Abstract

This white paper describes the fundamental design goals of the Halo/S data acquisition and control platform, as well as providing a high-level overview of how Halo/S works. This document should also provide potential integrators and platform partners with high level details regarding the architecture, and help explain the many advantages of using Halo/S over competing M2M/aquisition/control platforms.

Overview

Problem statement

Competing data acquisition and control systems have been building on top of legacy platforms for decades, and many "modern" concerns such as wireless and security have never been properly addressed at a low level. Additionally, communication APIs and hardware power requirements have been constantly growing to handle additional complexities, resulting in more expensive systems which are increasingly difficult to design and install.

There are also many new lightweight machine-to-machine (M2M) platforms which have cropped up as well. These new platforms are generally built around a cloud-based server which is completely controlled by the manufacturer and offer a very limited set of commands.

These competing systems generally share a core set of problems:

a) Installed cost: As hardware component costs and software development costs decline, the installed cost of competing systems has only slightly fallen over the past 10 years. This is primarily due to the substantial manpower required to design and install these complex competing systems. Additionally, the manufacturing "support costs" of developing and supporting a large lineup of custom hardware and software comprises a significant portion of the list price for competing products. Compting wireless platforms are mostly unusable in commercial/industrial settings, leaving most installations with no choice but to use a "wired" system and factor in the substantial installation costs related with wiring.

b) Integration complexity: For full-features data acquisition and control alternatives, vendor lock-in is the primary factor in preventing third parties from integrating with the platform. Manufacturers generally only allow integration using aging open protocols such as MODBUS or BACnet, severely limiting the amount of integration possible.

c) Upgradeability/expandability: All competing platforms have adopted a "controller-centric" approach, with all core logic and processing handled by isolated controllers (e.g. PLCs, cloud-based servers, etc). This dependence on centralized controllers creates a severe problem when multiple networks must be combined. Each of these controller-centric networks must then be tied together with yet another layer of communication complexity.

Additionally, much of the core functionality of these competing systems is contained in the actual controller hardware. When improved functionality or software upgrades are desired, the controllers or software systems must be completely replaced with more advanced systems, usually at significant cost to the user.

Design goals

The primary design goal of the Halo/S platform was an order of magnitude (10x) deduction in the manpower required to design and install a typical data acquisition or control system. Combined with a 50% reduction in hardware costs, the total installed cost of a Halo/S system was designed to be 1/3rd that of competing systems in 80% of all applications.

In addition to the substantial cost savings target, the other primary design goal of the Halo/S platform was to create a platform which could be integrated by third parties at practically every layer of the product "stack". By encompassing all of the hardware and software layers into a single platform, third parties can easily choose which layers of the stack they wish to implement themselves, while also offering a complete off-the-shelf solution for companies which prefer not to develop their own system components.

The Halo/S core v1.0 platform has achieved these design goals through a variety of innovative measures. Each of the major architectural design areas of Halo/S are explained in the following section.

Technology description

The architecture of the Halo/S core v1.0 platform is quite different from competing platforms. In order to fundamentally address the issues with other platforms, Halo/S has been designed from the ground up with a revolutionary new approach for M2M communication and control. A fundamental shift in architecture was needed to break the cycle of ever-increasing cost and complexity from competing alternatives in the data acquisition and controls space.

Decentralized

Halo/S has been designed to run on an ultra-low cost ($0.50) 8-bit 8051 MCU, allowing every sensor and control to be made "smart". This allows sensors, controls, and interfaces to execute their own software and communicate directly with each other, eliminating the need for centralized controllers. This provides unmatched upgradability/scalability, since any arbitrary sensor or control can be added to a Halo/S network without any concern for compatibility, controller input limits, or software limitations. Total hardware cost is also significantly reduced through the elimination of controllers and intermediary devices.

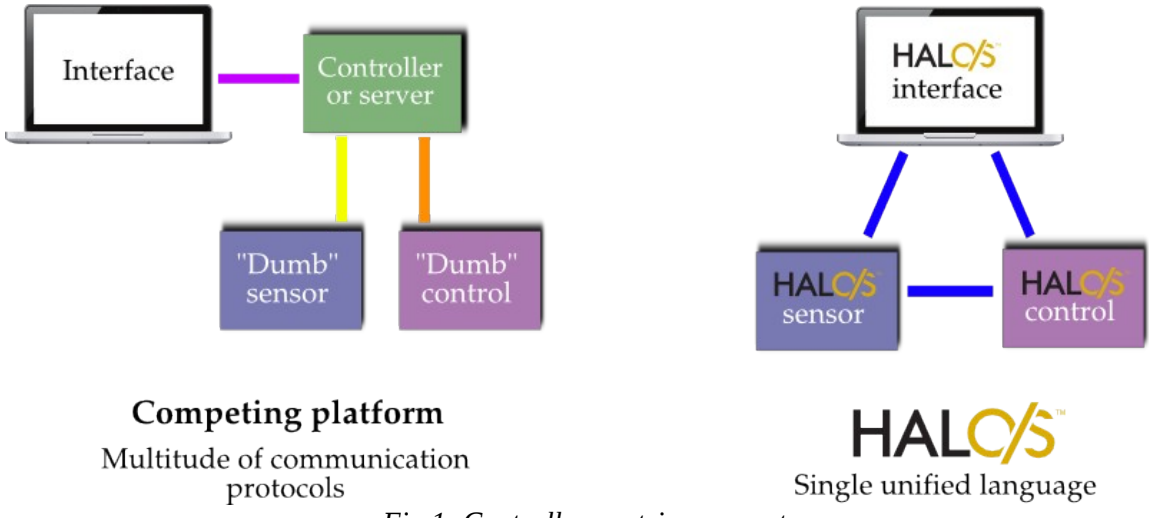

Fig 1: Controller-centric vs peer-to-peer

Communication options

In order to accommodate a wide variety of differing systems, a large number of communication "network transport" options have developed to facilitate information between Halo/S devices.

The TCP, HTTP, and WebSocket network options provide the simplest solution for high-level Halo/S communication, while the serial and wireless options provide the best method for communicating with low-cost hardware. All of these transport options can be completely secured via Halo/S security wrapping (see Security section) for complete end-to-end encryption and authentication.

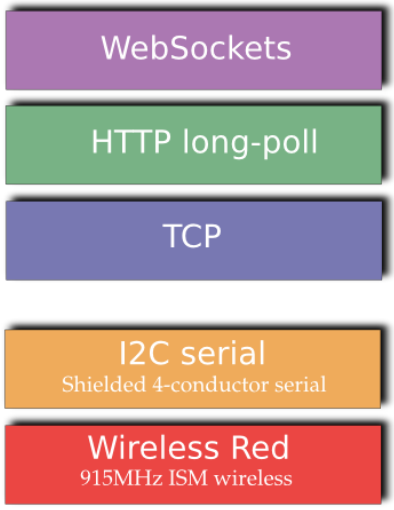

Fig 2: Halo/S network transports

Wireless

Of the 5 current transport options, special attention has been devoted to the "Wireless Red" option. This transport has been specifically developed to provide a next-generation, low-data-rate wireless protocol which vastly improves performance verses existing state-of-the-art wireless protocols.

The Wireless Red transport utilizes the 915MHz ISM band, which can be used in ITU Region 2 (Americas), subject to certification in each country (US and Canada certification is currently obtained). Additional wireless band options will be added in the future for use in Region 1 and 3 countries.

Time-synched

All devices on the network are synchronized to a master clock signal, linking all device clocks to within 30 microseconds. This time synchronization allows all devices to automatically adjust for oscillator crystal shift, and utilize a single precision clock for an entire network. This precision clock also allows devices to sleep for the vast majority of their life, waking up just in time to receive the next wireless synchronization beacon and returning to sleep in less than a millisecond.

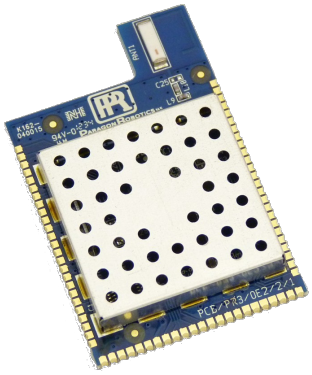
Fig 3: OE2x module

Multiple data-rate

The wireless system also provides vastly improved indoor range versus competitors such as Zigbee (4x indoor range), through the use of a first-of-its-kind multiple datarate protocol which allows devices to automatically drop down to a slower data rate in order to extend range, while allowing nearby devices to use the fastest rate and reduce congestion.

With slower datarates, wireless receivers can receive weaker signals, extending the possible range a device can maintain with a wireless peer. However, slower datarates reduce the amount of data that can be communicated on the network. By utilizing an adaptive multi-datarate protocol, range can be extended for certain devices, while maintaining the highest datarate possible for devices which are close to each other.

Frequency hopping

Frequency hopping (as opposed to spread-spectrum) is used by Wireless Red in order to further improve indoor reception. Much of the signal degredation indoors is caused by multipath interference. Because multipath interference is highly frequency-dependent, by utilizing frequency hopping Halo/S devices can quickly hop to another frequency channel in order to help try and improve the reception.

Next-gen low-cost hardware

Additionally, a next-gen wireless hardware design provides up to +128dBm wireless link budget, compared to a typical +105dBm budget with competitors. An integrated +20dBm transmitter on the target 8051 MCU eliminates the need for additional transceivers or amplifiers, significantly reducing the overall hardware cost.

Software-defined network parameters

In order to provide even more tuning flexibility in particular situations, most of the primary parameters of the Wireless Red protocol are changeable on-the-fly, allowing all devices on the network to change everything from channel-hopping timing to data rate parameters at will. This allows certain parameters like range or security to be further improved by changing to a particular set of tuning parameters.

This software-defined flexibility also significantly improves jamming resistance, as gateways can rapidly change channel hop sequences and then encrypt them such that jammers cannot reliably follow the channel hop schedule.

Software-defined (machine/app trees)

All hardware devices (sensors/interfaces/controls/gateways/etc) have an identical Halo/S control module with identical firmware onboard. Actual functionality is then provided by higher level software which can be configured and remapped on-the-fly and over-the-air. This allows unprecedented software upgradability to any network and allows all future upgrades to be done solely with software, reducing the upgrade cost.

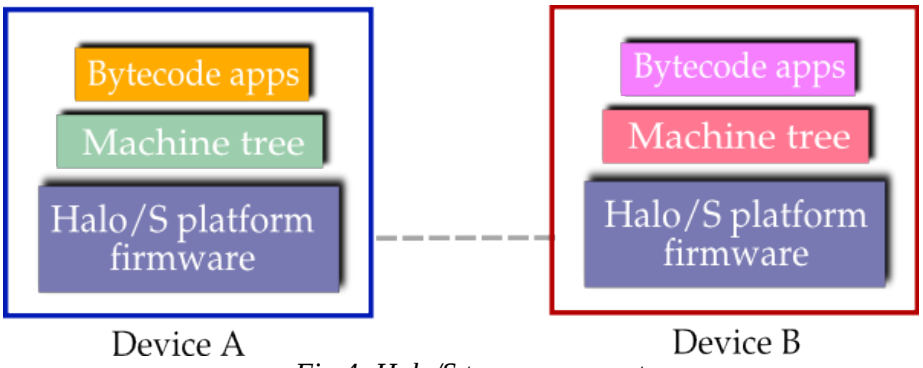

Fig 4: Halo/S tree components

Halo/S unifies data and code into a single "tree". When a device needs to communicate with a peer, the Halo/S interpreter is able to automatically copy a subset of its tree at its current location, and then simply transmit that tree subset to another device. The peer device then overlays this incoming tree subset onto its tree, and is able to automatically proceed without any specialized API decoding. This architecture completely eliminates the need for any specialized communication APIs, creating a system

which is infinitely expandable while exposing 100% of a device's functionality to the network.

Machine layer: On top of the common firmware for the Halo/S interpreter, each device has a high-level "machine layer" mapping of all input and output hardware it contains. This layer allows all hardware functionality to be fully exposed to the Halo/S network. For example, the machine layer for a temperature sensor would include everything from pin mappings for the A/D converter, calibration equations to expose the temperature reading as a SI value, and logging parameters. Control outputs are similarly configured by this machine layer. This layer is typically fixed for each device, and is not usually modified after device production.

Bytecode app layer: On top of the machine layer, the application layer contains the actual high-level control software. This layer provides a canvas for any arbitrary control programs to reside, ranging from simple alarms to extremely complex PLC algorithms. Many applications can reside on each device, and these applications can also seamlessly interact with each other as well as applications on other networked devices. The application layer is frequently changed by end-users, typically via high-level setup software GUIs.

Site-to-site networking

When remote networking or "site-to-site" networking is required, HaloCloud servers can be used to connect multiple Halo/S networks together. HaloCloud is unique in that it provides a completely private and secure connection between the user's networks. HaloCloud capitalizes on the multi-layer encryption ability of Halo/S, allowing users to completely secure the communications passing through HaloCloud with their own private key (in addition to the HaloCloud access encryption).

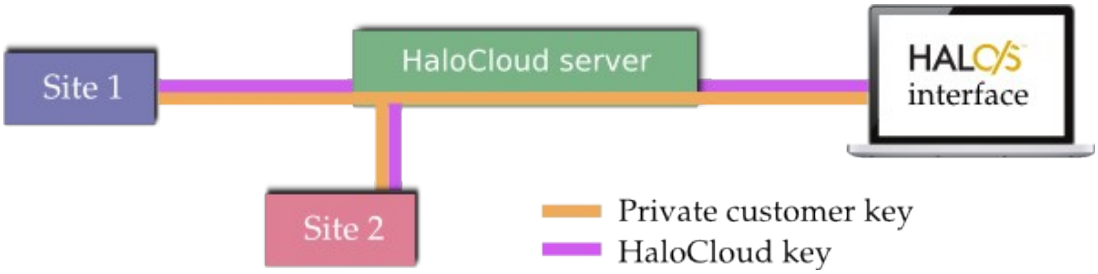

Fig 5: HaloCloud double-encryption

HaloCloud servers are extremely simple, since they are simply routing packets with very little overhead involved. HaloCloud server software and cloud-based services are available to integrators to easily set up private HaloCloud servers.

Security

AES-256 encryption/authentication is used for all Halo/S communications. Multiple keying allows messages to be encrypted at multiple levels, allowing for advanced security designs with multiple access privileges. The unprecedented network flexibility of Halo/S also allows for many different "backhaul" connection options ranging from dedicated LANs, to POTS systems, to secure tunneling across common LAN cabling. Fully supported scripting library and server scripts area available which

can fully access any Halo/S network, allowing for "buffer firewall" systems to relay incoming external requests to Halo/S hardware. This prevents arbitrary network packets from reaching any Halo/S hardware and eliminates that attack vector.

High-level software (HTML5/ Nodejs)

The official software libraries and development tools for Halo/S are written entirely in Javascript. This allows the Halo/S stack to be run on practically any hardware or operating system in modern use today. The end-user typically utilizes this stack on two distinct platforms: HTML5-compatabile browsers, and server-side using node.js.

Node.js: Node.js is a server/scripting platform which runs on top of Google's V8 Javascript engine, providing full server and scripting capability to Javascript applications. Using the same Javascript library packages as the HTML5-based software, complex servers and scripts can be developed which have complete access to a Halo/S network. The HaloCloud server software and others are all built on top of node.js.

HTML5 browsers: The most popular platform for user access to a Halo/S system is through a modern HTML5-based browser (such as Firefox or Chrome). All modern mobile browsers are also compatible, allowing access via any modern hardware or operating system.

As part of the core platform software offerings, the HaloView package provides the fundamental software platform for HTML5 applications and websites to be quickly assembled to interact with Halo/S systems. The software package is divided into several key applications:

haloView: provides a general HTML5 GUI framework for advanced Halo/S websites and applications, including much of the peripheral functions such as persistent storage, multi-window applications, charting, and more.

appDeveloper: a tool used by developers to create specific device machine trees, as well as control apps.

appManager: a user package for installing and configuring apps onto Halo/S device networks.

Additional software is also available through many of our partner companies, providing everything from GUI configuration tools to complex charting GUIs and other tools.

Feature-specific details

Data acquisition

In order to further strengthen the platform in the data acquisition and logging areas, several unique mechanisms have been developed into the Halo/S platform.

Built-in databasing: Several types of indexed databases have been built in to the core Halo/S platform, allowing logged data to be easily stored in a common database format, and allowing any other Halo/S device to read this stored data.

Time-synchronized: Taking advantage of the time-synched nature of the Halo/S platform, all stored data is time-aligned to within 30 us of other sensors and devices on the network, allowing precision data recording among systems of sensors. This also eliminates the need for any post-processing of data from different sensors in order to synchronize the times.

Indexing: To allow rapid retrieval of stored data, the built-in databases support indexing of the stored data so that long-term trends and averages can be quickly retrieved from the database without requiring the download of all data. This allows users to "zoom out" on a data chart and view decades of trend data for a certain sensor almost instantaneously, without requiring a long download of the individual data points.

Massive embedded storage: Embedded drivers for SD flash memory have been built into the OE22 modules, allowing massive storage space for logged data. A 32GB SD card can hold approximately 120 billion data points on a single device, providing ample space for practically every application.

Compress/queue/store: To minimize the communication bandwidth requirements between sensors and storage devices, all sensors provide compress/queue/store (CQS). With CQS, each sensor can be set up to compress its recorded datastream using a linear compression scheme, and then queue it in onboard memory. When this memory fills up, it packages the compressed datastream and sends it to the storage device for permanent storage in a dedicated database. This greatly improves battery life for wireless sensors, as well as allowing hundreds of logging sensors to coexist on the same wireless network.

Users can quickly download the stored data directly from the storage device, however if the most recent data is also required, Halo/s seamlessly downloads the queued data directly from the sensor and combine it with the permanently stored data.

Controls

Control of a Halo/S network can be categorized into two primary types: a) Halo/S bytecode algorithms, and b) high-level "external" control. A combination of these two types can also be used.

Bytecode apps: The decentralized architecture of a Halo/S network provides a number of advantages over controller-centric alternatives. In order to capitalize on these advantages, Halo/S bytecode apps can be run on any device, creating a truly peer-to-peer control scheme. Bytecode apps are developed as general-purpose packages which can be generally distributed to users. The user downloads the app package and configures any options or device requirements for the specific app, then installs it to a Halo/S device network. This entire process is handled by the appManager widget.

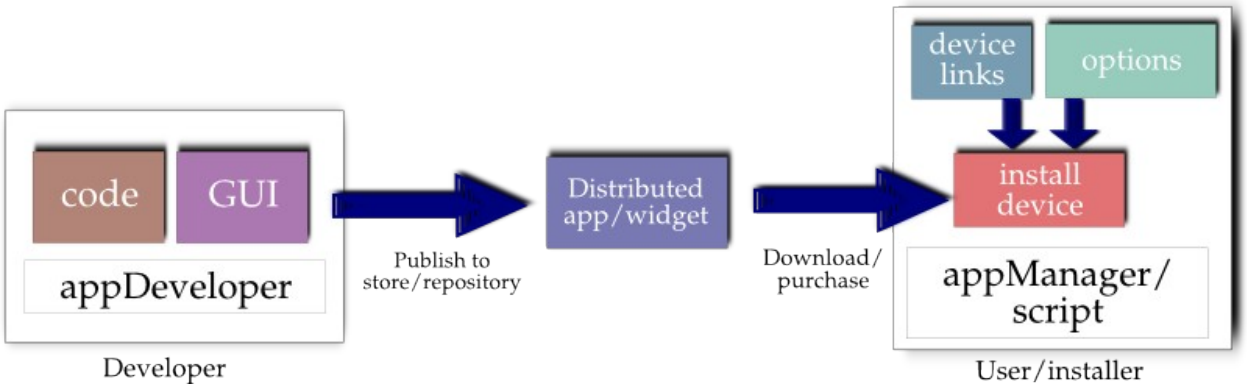

Fig 6: Bytecode app flow

External controls: Even though Halo/S bytecode are the preferred control method, for simpler control needs it is sometimes simpler to implement external "brute-force" control software to do the controlling. This external control is commonly done through node.js scripts or HTML5 applications, and allows a computer to read and write devices directly to implement a rudimentary control scheme. This method can prove to be the simplest to implement, however it can severely impact system performance such as battery life (due to more frequent wireless communication) and impact other bytecode apps running on the system.

Conclusion

The Halo/S core v1.0 platform introduces a truly next-gen system for data acquisition, controls, and M2M functionality. It has been designed to fundamentally reduce the installed system cost to 1/3 of all competing systems, while simultaneously offering an unmatched combination of upgradability, integration ease, security, and performance over other platforms.

For more information on Halo/S, please contact us at info@halo-s.com

Revision history

v1.0	2013/12/09	Initial release