Isochronous Period starts. Then, MN sends a PReq (i.e., Poll Request) frame carrying the output data to CNs which reply in broadcast with PRes (i.e., Poll Response) data frames. After all the CNs are polled, MN can optionally broadcast a PRes frame before the end of the Isochronous Period. After that, MN sends a SoA (i.e., Start of Acyclic) frame to all the CNs thereby initiating the Acyclic Period. After this message, MN or only one of the CN can send an acyclic message. Two types of *async* messages exist in EPL, namely *Powerlink* ASnd (i.e., Asynchronous Send) and Legacy Ethernet.

Assumptions. As described in Section II-A, we assume that: (i) all the CNs are polled every cycle; (ii) MN does not send data to CNs; (iii) acyclic phases are not present; (iv) the network topology is *daisy chain*; (v) the length of each cable is $0.01m$; (vi) the time spent by MN and CNs to elaborate requests and to send replies is not considered.

Cycle Time analysis. With *cycle time* we mean the minimum time necessary to exchange input/output data between the MN and all the other CNs. The cycle time can be computed as:

$$T_{epl} = T_{st} + T_{is} + T_{as} + T_{id}, \tag{3}$$

where T_{st} is the duration of the Start Period plus a IFG safety margin, T_{is} and T_{as} are the duration of the isochronous and asynchronous periods, whereas T_{id} is an idle period. T_{st} can be obtained as:

$$T_{st} = \frac{P_{\texttt{SoC}} + E_{ov}}{b} + T_{\texttt{IFG}}, \tag{4}$$

where $P_{\texttt{SoC}}$ is the $46byte$ SoC frame payload, E_{ov} is the $26byte$ overhead due to Ethernet (see Figure 4: $8byte$ preamble, $6byte$ Destination Mac Address, $6byte$ Source Mac Address, $2byte$ EtherType and $4byte$ CRC), b is the byte rate (amounting to $12.5byte/\mu s = 100Mbit/s$), whereas $T_{\texttt{IFG}}$ is the InterFrame Gap, which is at its minimum $0.96\mu s$ from the Ethernet specification. T_{is} can be expressed as:

$$T_{is} = T_{rq} + T_{rs} + D_{mn} + D_{cn} + D_{nk}, \tag{5}$$

where T_{rq} is the time necessary for MNs to send all the PReq requests, T_{rs} is the time necessary for all CNs to send PRes frames, whereas D_{mn} and D_{cn} are the delays introduced by MN and CNs. Finally, D_{nk} represents all the delays introduced by network components including hubs and propagation. These values are obtained as:

$$T_{rq} = \frac{n(H_{\texttt{PReq}} + \texttt{data} + \texttt{pad} + E_{ov})}{b}, \tag{6}$$

where n is the number of CNs, $H_{\texttt{PReq}}$ is the $10byte$ header size of the PReq message, data is the amount of the data sent by the MN to each CN, whereas pad is the value of the padding bytes necessary to reach the $46byte$ minimum size of an Ethernet frame. Analogously:

$$T_{rs} = \frac{n(H_{\texttt{PRes}} + \texttt{data} + \texttt{pad} + E_{ov})}{b}, \tag{7}$$

where $H_{\texttt{PRes}}$ is the $10byte$ header size of the PRes message, and D_{mn} and D_{cn} are assumed to be $1\mu s$. D_{nk} in Equation 5 is given by:

$$D_{nk} = 2n(D_{hub} + D_{medium}) + 2\sum_{i=1}^{n} i(D_{hub} + D_{medium}), \tag{8}$$

where D_{hub} is the $0.3\mu s$ delay introduced by each hub present in each node and D_{medium} is the delay due to the propagation in the medium. Going back to Equation 3, T_{as} is the sum of the following terms:

$$T_{as} = T_{\texttt{SoA}} + T_{\texttt{ASnd}}, \tag{9}$$

where $T_{\texttt{SoA}}$ is the time needed to transmit the SoA frame and T_{fr} is the time needed for the ASnd frame transmission. Specifically, $T_{\texttt{SoA}}$ is obtained as:

$$T_{\texttt{SoA}} = \frac{H_{\texttt{SoA}} + E_{ov}}{b} + T_{\texttt{IFG}}, \tag{10}$$

where $H_{\texttt{SoA}}$ is the $46byte$ header size of the SoA message. $T_{\texttt{ASnd}}$ is computed using the following Equation:

$$T_{\texttt{ASnd}} = \frac{H_{\texttt{ASnd}} + \texttt{data} + \texttt{pad} + E_{ov}}{b}, \tag{11}$$

where $H_{\texttt{ASnd}}$ is the $5byte$ header size of the ASnd message, and data is the amount of data sent during the *async* phase.

2) EtherCAT: EtherCAT (ECAT in short) is a "modified Ethernet" standard. Every cycle an Ethernet frame is sent by the master node to all the slave nodes, each node being provided with an input and an output Ethernet interface. Each slave reads its own data and communicates to the master inserting the data in the flowing frame, which reaches again the master node. Each node operates "on the fly": frames only flow between the two Ethernet interfaces of each node, i.e., they are read by the first and written to the second, *byte per byte*. If the transmitted data can not fit within one Ethernet frame, more frames can be used.

ECAT adopts standard Ethernet frames encapsulating *telegrams* as shown in Figure 5. Slaves elaborating data "on the fly" must be equipped with dedicated hardware, implemented in the so-called *ESC* chip and it is usually an *ASIC* or

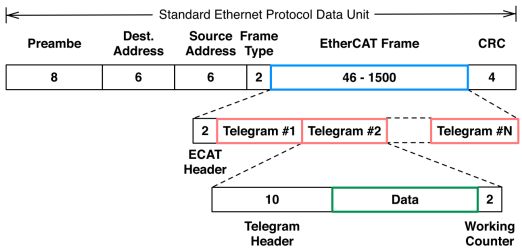

Fig. 5. EtherCAT Frame.

an *FPGA*, which decouples the actual fieldbus from the microcontroller on the slave node[5].

Considering a *daisy chain* configuration, we assume that $k = 0$ and $k = 1$ are the input and output ports of the master node, whereas higher values for k are associated with slave nodes. The *ESC* chip on a given slave receives an Ethernet frame on the port k and forwards it to port $k+1$. If neither port $k+1$ exists nor any connection is present to port $k+1$, the Ethernet frame is then forwarded to port 0, and the loop is closed. Each Ethernet frame in the payload contains a $2byte$ ECAT header and n ECAT telegrams. Each telegram contains a $10byte$ header, which specifies the operation type (e.g., *read*, *write*, *read/write*, etc.), a data field, where master and/or slave nodes write information (i.e., from master to slaves, from slaves to the master or between slaves) and a $2byte$ working counter, which is used by the master to verify the correctness of the execution of the operation specified in the telegram header. With respect to fault tolerance, since the Ethernet frame CRC is checked by each device, bit errors are not only detected immediately (i.e., between two nodes), but can be also located exactly by checking the error counters.

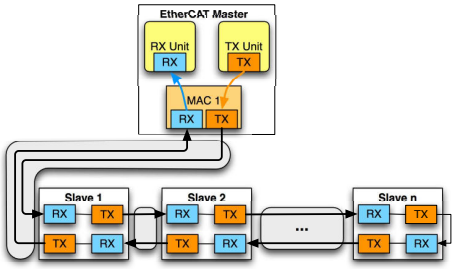

Fig. 6. EtherCAT: non redundant topology.

We consider two network topologies, i.e., "open ring" (see Figure 6) and "closed ring" (see Figure 7), which is aimed at guaranteeing networking even in presence of broken nodes or damaged cables. As Figure 8 shows, in case of a broken connection, the ECAT loop can be closed by folding back the fieldbus. All slave devices with two or more Ethernet ports support the cable redundancy feature. According to ECAT standard benchmarks, the recovery time in case of cable failure is shorter than $15\mu s$.

[5]The *ESC* chip can be strongly embedded due to its small footprint: examples of common sizes are Beckhoff *ET1100* (BGA128, $10 \times 10mm$) or Beckhoff *ET1200* (QFN48, $7 \times 7mm$).

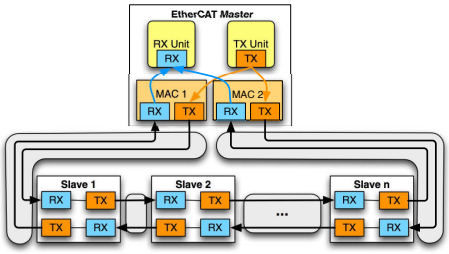

Fig. 7. EtherCAT: redundant topology.

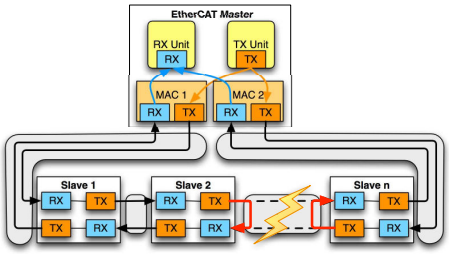

Fig. 8. EtherCAT: redundant topology - cable failure.

Assumptions. Similarly to the previous Section, we assume that: (i) all the slaves are polled every cycle; (ii) the master does not send data to slaves; (iii) acyclic phases are not present; (iv) network topology is *daisy chain* (both the non redundant and the redundant cases are considered); (v) the length of each cable between nodes is $0.01m$; (vi) only the communication with the *ESC* chip is considered. This is a realistic model of the system behaviour: ECAT slaves must be ready to send data as soon as the ECAT frame flows through the corresponding ports, and therefore processes running on the node microcontroller run intrinsically in parallel with respect to the frame itself.

Cycle Time analysis. With *cycle time* we mean the time necessary for the ECAT frame to transport data between the master and all the other slaves. The master node imposes the cycle time by sending the ECAT frame through the fieldbus. Differently from existing literature, this analysis considers issues related to processing requirements in slave nodes [11], [14]. The cycle time can be evaluated as follows:

$$\begin{aligned} T_{ecat} &= n(T_{ecat\_fwd} + T_{\mathtt{data}}) + N_{cables} \times T_{medium} \\ &+ n_{fr} \times T_{eth} + T_{ecat\_ov} + T_{\mathtt{pad}}, \end{aligned} \quad (12)$$

where n is the number of slaves, T_{ecat_fwd} is the time necessary, for each slave, to forward the received frame from interface k to $k+1$ (estimated to be up to $1.35\mu s$ in non-redundant configurations, and $0.675\mu s$ in redundant configurations), $T_{\mathtt{data}}$ is the time necessary to send the payload specific for each node, and it is evaluated as:

$$T_{\mathtt{data}} = \frac{D_{\mathtt{data}}}{b}. \quad (13)$$

In Equation 13, $D_{\mathtt{data}}$ is the amount of data sent to each slave and b is the byte rate (in the considered scenario it amounts to $12.5byte/\mu s$ = $100Mbit/s$). Going back to Equation 12, N_{cables} is the number of cables in the whole

network (it equals to $2n$ for an "open ring" topology, and to $n+1$ for a "closed ring topology"), T_{medium} is the delay due to the propagation in the medium (it is approximately $0.0005\mu s$ for a $0.01m$ cable), whereas n_{fr} is the number of Ethernet frames needed to send all data to all the slaves, which can be evaluated as:

$$n_{fr} = \left\lceil \frac{max_{tgram}}{n} \right\rceil, \tag{14}$$

where max_{tgram} is given by:

$$max_{tgram} = \left\lfloor \frac{D_{eth\_max\_payload} - D_{ecat\_hd}}{D_{\mathtt{data}} + D_{ecat\_tgram\_hd} + D_{ecat\_wc}} \right\rfloor. \tag{15}$$

In Equation 15, $D_{eth\_max\_payload}$ is the $1500byte$ maximum size of an Ethernet frame payload, D_{ecat_hd} is the $2byte$ ECAT header size, $D_{ecat\_tgram\_hd}$ is the $10byte$ header size for each telegram and D_{ecat_wc} is the $2byte$ working counter size for each telegram. In Equation 12, T_{eth} is the time needed to send all the data due to the Ethernet overhead, and it can be evaluated as:

$$T_{eth} = \frac{(D_{ethernet} + D_{\mathtt{IFG}})}{b}, \tag{16}$$

where $D_{ethernet}$ is the sum of the Ethernet header and the trailer size, i.e., $26byte$ made up of $8byte$ preamble, $6byte$ Destination Mac Address, $6byte$ Source Mac Address, $2byte$ EtherType and $4byte$ CRC (see Figure 4), whereas $D_{\mathtt{IFG}}$ is the $12byte$ InterFrame Gap size. Furthermore, T_{ecat_ov} is the overhead due to ECAT, which can be estimated as:

$$T_{ecat\_ov} = \frac{n_{fr} \times D_{ecat\_hd} + n(D_{ecat\_tgram\_hd} + D_{ecat\_wc})}{b}. \tag{17}$$

In Equation 12 we must still evaluate the term $T_{\mathtt{pad}}$, which represents the time necessary to send the possible padding bytes required to have an Ethernet frame with a minimal standard length:

$$\begin{aligned} D_{ECAT} &= D_{ecat\_hd} + (D_{ecat\_tgram\_hd} + D_{ecat\_wc} \\ &+ D_{data}) \times (n \mod max_{tgram}), \end{aligned} \tag{18}$$

and therefore:

$$T_{\mathtt{pad}} = \frac{D_{eth\_min\_pl} - D_{ECAT}}{b}, \tag{19}$$

where $D_{eth\_min\_pl}$ is the minimum size of the Ethernet payload, i.e., $46byte$ from specifications. The Equation 19 is evaluated only if two conditions hold, namely $D_{ECAT} < D_{eth\_min\_pl}$ and $n \mod max_{tgram} \neq 0$, otherwise $T_{\mathtt{pad}}$ is equal to 0.

III. Numerical Analysis and Discussion

This Section reports about a comparative analysis based on the cycle time indicator. Tables I, II and III show computed cycle times in μs for systems based on Ethernet *Powerlink* and *EtherCAT* without and with redundancy. The number of slaves as well as the payload in bytes have been varied in the ranges $10 \div 200$ and $2 \div 256$. However, only results for a number of slaves greater than 130 are shown.

TABLE I
Ethernet Powerlink Cycle Time [μs]

	Bytes			
Nodes	**2**	**32**	**128**	**256**
130	6967.65	6967.65	7924.45	9255.65
140	7923.21	7923.21	8953.61	10387.21
150	8938.88	8938.88	10042.88	11578.88
160	10014.64	10014.64	*11192.24*	*12830.64*
170	11150.51	11150.51	*12401.71*	*14142.51*
180	12346.47	12346.47	*13671.27*	*15514.47*
190	13602.54	13602.54	*15000.94*	*16946.54*
200	14918.70	14918.70	*16390.70*	*18438.70*

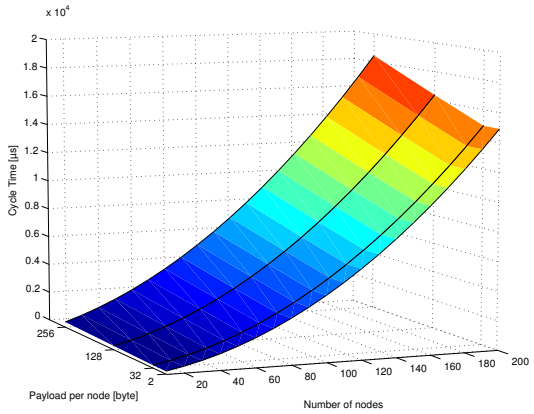

Fig. 9. Ethernet Powerlink Cycle Time.

Rows representing microcontrollers range from 160 to 200. Since each data packet is in the order of $192byte$, EPL cycle times are comprised between $11.1ms$ and $12.8ms$, which is definitely not suitable to implement tactile based reflexive behaviours. The time spent to manage the communication at the software level is not considered, and therefore real cycle time values are underestimated. Since the huge number of hubs in sequence introduces a delay affecting the cycle time, this strongly limits the actual number of nodes that can be connected to guarantee tactile-based control loops. Figure 9 shows the EPL cycle time trend, which is almost *exponential* in the number of nodes.

TABLE II
EtherCAT Cycle Time - Non-redundant configuration [μs]

	Bytes			
Nodes	**2**	**32**	**128**	**256**
130	327.63	646.03	1673.23	3046.03
140	352.34	697.94	1801.94	3280.34
150	377.05	746.65	1930.65	3514.65
160	401.76	795.36	*2059.36*	*3748.96*
170	426.47	844.07	*2188.07*	*3983.27*
180	451.18	895.98	*2316.78*	*4217.58*
190	475.89	944.69	*2445.49*	*4451.89*
200	500.60	993.40	*2574.20*	*4686.20*

When considering ECAT, both the non-redundant (see Table II) and the redundant (see Table III) configurations behave better that EPL. Focusing on at least 160 microcontrollers and assuming $192byte$ data packets, realistic cycle times are in the order of $1.9 \div 3.0ms$, which are time

TABLE III

ETHERCAT CYCLE TIME - REDUNDANT CONFIGURATION [μs]

	Bytes			
Nodes	**2**	**32**	**128**	**256**
130	239.82	558.22	1585.42	2958.22
140	257.77	603.37	1707.37	3185.77
150	275.73	645.33	1829.33	3413.33
160	293.68	687.28	*1951.28*	*3640.88*
170	311.64	729.24	*2073.24*	*3868.44*
180	329.59	774.39	*2195.19*	*4095.99*
190	347.55	816.35	*2317.15*	*4323.55*
200	365.50	858.30	*2439.10*	*4551.10*

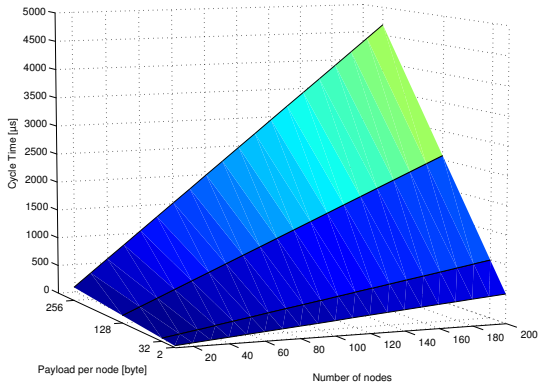

Fig. 10. EtherCAT Cycle Time - Non-redundant configuration.

responses compatible with reactive robot behaviours. The huge number of required hubs in slave nodes does not affect cycle times, since each *ESC* chip reads and writes on the fly. Figures 10 and 11 show the ECAT cycle time *almost linear* trend of the corresponding Tables.

IV. CONCLUSIONS

This paper introduces a comparative assessment of Real-Time Ethernet protocols, focusing on Ethernet *Powerlink* and *EtherCAT*. The two approaches have been analytically and numerically compared with respect to a well-defined scenario, namely the infrastructural design of a whole-body tactile sensing architecture for humanoid robots. Both the approaches represent viable solutions: *Powerlink* is *slower* although the pure *software* implementation makes it suitable to update existing architectures; *EtherCAT* is extremely fast and allows an increased number of nodes to be used maintaining a bounded cycle time: the required electronics can be a drawback in existing humanoid platforms.

REFERENCES

[1] B. Bauml, G. Hirzinger. When Hard Real-Time Matters: Software for Complex Mechatronic Systems. In *Robotics and Autonomous Systems*, vol. **56**(1):5-13.

[2] G. Cannata, R. Dahiya, M. Maggiali, F. Mastrogiovanni, G. Metta, M. Valle. Modular Skin for Humanoid Robot Systems. In *Proc. of the Fourth Int.l Conf. on Cognitive Systems (CogSys 2010)*, Zurich, Switzerland, January 2010.

[3] G. Cena, L. Seno, A. Valenzano, S. Vitturi. Performance Analysis of Ethernet Powerlink Networks for Distributed Control and Automation Systems. In *Computer Standards and Interfaces*, vol. **31**(3):566-572, 2009.

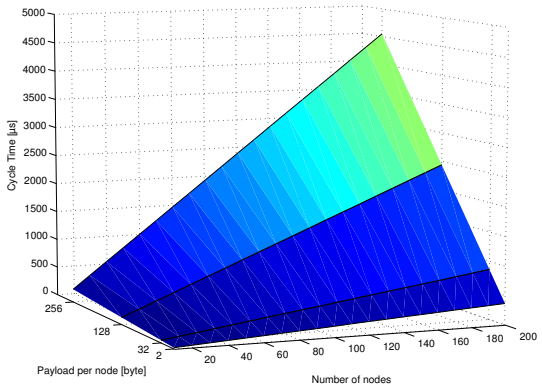

Fig. 11. EtherCAT Cycle Time - Redundant configuration.

[4] Y. Cha, B. You. Design and Implementation of Mobile Humanoid Robot Control System based on Dual Network. In *Proc. of the 2009 IEEE Int.l Conf. on Robotics and Biomimetics (RoBio'09)*, Guilin, China, 2009.

[5] M. Felser. Real-Time Ethernet for Automation Applications. In R. Zurawski (Ed.), *Embedded Systems Handbook*. CRC Press, 2009.

[6] M. Fujita, K. Kageyama. An Open Architecture for Robot Entertainment. In *Proc. of the First Int.l Conf. on Autonomous Agents (AA'97)*, Marina del Rey, CA, USA, 1997.

[7] J.A. Hunt. Ethernet Cuts Fieldbus Costs in Industrial Automation. In *Assembly Automation*, vol. **28**(1):18-26, 2008.

[8] M. Konyev, F. Palis, Y. Zavgorodniy, A. Melnikov, A. Rudskiy, A. Telesh, U. Schmucker, V. Rusin. Walking Robot Anton: Design, Simulation, Experiments. In *Proc. of the 11th Int.l Conf. on Climbing and Walking Robots (CLAWAR)*, Coimbra, Portugal, 2008.

[9] Y. Kuroki, T. Fukushima, K. Nagasaka, T. Moridaira, T. Doi, J. Yamaguchi. A Small Biped Entertainment Robot Exploring Human-Robot Interactive Applications. In *Proc. of the 2003 IEEE Int.l Workshop on Robot and Human Interactive Communication (ROMAN'03)*, Millibrae, CA, USA, 2003.

[10] P. Neumann. Communication in Industrial Automation What is Going On? In *Control Engineering Practice*, vol. **15**(11):1332-1347, 2007.

[11] G. Prytz. A Performance Analysis of EtherCAT and ProfiNET IRT. In *Proc. of the 2008 IEEE Int.l Conf. on Emerging Technologies and Factory Automation (ETFA'08)*, Hamburg, Germany, September 2008.

[12] M. Sarker, C. Kim, S. Baek, B. You. An IEEE-1394 Based Real-time Robot Control System for Efficient Controlling of Humanoids. In *Proc. of the 2006 IEEE/RSJ Int.l Conf. on Intelligent Robots and Systems (IROS'06)*, Beijing, China, 2006.

[13] L. Seno, C. Zunino. A Simulation Approach to a Real-Time Ethernet Protocol: EtherCAT. In *Proc. of the 2008 IEEE Int.l Conf. on Emerging Technologies and Factory Automation (ETFA'08)*, Hamburg, Germany, September 2008.

[14] L. Seno, S. Vitturi, C. Zunino. Real Time Ethernet Networks Evaluation using Performance Indicators. In *Proc. of the 2009 IEEE Int.l Conf. on Emerging Technologies and Factory Automation (ETFA'09)*, Palma de Mallorca, Spain, September 2009.

[15] J. Shan, F. Nagashima. Neural Locomotion Controller Design and Implementation for Humanoid Robot HOAP-1. In *Proc. of the 20th Annual Conf. of the Robotics Society of Japan*, Osaka, Japan, 2002.

[16] Y. Sung, B. Hwang, J. Kong, B. Lee, J. Kim. Messages Scheduling for a Humanoid Robot in the CAN Bus. In *Proc. of the 2004 IEEE Annual Conf. of the Industrial Electronics Society (IECON'04)*, Busan, Korea, 2004.

[17] Z. Yu, Q. Huang, L. Li, Q. Shi, X. Chen, K. Li. Distributed Control System for a Humanoid Robot. In *Proc. of the 2007 IEEE Conf. on Mechatronics and Automation (ICMA'07)*, Harbin, China, 2007.

[18] C.Y. Yu, C.H. Lin, Y.H. Yang. Human Body Surface Area Database and Estimation Formula. In *Burns*, May 2009.

[19] H. Zhong, Z. Wu, B. Li, C. Bu, Y. Li. Design and Implementation of Humanoid Robot Controller based on CAN. In *Proc. of the 2004 IEEE Int.l Conf. on Robotics and Biomimetics (RoBio'04)*, Shenyang, China, 2004.