Proceedings of the Federated Conference on Computer Science and Information Systems pp. 751–755
DOI: 10.15439/2016F481
ACSIS, Vol. 8. ISSN 2300-5963

Comparison of Network Architectures for a Telemetry System in the Solar Car Project

Cody R. Barnes, Ethan G. Toney, Jerzy W. Jaromczyk
University of Kentucky
Department of Computer Science
Lexington, KY 40506, USA
Email: cody.barnes@uky.edu, egto222@uky.edu, jurek@cs.uky.edu

***Abstract*—A solar car is an electric vehicle that runs entirely on solar energy. Designing, building and racing solar cars has been a longstanding worldwide challenge for engineering and computer science students, with the overarching goal being to design devices that use sustainable energy sources. This article describes our experience and educational outcomes (*the modeling and design of computer-based systems in a way that demonstrates comprehension of the trade-offs involved in design choice*) attained while designing the network architecture for a solar car project.**

The computer science members of the University of Kentucky Solar Car Strategy Team are tasked to reliably collect and analyze car data in real-time, both to assist in the car development process, and then to provide important sensor readings to the driver during racing. The challenges in designing the architecture and protocols for the computer system that supports the solar car are to ensure that: (1) energy consumption is minimal, (2) data collection is reliable, (3) the network system is secure, and (4) the implementation of the system is not overly complex.

Our computer system supporting the telemetry tasks uses three micro-controllers to collect and send data over serial communications to a master micro-controller (the Raspberry Pi), that parses and stores data in an on-board database.

We compare three protocols: a simple USB-based protocol, and two protocols used in traditional non-solar cars: CAN and Ethernet. We analyze (1) their energy consumption over a period of time, (2) their reliability (by performing stress tests such as disconnecting devices and driving over bumpy terrain), (3) their security (by attempting to compromise the system by remotely sending data over communication lines), and (4) their complexity in terms of time and effort for implementation and development.

I. Introduction

SOLAR Car racing (see Figure 1), the competitive racing of fully electric vehicles using solar energy, has existed since 1985. Universities and businesses around the globe participate with different goals in mind. Universities typically participate in order to improve engineering and technical knowledge and skills of the students, while businesses typically participate to develop renewable energy technologies. Traditionally, a solar car requires a collaboration of electrical, mechanical, and computer engineers (see [6]). However, with data storage becoming cheaper, and micro-controllers becoming more abundant and inexpensive (such as the Arduino and Raspberry Pi) these make programming embedded systems just as easy as programming an application for a typical computer. In other words there is a growing need for computer science students to be a part of the team. Solar car teams can easily capture, store, and analyze data in order to improve the overall performance of the car.

In the context of power consumption, we analyzed a telemetry system for the solar car project. An automated communications process collected and transmitted car performance data to receiving equipment and personnel for monitoring, processing, testing and decision making. When choosing the network architecture to support telemetry to achieve the aforementioned capturing, storing, and analysis of data, we must determine trade-offs that are critical to the car's functionality. It is also important to carefully consider and design the underlying software architecture that is the driving force behind the network architecture.

Minimizing the power consumption, and comparing various solutions and their trade-offs, are some of the contemporary challenges, addressed in many projects, including the past workshops of this conference. See for example, [1], [2], [4], and [5].

This paper reports on experimental results of a student team, for whom the Solar Car project serves as an attractive way of learning engineering topics. Clearly, tools, instruments, and methods are unequal to sophisticated, advanced, and likely expensive telemetry systems developed for professional motor racing, such as in Formula One. However, the experience with a real-life project of designing the network architecture for a solar car project, contributes to the important computer science student outcome: *the modeling and design of computer-based systems in a way that demonstrates comprehension of the trade-offs involved in design choice*. See also [3] for a discussion on using real-life projects for "developing skills needed for the proper formulation of system visions and requirements specifications."

The subsequent sections describe and compare three different network solutions, the architecture of our telemetry system, and finally, provide our conclusions.

II. Network Comparison

We now discuss the trade-offs of using USB [7], CAN [8], and Ethernet while trying to meet these four challenges: low power-consumption, reliability, security, and simplicity.

A. Low Power-Consumption

Low power-consumption is by far one of the most critical requirements in the development of a solar car. Naturally, the

Fig. 1. Solar Car – team UK

architectural design of the network was not exempt. Since we are racing the car, keeping the power consumption down allows us to allocate more power towards other critical parts of the car. Saving this power for the motor has the potential to result in more mileage out of the total amount of power. We hypothesized that USB and CAN would require about the same amount of power and that Ethernet would require much more than the previous two. Consequently, we chose to compare USB and CAN first and then talk about Ethernet afterwards.

1) USB versus CAN: USB and CAN are known to be very low power networks as long as they are not being used to power a device. To accurately compare these two approaches, we chose to measure the amount of power required to power a CAN chip and a USB chip with no data being sent over them. From experiments, we found that the CAN chip required 11% less power than the USB chip. Although 11% may appear big, in practice the difference is negligible due to the measured values being low.

2) Ethernet: Compared to USB and CAN, Ethernet is known to use significantly more power. This is because of larger hardware requirements to allow for an Ethernet network. Not only would one need to put a router or switch on the car, one would also have to add more hardware on the actual micro-controllers to be able to communicate with Ethernet.

B. Reliability

The reliability of each of the protocols is important because it ultimately determines whether or not data are received. When discussing the reliability of the three network architectures, we focus on the hardware aspects.

1) USB: USB provides data integrity within the cables, meaning that there is a high degree of confidence that the data sent over the transmitting end will make it to the receiving end. Any unrecoverable errors will likely be noticed and reported to the appropriate end of the cable. Data integrity is provided through USB's self-recovery system. This approach guarantees that a message will be resent at least three times before reporting an error to the client software, and will throw time-outs for lost or invalid packets.

Unrelated to the actual data integrity within the cable, but still considered in our evaluation, is the observation that USB has a high likelihood of becoming mechanically disconnected from its transmitting or receiving end due to jostling that occurs while the car is in motion.

2) CAN: CAN is similar to USB in that it also provides data integrity. The CAN protocol defines no less than five different ways to detect error:

- Cyclic Redundancy Check (CRC) acts similar to a checksum by doing polynomial division on the bits and comparing the end result to the 15 bit CRC field located within the packet.
- Acknowledgment (ACK) Check - The node that transmitted a message has essentially sent a recessive level and would expect to receive a dominant ACK message. If it does not, then it acts as if the previously sent message was lost and responds accordingly.
- Form Error Check - If there is a dominant bit in the CRC field delimiter, ACK field delimiter, or EOF (end of frame) then the message is re-sent because, per protocol definitions, there must not be a dominant bit in these fields.
- Bit Stuffing - If six consecutive bits with the same polarity occur between the SOF (start of frame), then the CRC field throws an error. The EOF field should be the only field with six consecutive bits of the same polarity.
- Bit error occurs when the transmitter reads a signal that is opposite of what it sent except during arbitration and in the ACK field. Arbitration is a method that provides a bitwise losslessness if all of the nodes on the CAN network are synchronized to allow for every bit on the CAN network to be sampled at the same time.

In contrast to USB, the few devices on the car that utilize CAN have never had any issues with disconnectivity due to the jostling of wires.

3) Ethernet: Ethernet in itself is not reliable. It does not support retransmission, it does not provide acknowledgment of successful frame delivery, and if a frame were to become corrupt it simply drops it without letting the transmitting or receiving end know. Due to this characteristic, we consider the use of Ethernet with TCP, Transmission Control Protocol. TCP is a protocol that is often overlaid on top of Ethernet. TCP supports detection of duplicate data, retransmission, and sequencing. Because Ethernet is usually used to send packets over long distances, and because multiple Ethernet switches are used, packets can be duplicated. TCP detects duplicate packets and drops the unnecessary ones. In conjunction with this duplication, packets can get out of order. Therefore TCP

supports sequencing - placing packets in the order that they were sent. Additionally, if a packet gets lost or corrupted, TCP provides retransmission of the affected packet.

C. Security

The UK solar car was designed just for races, and not intended for mass production. Nevertheless, network security is becoming a general concern and should always be accounted for. Especially considering the recent cyber-security incidents related to private cars being hacked ([10]). Even though our specific network application represents a minimal security threat, security was not ignored.

1) USB: From a security standpoint USB is actually a very good option. It only allows for peer-to-peer communication, so there is essentially no network to abuse. However, due to the nature of peer-to-peer, there is no form of authentication. Simply, each computer does not have a way of being completely sure who it is talking to. Given this situation, the only realistic way to attack a USB-based system would be to plug in directly to the device you wanted to lie to.

2) CAN: A CAN network is riddled with security problems stemming from the underlying protocol. Much like the IP protocol, it relies on the sender to accurately choose a message ID. From this message ID you can tell what kind of data are being sent, as well as what device is sending it. This is done by a device choosing a range of IDs, and making sure that every message it sends is inside of this range. Given this type of protocol, it is trivial to impersonate another device on the network. A person could just send a message onto the network with an ID that is not their own, and the receiving device would not know the difference. As a proof of concept, we simulated this vulnaribility and were able to get the Raspberry Pi on our CAN network to tell the motor to accelerate even though the pedal was not being pressed. It is understandable why the developers of this network originally left this vulnearibility, because CAN was always intended to be an internal network. Getting on to the network to abuse its security faults is not easy. This vulnerability is also why there has been so much news about cars being hacked lately; see [10]. Since CAN is the industry standard, it is not too difficult for a hacker to use a laptop with a CAN cord and plug into the network manually to break in. In some cases, where the car has an Internet access, there may be ways to get from the Internet network to the CAN network and remotely hack a car.

3) Ethernet: Ethernet has almost the same set of vulnerabilities as CAN. However, it is substantially easier to abuse a CAN network than it is to abuse the security vulnerabilities in Ethernet. Numerous protocols built on top of Ethernet reduce its vulnerability. Clearly, it is still possible for hackers to abuse Ethernet network by misrepresenting the identity and impersonating devices on the network.

D. Simplicity

Simplicity, a universal value and software development practice, is commonly used by student teams. These teams operate on schedules dictated by the academic calendar. In addition, once they graduate, the code base is picked up by future Computer Science students to continue assisting the Solar Car project; see [11] for the repository with our code.

1) USB: On the software side, USB is relatively simple. The majority of languages that would be commonly used in these type of applications (Python, Java, C, C++, etc.) support serial communications. One must simply identify the port to communicate with and open a serial connection with said port.

On the hardware side, USB actually is not native to the majority of the solar car peripherals except for the micro-controllers themselves. As a result, USB use can cause problems for the developers with the devices on the car that communicate through CAN. To alleviate this difficulty, the team utilizes CAN-to-USB converters in order to receive data.

2) CAN: Even though, communicating with CAN through software is not as common, it can be done in a relatively standard way. In fact, communication support is implemented in `Python`, the main language used on the Raspberry Pi. However, CAN is more complex to use rather than USB or Ethernet. In fact, even the documentation about the implementation through `python-can` library suggests that working with `python-can` is much more complex than the implementation of USB through the `pyserial` library. Thus, in our solution, we actually use a CAN to USB converter between the CAN network and the Raspberry Pi, to keep the software implementation as simple as possible.

The hardware side for CAN is the upside, however, with CAN being the industry standard for regular cars due to its reliability, upgradeability, performance, and cost. As the benefit of standardization, one can expect that the majority of the car peripherals communicate over CAN. Among them is the Motor Controller that drives the rear third wheel of the solar car.

3) Ethernet: Ethernet is very similar to USB when it comes to software aspects. Almost all languages have some kind of built in library that handles communication over sockets, which are the Ethernet equivalent to ports in USB. This gives Ethernet a distinct advantage over CAN when it comes to software implementations.

For the hardware side, Ethernet is in the same category as USB. Ethernet is not common among any of the car peripherals besides the micro-controllers. In effect, Ethernet switches and CAN to Ethernet converters have to be used.

Table I summarizes the above evaluation.

TABLE I
SUMMARY OF THE COMPARISON FOR THE SOLAR CAR PROJECT

	Power usage	Reliability	Security	Simplicity
USB	low	good	good	poor hardware support
CAN	low	good	poor	nontrivial software support
Ethernet	high	good (with TCP)	poor	poor hardware support

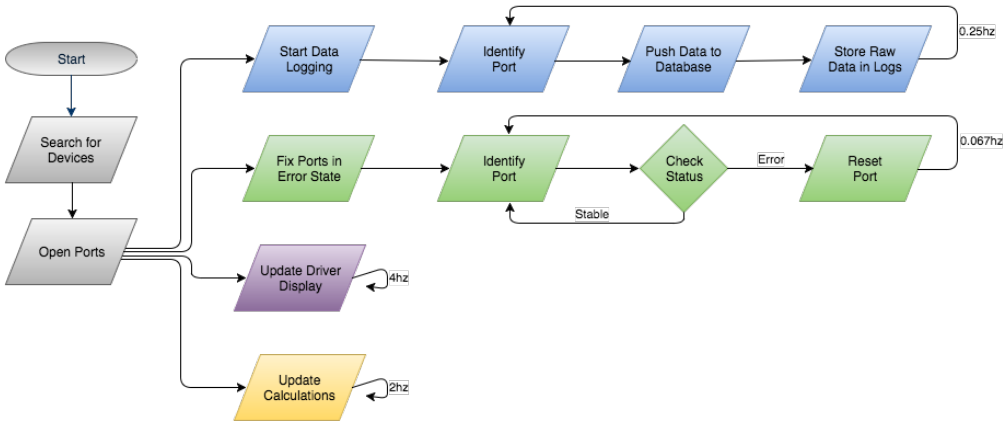

Fig. 2. High-level overview of the software architecture

III. Software Architecture

There are multiple choices for the network architecture and the final decision on whether or not CAN, USB, or Ethernet would be better choice for us is greatly influenced by the current software architecture.

Depicted in Figure 2 is a high-level overview of our software architecture. When the program is started on boot, it immediately opens the ports where data are expected, and then connects to the corresponding devices for those ports.

Rules in UDEV - a device manager for the Linux kernel - are used so that each device is assigned a unique identifier based off of the product id on the USB chip.

The flow of control is as follows. The program starts off by initially searching for any connected devices and opening up a connection to them. After this point four threads are started up and read from a global state containing of a list with each device. The threads run concurrently, and access the global state, but do not communicate directly with one another. The first thread, *the logging thread*, starts off by ensuring that the device has been identified. Then it logs the device buffer to the database on the car, and stores the raw data logs for the *post mortem* analysis.

The second thread, *error correcting thread*, iterates through all of the devices and identifies their kind. Then, this thread looks at the statistics of each device to determine whether the device is in an error state.

The third thread, *update drive display*, sends the vehicle speed and battery current measurements to the attached Arduino. The Arduino then takes these values and displays them on two seven segment displays.

The last thread, *update calculations*, does the calculations that would otherwise require too much bandwidth to send the sufficient amount of information for the other end of the telemetry system to calculate.

The responsibilities of the threads described above, and presented in Figure 2, are:

1) Thread Data Logging: This thread loops through every device (port), logs any data that have been saved up in the device buffer, pushes the data to the SQL database and writes the data to raw text files as a backup.
2) Thread - Port Fixing: This thread also loops through every device, but instead of logging data it checking to ensure that the device is still connected and that we are receiving data. If the device is in an error state then this thread will attempt to fix it until it attains a stable state again. This thread is also the only thread with the capabilities to significantly change the global state, so its actions are heavily synchronized with the other threads.
3) Thread - Updating Driver Display: This thread updates the driver display with the current car speed and voltage usage.
4) Thread - Update Calculations: This thread updates any calculations that are being sent over the radio telemetry system in the trail (chase) car. As an alternative solution, these calculations could be moved to the Java application on the receiving end to lessen the amount of work that the program on the Raspberry Pi has to do. The tradeoff would be in using more bandwidth to send the data.

The diagram in Figure 3 shows the general layout for the current USB based telemetry system for our solar car. There are multiple devices that are plugged into a powered hub: two battery boxes, the motor CAN network, an Arduino, and a telemetry box. Also you may note that the Arduino controls many devices: two seven-segment displays (used to show speed and current), gyro, accelerometer, and GPS. These extra devices data are eventually sent on to the Raspberry PI for processing. There is also a small CAN network between the drivers control box and motor controller that must be there regardless of the choice of architecture. Regarding the powered hub, on the other end there is a Raspberry PI micro-computer that uses this powered hub to create separate connections to each device and manage each of them individually. The micro-computer then reads and interprets data from each device, and then sends them to the telemetry box; a matching telemetry box in the chase car receives these data. During the interpretation of data on the Raspberry

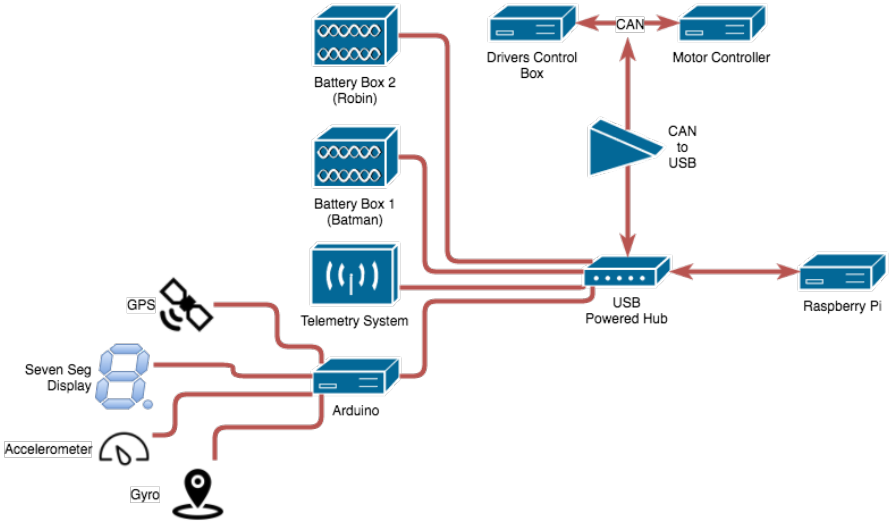

Fig. 3. Architecture of our telemetry system

Pi that data are permanently stored in an on-board MySQL database.

IV. Conclusion

By analyzing various network solutions, we have determined trade-offs in Solar Car network architectures that try to meet the four challenges: low power consumption, reliability, security, and simplicity. In comparing these solutions, we have found that USB and Ethernet are very respectable options. USB provides a quick starting point. For example, if a project involves a micro-controller then it is very likely that the micro-controller contains, at the least, a USB port. Furthermore, communications over a serial connection, which is what USB is, are very easy to implement. On the other hand, Ethernet, being a class D network, can handle very high data rates (up to 100 mb/s). If the Solar car needed to transfer this much data then Ethernet high energy usage could be countered by the trade-off for high data rates and easy implementation. Even with USB and Ethernet being respectable options, CAN still has been found to be the best overall option. The hardware is easy to implement, cheap, and upgradeable. It uses a low amount of energy, which is highly critical to our application, and although it is not as common to implement in software, there exists workarounds. Working on a solar car design, along with implementing and testing solutions, has provided us with incomparable learning outcomes in networking, security and teamwork, and have allowed all the participating students to clearly see design trade-offs, even when some of them are subtle.

References

[1] R. Banach, P. Van Schaik, and E. Verhulst. Simulation and formal modelling of yaw control in a drive-by-wire application. In M. Ganzha, L. Maciaszek, and M. Paprzycki, editors, *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems*, volume 5 of *Annals of Computer Science and Information Systems*, pages 731–742. IEEE, 2015.

[2] M. Eshaftri, A. Al-Dubai, I. Romdhani, and M. Bani Yassein. A new energy efficient cluster based protocol for wireless sensor networks. In M. Ganzha, L. Maciaszek, and M. Paprzycki, editors, *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems*, volume 5 of *Annals of Computer Science and Information Systems*, pages 1209–1214. IEEE, 2015.

[3] J. Král and M. Žemlička. Experience with real-life students' projects. In M. Paprzycki M. Ganzha, L. Maciaszek, editor, *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, volume 2 of *Annals of Computer Science and Information Systems*, pages pages 827–833. IEEE, 2014.

[4] C. Panait and D. Dragomir. Measuring the performance and energy consumption of AES in wireless sensor networks. In M. Ganzha, L. Maciaszek, and M. Paprzycki, editors, *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems*, volume 5 of *Annals of Computer Science and Information Systems*, pages 1261–1266. IEEE, 2015.

[5] T. Szydlo, P. Nawrocki, R. Brzoza-Woch, and K. Zieliński. Power aware MOM for telemetry-oriented applications using gprs-enabled embedded devices - levee monitoring use case. In M. Paprzycki M. Ganzha, L. Maciaszek, editor, *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, volume 2 of *Annals of Computer Science and Information Systems*, pages pages 1059–1064. IEEE, 2014.

[6] R. Mangu, K. Prayaga, B. Nadimpally and S. Nicaise, Design, Development and Optimization of Highly Efficient Solar Cars: Gato del Sol I-IV, in *2010 IEEE Green Technologies Conference*, Grapevine, TX, 2010, pp. 1-6. doi: 10.1109/GREEN.2010.5453800, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5453800&isnumber=5453775

[7] USB Specifications http://www.usb.org/developers/docs/. Online; accessed 9-May-2016

[8] CAN Specifications http://www.usb.org/developers/docs/. Online; accessed 9-May-2016.

[9] M. Faezipour, N. Faezipour, M. Adnan, S. Adnan, S. Addepall., Progress and challenges in intelligent vehicle area networks, in *Communications of the ACM*, 55 (2), 2012, pp. 90–100,

[10] J. Vanian. Hacking Cars Is Easy, http://fortune.com/2016/01/26/security-experts-hack-cars/, Online: accessed 10-July-2016.

[11] E. Toney. https://github.com/KentuckySolarCar/RaspberryPi