Helios: A Programmable Software-defined Solar Module

Noman Bashir
UMass Amherst
nbashir@umass.edu

David Irwin
UMass Amherst
deirwin@umass.edu

Prashant Shenoy
UMass Amherst
shenoy@cs.umass.edu

ABSTRACT

The declining cost and rising penetration of solar energy is poised to fundamentally impact grid operations, as utilities must continuously offset, potentially rapid and increasingly large, power fluctuations from highly distributed and "uncontrollable" solar sites to maintain the instantaneous balance between electricity's supply and demand. Prior work proposes to address the problem by designing various policies that actively *control* solar power to optimize grid operations. However, these policies implicitly assume the presence of "smart" solar modules capable of regulating solar output based on various algorithms. Unfortunately, implementing such algorithms is currently not possible, as smart inverters embed only a small number of operating modes and are not programmable.

To address the problem, this paper presents the design and implementation of a software-defined solar module, called Helios. Helios exposes a high-level programmatic interface to a DC-DC power optimizer, which enables software to remotely control a solar module's power output in real time between zero and its current maximum, as dictated by the Sun's position and weather. Unlike current smart inverters, Helios focuses on enabling direct programmatic control of real solar power capable of implementing a wide range of control policies, rather than a few highly-specific operating modes. We evaluate Helios' performance, including its latency, energy usage, and flexibility. For the latter, we implement and evaluate a wide range of solar control algorithms both in the lab, using a solar emulator and programmable load, and outdoors.

CCS CONCEPTS

•**Hardware → Smart grid;**

KEYWORDS

Solar, Control, MPPT, Net meter

ACM Reference format:
Noman Bashir, David Irwin, and Prashant Shenoy. 2018. Helios: A Programmable Software-defined Solar Module. In *Proceedings of The 5th ACM International Conference on Systems for Built Environments, Shenzen, China, November 7–8, 2018 (BuildSys '18),* 11 pages.
DOI: 10.1145/3276774.3276783

BuildSys '18, Shenzen, China

978-1-4503-5951-1/18/11…$15.00
DOI: 10.1145/3276774.3276783

1 INTRODUCTION

Due to its continuously declining cost, solar generation capacity is rapidly expanding, with the aggregate amount increasing 50% worldwide in just the last year from 50GW to 76GW [16]. The increase is largely due to a steady drop in the cost of solar modules, which decreased by 2× from 2009 to 2015 (from ~$8/W to ~$4/W) [7]. Utilities and governments are responding to rising solar penetration in multiple ways. Since solar is typically installed "behind the meter" and, similar to demand, viewed as uncontrollable, many utilities are altering their generation portfolio to include more small "peaking" generators with fast start-up times and high ramp rates to better compensate for large solar fluctuations. Unfortunately, these peaking generators are much less efficient—in terms of both cost and carbon emissions—than intermediate and baseload generators. In parallel, governments are defining complex interconnection policies that determine who can connect solar modules to the grid. In the U.S., these policies vary widely by state, but generally place hard caps on the total solar capacity that can connect to the grid. After reaching these caps, users can no longer connect solar to the grid until the caps are raised, which may take months-to-years. For example, homeowners in Hawaii were recently barred from connecting solar for more than two years [5, 10].

Rather than prevent users from connecting solar to the grid, an alternative approach is to let anyone connect solar to the grid, and then actively regulate or control the solar power fed into the grid in real time. This approach is reminiscent of congestion control in the Internet, which enables anyone to connect, but then rate-limits traffic, using transport protocols such as TCP, to ensure the network does not collapse and users receive their fair share of the bandwidth. Such an approach, when applied to solar rate control, effectively transforms solar modules into small "dispatchable" generators similar to fuel-based mechanical generators, albeit with some important differences. For example, while solar enables more precise and rapid control than mechanical generators, its maximum capacity is not static, but is a function of the time and weather. There has been a significant amount of recent prior that has argued for such dynamic solar congestion control algorithms in the context of smart solar arrays. This ensemble of work proposes mechanisms and policies that use dynamic solar power curtailment to meet a variety of different objectives, including maintaining constant power [13, 14], providing voltage/frequency stability [12, 15], improving grid reliability under high solar penetration [8], ensuring fairness [3, 4, 13], and preserving energy data privacy [11].

However, these policies implicitly assume the presence of "smart" solar modules or arrays capable of regulating their output programmatically in real time based on well-defined algorithms. Unfortunately, there is currently no way to implement such algorithms, as smart inverters embed only a small number of operating modes,

primarily focused on improving AC power quality and grid reliability, and are not programmable. Thus, the policies above were evaluated in simulation and never implemented.

To address the problem, this paper presents the design and implementation of a programmable software-defined solar module, called Helios. Helios exposes a high-level programmatic interface to a DC-DC power optimizer, which enables software to remotely control a solar module's power output between zero and its current maximum, which is dictated by the Sun and weather. Unlike current smart inverters, Helios focuses on enabling direct programmatic control of real solar power capable of implementing a wide range of control policies and protocols, rather than a few highly-specific operating modes. The operating modes implemented by smart inverters typically focus on improving AC power quality and grid reliability, and are not related to current policies for managing solar penetration. Note that Helios' solar control is not intended to replace the use of energy storage. However, in most case, energy storage remains too expensive to install and maintain at most small-scale solar sites. Helios is orthogonal to existing demand response programs that regulate loads to balance grid supply and demand. Helios effectively enables similar capabilities for solar.

We implement a Helios prototype, which includes a library that exposes a simple programmatic interface and enables fine-grained control of solar module output. Helios is self-powered using power harvested from the solar module. We describe the programmatic interface in detail in §3. We evaluate both Helios' platform performance and its flexibility. Our performance evaluation focuses on Helios' basic functions, including its energy usage and response latency. To evaluate flexibility, we implement multiple solar control policies proposed in prior work, including constant power generation [13, 14], Weighted Power Point Tracking (WPPT) [13], and load imitation for privacy [6, 9, 11], as well as develop and implement new algorithms, such as constant net metering and ramp rate control. We show that Helios admits simple implementations for each these algorithms. Our hypothesis is that Helios is simple, reliable, and cheap, and its interface enables the implementation of a wide range of solar control algorithms. In evaluating our hypothesis, we make the following contributions.

Solar Control Background. We describe the basic functions Helios uses to enable solar control. We then outline the potential benefits of dynamic solar control to grid operations, and place Helios in context with emerging work on smart inverters, which leverage inverters to improve grid reliability and power quality.

Helios Design and Implementation. We present Helios' design and implementation, which exposes a simple programmatic API that applications use to control solar output. We then describe the implementation of multiple solar control algorithms atop this API.

Experimental Evaluation. We evaluate Helios' performance and energy usage, as well as the algorithms we implement. To enable repeatability, in addition to outdoor experiments, we also use a solar array simulator (the Chroma 62000H-S), which mimics solar's current response to voltage changes and replays solar I-V curves from weather data traces. Our results show that Helios consumes little energy (<1.6W), is able to precisely and nearly instantly control solar output, enabling a wide range of higher-level applications.

2 BACKGROUND

Grid-tied solar deployments consist of multiple solar modules wired together and interconnected with the grid via one or more inverters, which convert the low-voltage DC power generated by the modules to the higher voltage (120V) AC power of the grid. Most solar deployments use inverters to connect to the grid, since this enables them to feed surplus solar energy into the grid and relieves them of having to locally balance electricity's supply and demand. In contrast, standalone solar deployments that are not grid-tied require energy storage, generally in the form of a battery, to store surplus solar energy and balance supply and demand. Since batteries are expensive to install and maintain, there are few standalone solar deployments with batteries. In addition, most solar deployments consist of arrays of multiple connected solar modules. The output of these modules is dependent, not only on the solar irradiance incident on them, but also the module wiring and the placement of the inverters. In particular, solar deployments generally include electronics that perform Maximum Power Point Tracking (MPPT) algorithms, which dynamically adjust the operating voltage to extract the maximum power from the module (or array).

MPPT and Solar Array Architectures. The power output of a photovoltaic (PV) panel is governed by its I-V curve, which defines the relationship between a solar module's output current (I) and its operating voltage (V), as depicted in Figure 1. Any operating voltage, up to the maximum shown, can be chosen to operarate the panel, and the I-V curve dictates the current output for that operating voltage, which in turn determines the power that will be generated by the panel. The nature of the I-V curve is such that, as the operating voltage increases, the output current remains steady up to a knee point, after which it begins to decrease. Since the output power generated is the product of voltage and current, the power produced for different operating voltages, depicted by the I-V curve in Figure 1 , increases monotonically until the knee, which represents the maximum possible power that the solar module can generation, which is referred to as the maximum power point (MPP). Importantly, the MPP is not static, since a module's I-V curve continuously changes based on the incident solar irradiance, which is a function of many factors, including weather, time, temperature, shading, dust, snow, etc. In addition, a solar array's I-V curve can be highly complex, as it is an additive combination of the I-V curves of the underlying modules. Nearby modules may also experience different conditions that cause their I-V curves to differ significantly, e.g., such as a shaded module next to a module without shade, which also contributes to the complexity.

Since module and array dynamics are impossible to predict *a priori*, MPPT algorithms continuously search for the operating voltage that yields the MPP. The most common algorithm is Perturb and Observe (P&O), which simply searches for the MPP by perturbing the operating voltage by a small amount and measuring the current to compute a new power ($P(t)$). If this new power is greater than $P(t-1)$, the algorithm changes the voltage again in the same direction; however, if the new power is less than $P(t-1)$, then the algorithm reverses the direction of change. While more advanced MPPT algorithms that converge faster exist, in most instances, P&O is fast enough, often converging in less than one

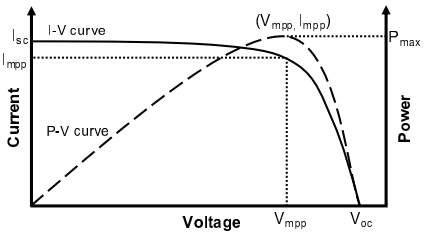


Figure 1: Idealized I-V and P-V curves for a solar module.

cycle of the grid's 60Hz frequency. As a result, most MPPT chips, including the SM72442 that Helios uses, employ P&O for MPPT.

MPPT algorithms and inverters can be integrated into solar arrays using multiple architectures. For example, the traditional configuration directly wires solar modules together and connects them to a centralized, i.e., string, inverter, which both executes MPPT and converts solar's DC input to AC output synchronized to the grid's frequency and phase. However, this approach is inefficient as the string inverter is only able to find the MPP of the entire array and not each module. As a result, the current output of modules wired in series will be limited by the current of the lowest producing module, i.e., the most shaded module. Thus, another approach is to use microinverters at each module, which also execute MPPT as well as convert DC to grid-synchronized AC. Microinverters are an example of Module-Level Power Electronics (MLPE) that improve efficiency, since they enable each module to independently optimize its own MPP. However, microinverters are expensive, especially for large-scale deployments, as they replicate the complex inverter electronics across every module. Microinverters are also viewed as less reliable, since, unlike with a string inverter, which is typically placed inside, they are attached to the module and subject to harsh weather conditions. Thus, another architecture has emerged that uses DC power optimizers, which, like microinverters, enable each module to operate at its MPPT. However, unlike microinverters, DC power optimizers only execute MPPT and do not convert DC to AC, instead relying on a separate, often centralized, inverter.

There are numerous tradeoffs between solar architectures based on string inverters, microinverters, and DC power optimizers. We chose to design Helios as a programmable DC power optimizer for multiple reasons. First, power optimizers are simpler, cheaper, and more reliable than microinverters because they have fewer sensitive electronic components, and perform a narrower set of functions. In addition, power optimizers are not grid-facing and capable of controlling solar output independent of grid-quality metrics, which are the focus of much of the existing work on smart inverters. That is, power optimizers are orthogonal to, and do not interfere with, many of the grid power quality optimizations being implemented in smart inverters. There are many grid-quality support functions inverters can provide, such as reactive power compensation and voltage/frequency ride through, that are not related to regulating solar power output. In the U.S., many of these functions are strictly regulated by local ISOs and RTOs. Since power optimizers are not grid-facing and focus on real power, they are not subject to the same regulations as smart inverters, i.e., IEEE 1547-2018 [1].

Helios Motivation. Helios' basic approach leverages the same voltage control mechanisms as the MPPT algorithms already included in DC power optimizers. However, rather than search for the voltage that yields the MPP, Helios enables applications to set the output voltage such that it yields any power point between zero and the maximum power. Since only solar's MPP is intermittent, based on environmental conditions, Helios enables applications to precisely and rapidly control power output below the MPP.

Helios is broadly related to solar and wind curtailment, which focuses on disconnecting large solar and wind farms from the grid during off-peak times when their energy is not needed, i.e., when prices go negative. However, solar and wind curtailment is generally a coarse mechanism used as a last resort to preserve grid reliability. In contrast, Helios exposes mechanisms that enable fine-grained control of solar output at the module level. We initially expect these mechanisms to be useful in setting solar net metering policies. Current policies are static, and limit the aggregate capacity of deployments that can connect to the grid, even though these deployments rarely, if ever, generate their rated capacity. Helios would enable new dynamic policies that actively rate limit the solar power injected into the grid, as discussed in recent work [3].

In addition, the presence of rate-limited solar deployments operating below their MPP opens up a new high-quality form of reserve capacity for the grid. That is, the grid can call on this capacity instantaneously, i.e., at sub-second scales, if necessary to satisfy unexpected increases in demand. In this sense, solar represents much higher quality reserve capacity than mechanical generators, which may take minutes to tens of minutes to activate. In addition, mechanical generators operated as reserve capacity must be constantly maintained and tested to ensure their correct operation. Similar maintenance and testing for solar is less costly. In some sense, Helios can be thought of as enabling the equivalent of demand response for solar by exposing mechanisms for utilities (or others) to remotely decrease (or increase up to the MPP) the solar supply. However, unlike existing demand response resources, such as air conditioners and heaters, solar control is transparent to users.

Current work on "smart" solar functions have focused on smart inverters. However, there are no commercial offerings similar to Helios that provide remote programmatic control of solar output. As discussed earlier, smart inverter functions are governed by strict regulations, and generally focus on implementing specific approved operating modes that support grid reliability. These operating modes are generally not programmable, and target AC power quality functions and not power generation. While the lack of programmability is likely due to the stringent regulations around grid-facing equipment, it does hinder grid innovation, especially as the grid evolves into a more decentralized architecture. As we show, Helios is capable of implementing a wide range of solar control algorithms, and also serves a platform for developing new algorithms.

3 HARDWARE AND SOFTWARE DESIGN

We describe both the hardware and software design of Helios, our programmatic software-defined solar module, below.

3.1 Hardware Design

Our hardware design consists of four primary components: a programmatic MPPT chip (the SM72442), a DC-DC converter, a DC power supply, and a processing and communication platform, as shown in the block diagram in Figure 2(left). The output of the

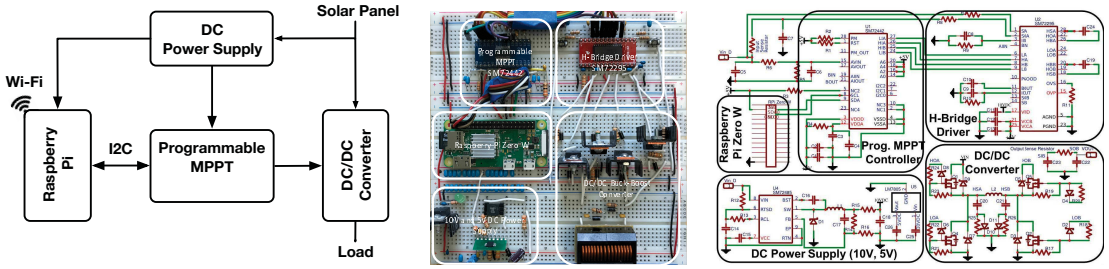


Figure 2: Helios block diagram (left), board layout (middle), and circuit diagram (right).

solar module first connects to a DC-DC converter, which itself connects to the MPPT chip. By default, the chip executes the MPPT algorithm, which controls the DC-DC converter to maintain the module's operating voltage at the MPP, e.g., the knee of the curve from Figure 1. While the DC-DC converter varies the operating voltage, it maintains a fixed output voltage using a buck-boost converter, enabling it to connect to other devices requiring a fixed voltage, such as other modules, an inverter, or a battery.

The MPPT algorithm is embedded into the MPPT chip (the SM72442), which also provides a programmatic interface to set thresholds via registers for the module's maximum output voltage and current, as we discuss in §3.1.1. A network-connected embedded processor connected to the MPPT chip is capable of controlling these thresholds via software using the chip's low-level API. The platform need not be connected to external power as it runs on power harvested from the solar module via a DC power supply. The design admits a compact implementation, intended to be encased and installed in the rear of the panel. Figure 2(middle) shows the board layout of our current prototype with the major components labeled. We are currently fabricating a Printed Circuit Board (PCB), which enables a more compact layout. Figure 2(right) shows the circuit diagram with the precise connections between each of the components. Below, we discuss each of the major components.

3.1.1 Programmable MPPT Chip. Many commercial integrated circuits (ICs) implement MPPT, but few expose dynamic control over a solar module's output voltage and current. To the best of our knowledge, TI's SM72442 is the only MPPT controller IC that exposes a programmatic interface that enables such control. TI has an evaluation board (the SM3320) that uses the SM72442's control interface as part of a battery charge controller, which steadily reduces current as the battery voltage rises (indicating that it is nearing full capacity). Other similar evaluation boards exist for battery charge controllers, which also enable setting current limits based on voltage, but they do so in hardware, enforce only a single limiting policy (for a particular battery), and are not programmable. The SM72442 chip is also used for remote on/off control and standard MPPT in some DC solar power optimizers. However, there are no commercial products that expose the chip's full functions for dynamic control over a solar module's output voltage and current.

The key components of the SM72442 IC are its MPPT controller, Analog-to-Digital Converter (ADC) controller, and I2C module, which enables serial communication to configure the chip. The chip implements the P&O MPPT algorithm, which generates the pulse-wide modulation (PWM) signal that varies the duty cycle of the DC-DC converter (described below) to determine the module operating voltage that maximizes solar power output. The chip's digital controller logic also implements the voltage/current limiting functions. These functions also generate a PWM signal, which varies the duty cycle of the DC converter, but instead does so to enforce a limit on current or voltage, rather than maximize power. The current and voltage limiting functionality requires an algorithm similar to MPPT, as it must also vary the operating voltage as the I-V curve changes to enforce the limit [13]. Thus, the chip uses a similar P&O algorithm as MPPT to maintain a current, voltage, or power limit. If the maximum solar output falls below the power limit, then the chip falls back to MPPT and maximizes power. The MPPT algorithm converges rapidly, and is able to track a module's MPP within 12ms, or less than the length of a single 60Hz AC cycle.

The chip's ADC controller controls an 8-channel, 12-bit ADC used to sense the input and output voltage and current, as well as set the chip's configuration. The controller exposes four A/D channels used to configure the maximum current and voltage thresholds, the conditions at startup, and the output voltage slew rate, i.e., the rate at which voltage is allowed to change. However, the I2C module provides an interface to override these external A/D channels and configure the chip using the SM72442's internal registers. Our design overrides the external A/D channels and programmatically configures the chip using these registers. There is a 10-bit field in the register for setting each of the maximum output voltage and current thresholds. Similarly, all of the other settings for the MPPT chip can be configured via these registers. The chip's API also enables monitoring of the instantaneous input and output voltage and current. As we discuss, our software API includes functions that expose these sensor values to higher-level applications.

3.1.2 DC-DC converter. Our DC-DC converter is a standard buck-boost converter that can increase or decrease a solar module's output voltage to a desired output voltage. In addition, the converter works in conjunction with the SM72442's algorithm above to control the module's power output. In particular, the converter acts as a load on the solar module, whose resistance can be controlled between zero and ∞ by varying the converter's duty cycle. In this case, a duty cycle of zero represents infinite resistance and voltage (resulting in zero output current and power), while a duty cycle of one represents zero resistance and voltage (resulting in short-circuit output current and zero power). The SM72442's MPPT and limiting algorithm adjust the module's voltage along the I-V

curve by altering the DC-DC converter's duty cycle between these extremes, and observing the resulting current and power output.

The core of the buck-boost converter is an inductor combined with high frequency switches, which control the charging and discharging of the inductor that ultimately dictates the resistance and input/output voltage. We use IRF3205 N-channel MOSFETs as switches. In our prototype, these have a current rating of 110A and voltage rating of 50V, and thus can accommodate standard 300W solar modules. However, the voltage level and current provided by the SM72442 MPPT chip is not high enough to directly drive the switches at high frequency. As a result, we also use a MOSFET driver chip, the SM72295, to translate the 5V from the MPPT chip (SM72442) to the 10V required to drive the switches and provide high enough current for fast switching. In addition, the MOSFET driver chip provides amplifiers for input/output current sensing that removes ripple current and provides averaged current data to the MPPT chip. The buck-boost converter also uses snubber circuits at its input and output to reduce voltage ripples. The MOSFET driver chip has buffered outputs available to provide both isolation and an interface to the current sense pin of the MPPT chip. Since the MOSFET switches are rated at 175C, we use heatsinks with them, as Helios' operating temperate may exceed this when placed behind a module installed on a roof under intense sunlight.

3.1.3 Sensing Circuitry. The MPPT chip (SM72442) requires measurements of both the solar module's and the buck-boost converter's output voltage and current as inputs to the dynamic MPPT and current/voltage limiting algorithm. The voltage sensing at both points is done using a resistor divider circuit that scales the maximum voltage to ~4V, which is 80% of the full scale A/D input. We introduce this rough limit to avoid high voltage at the SM72442 pins and possible burn out. The current sensing at both points is done using 0.04Ω sense resistors. We use the current sense amplifiers of the SM72295 MOSFET driver chip set to 0.44V/A gain, and filter the voltage across the sense resistors. Finally, the buffered output from the MOSFET driver chip is fed to the SM72442 MPPT chip.

3.1.4 DC Power Supply. Our design powers all components using the available solar power. We first use a DC-DC switching regulator IC (the SM72485) to step-down solar module voltage to 10V, which is required by the MOSFET driver chip for switching the MOSFETs. We then use a low power 5V regulator (the SM72238), to further step down the voltage for powering the SM72442 and our processor (the Raspberry Pi Zero). We design the DC power supply circuitry to work with solar modules having an open circuit voltage in the range of 9V to 50V, which is compatible with standard residential and commercial solar module capacities.

3.1.5 Processing and Communication Platform. Rather than an embedded microcontroller, our design leverages a Raspberry Pi Zero (W), which includes a built-in WiFi module. The Raspberry Pi zero is low power, low cost (~$10), and supports a standard Linux software stack. We select this full-featured platform to both enable advanced control algorithms that require non-trivial local computation, such as advanced solar forecasting techniques, and to ease programmability by exposing a high-level programming environment, such as python, to developers. The platform includes a 1Ghz single-core CPU, 512MB RAM, an 802.11 b/g/n wireless module and consumes a maximum of 170mA at 5V (or 0.85W) when WiFi is activated. The platform also includes an I2C interface enabling it to communicate with the SM72442 MPPT chip's I2C interface. We set the baud rate of the I2C bus to 100kHz to enable high speed monitoring and updates of voltage and current limits. Network connectivity is important to enable remote programmability. In cases where WiFi does not penetrate roofs, using powerline networking or the cell network are options, although our current prototype does not support them. In a solar array, as long as one solar module has WiFi connectivity, e.g., one near a window, it can act as a gateway for an ad hoc mesh network that connects the others.

3.2 Software Design

The Raspberry Pi platform above runs a full Linux software stack, including python and various drivers that interact with the MPPT chip. Our software includes a python library that exposes a simple API to developers. The API's functions use the smbus library protocol to write the required register values to the SM7442. Table 1 shows the five primary functions in this API, which mostly consist of getter and setter methods. The `init()` function initializes the SM72442 MPPT chip and sets the appropriate configuration bits in its registers to override the external ADC channels, enabling software control of its functions. The `ping()` function simply checks if the MPPT chip is alive and responding to commands.

The APIs' three key functions focus on setting a solar power limit, monitoring power output, and retrieving the maximum power point. `setPowerLimit(int)` takes a power value in Watts as input and configures the MPPT chip to limit solar output to that value. Note that setting the power limit to any value below zero or beyond the MPP causes the MPPT chip to fall back to tracking the MPP. Our prototype enables software to set power limits at the granularity of a Watt, which should be sufficient for standard modules with maximum capacities on the order of 300W. Likewise, `getPowerOut()` returns the current output power in Watts; if a limit is set, then this value should be at the limit, since there is little imprecision in the tracking algorithm. Finally, `getPowerMax()` returns the current MPP. This meta-function configures the MPPT chip to perform MPPT, stores the MPP value and then resets the power limit, if any, before returning to the MPP. We include this meta-function as it is often useful to know how much solar power we are curtailing.

Table 2 lists some additional auxiliary functions the API supports These functions generally enable setting voltage and current limits separately, as well as getting voltage and current at Helios' input (at the solar module) and output (at the inverter/battery), as well as the current/voltage limit settings and the current/voltage at the MPP. Note that we label the return values of these functions as dynamic if they represent a dynamic system measurement determined by the environment, and not a stored value. Our software packages these functions into a `helios` python library that software running locally can import. In addition, we also wrote a small server in python that exposes these functions remotely via a REST API, which enables remote control by an external client. We present the REST API, in Tables 1 and 2, where 127.0.0.1 can be replaced with Helios' IP address. Such remote control may be useful for coordinating control of multiple modules, where they communicate to perform various tasks such as enforcing aggregate limit across distributed

Function	REST API	Description	Type
`setPowerLimit(int)`	POST 127.0.0.1:5000/helios/P:<int limit>	set power limit	-
`getPowerOut()`	GET 127.0.0.1:5000/helios/power-out	get output power	dynamic
`getPowerMax()`	GET 127.0.0.1:5000/helios/power-max	get MPP power	dynamic
`init()`	POST 127.0.0.1:5000/helios/init	initializes platform	-
`ping()`	POST 127.0.0.1:5000/helios/ping	checks chip status	-

Table 1: Helios Library Primary Functions

Function	REST API	Description	Type
`setVoltageLimit(int)`	POST 127.0.0.1:5000/helios/V:<int limit>	set maximum voltage	-
`setCurrentLimit(int)`	POST 127.0.0.1:5000/helios/I:<int limit>	set maximum current	-
`getVoltageLimit()`	GET 127.0.0.1:5000/helios/voltage-limit	get maximum voltage	static
`getCurrentLimit()`	GET 127.0.0.1:5000/helios/current-limit	get maximum current	static
`getVoltageOut()`	GET 127.0.0.1:5000/helios/voltage-out	get output voltage	dynamic
`getCurrentOut()`	GET 127.0.0.1:5000/helios/current-out	get output current	dynamic
`getVoltageIn()`	GET 127.0.0.1:5000/helios/voltage-in	get input voltage	dynamic
`getCurrentIn()`	GET 127.0.0.1:5000/helios/current-in	get input current	dynamic
`getVoltageMax()`	GET 127.0.0.1:5000/helios/voltage-max	get MPP voltage	dynamic
`getCurrentMax()`	GET 127.0.0.1:5000/helios/current-max	get MPP current	dynamic

Table 2: Helios Library Auxiliary Functions

modules. While coordinating control of multiple Helios modules presents an interesting research problem, it is outside the scope of this paper. Note also that, while Helios is designed as a DC power optimizer for an individual solar module, it can also work with arrays of multiple solar modules (based on their aggregate I-V curve), which eliminates the need to coordinate multiple modules.

4 IMPLEMENTATION CONSIDERATIONS

Challenges. We faced a number of challenges in Helios' implementation. For example, at the end of each day the voltage output of Helios' DC power supply would not switch directly from its required 10V/5V (for the Raspberry Pi and MPPT chip) to 0V, but would drop below the required voltage. When applied to the MPPT chip (SM72442) and the MOSFET driver chip (SM72295), such low voltages would alter the chips' configuration, which would persist until the devices were powered back on the next day. To address the problem, we save the configuration state in the Raspberry Pi and reset its configuration at the start of each day. A similar problem occurred during cloudy days, as voltage may drop below threshold briefly due to clouds. To address intermittent drops, we added a capacitor at the DC power supply as a buffer to store energy and supply it to maintain voltage during cloudy periods.

Our implementation also required python's smbus library to read the MPPT chip's registers. However, the chip's communication protocol requires "repeated start" functionality, which required explicit activation within the smbus library to use. Specifically, to read a register with "repeated start," the address of the slave devices and the target register first have to be written on the I2C bus; only then will the device respond by sending the length of data bytes followed by the register contents. This read sequence requires the master to switch from initially writing to reading without terminating the communication at the end of the first set of writes, i.e., a "repeated start." A standard write will not work, as it terminates the communication before it switches to reading. This basic protocol enables software running in the platform to control the MPPT chip via its low-level interface.

Finally, the MPPT chip's datasheet specifies that changing the voltage and current limits requires writing a 10-bit field to a specific register. However, the datasheet did not specify how values within this 10-bit field translate into decimal voltage and current values. Thus, we had to reverse engineer the translation through experimentation, e.g., by setting values and reading the voltage output. Since the there were only 1024 values of the registers, it was possible to set every value and observe its result.

Cost. Table 3 breaks down the cost of our prototype and Helios' cost at scale. For comparison, a 300W SolarEdge power optimizer (P300) currently costs ~$60. Thus, Helios' cost is in-line with current power optimizers. While its at-scale cost is significantly less, the cost of commercial products include additional costs beyond hardware, e.g., marketing, certification, etc. However, our breakdown does indicate that Helios appears to be in the same cost range as existing power optimizers that are not programmable.

5 EXPERIMENTAL EVALUATION

We evaluated Helios in emulation and using a deployed solar module. Our emulation connects Helios to both a solar array simulator (SAS) and a programmable DC load. The SAS acts like a programmable power supply that mimics the electrical response of an I-V curve. That is, the SAS enables us to configure a specific I-V curve, such that when its operating voltage changes (due to the attached load) it will alter its output current according the curve. In contrast, a traditional DC power supply will not change its output current in response to changes in voltage, but will instead maintain a steady current (depending on its setting). We use the Chroma 62020H-150S as our SAS, which also supports a real-world weather mode. This mode is capable of replaying solar radiation traces that alter the maximum power point of the I-V curve to match the traces. In our emulation, Helios sits between the SAS and a programmable DC load, which is also able to replay power traces of demand. Figure 3 depicts our lab setup. Our real experiments simply replace the SAS with a solar module. However, there are many drawbacks to live experimentation, since they must be run outdoors subject

Component	1x cost ($)	10,000x cost ($)
SM72442	11.68	5.31
SM72295	4.56	2.22
SM72485	3.19	2.16
Raspberry Pi	10	10
Inductor	10.19	7.71
MOSFETS	6.4	2.48
Other components	10	7
PCB development	10	1
Total	**$66.02**	**$37.88**

Table 3: Helios cost breakdown

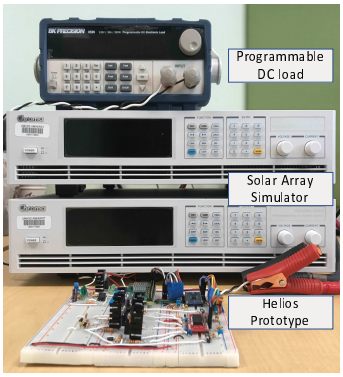


Figure 3: Emulation configuration with Helios sitting between a solar array simulator and a programmable DC load.

to real weather conditions (requiring weather-proofing) and are not repeatable. Thus, most of our experiments use the SAS. We run microbenchmarks to evaluate Helios' response latency and power consumption. However, Helios' primary contribution is in the flexibility it provides to implement a wide range of control algorithms. To evaluate this, we conduct a number of case studies that implement a wide variety of solar control algorithms proposed in prior work, but never implemented.

5.1 Microbenchmarks

Response Latency. Helios' response latency is significant as it dictates its ramp rate, or the rate at which it can increase or decrease its power output. A conventional dispatchable generator's ramp rate is an important metric, since it determines how well the generator can respond to changes in grid demand. We measure response latency by recording the time between calling Helios' application-level function in python to set a power limit to when we can verify that the power limit has been set. To verify the latter, we call Helios' application-level function to verify the current power state. We ran 1000 experiments that altered the power limit in this way, and found that the average latency to change power was 12ms at 99th percentile with worst case latency of 34ms. This latency was independent of the magnitude of the power changes, and less than the 16.7ms cycle length of 60Hz AC power. These experiments demonstrate that, from the perspective of the AC grid, Helios effectively enables an infinite ramp rate 99% of the time. Even the worst case response of 34ms is multiple orders of magnitude less

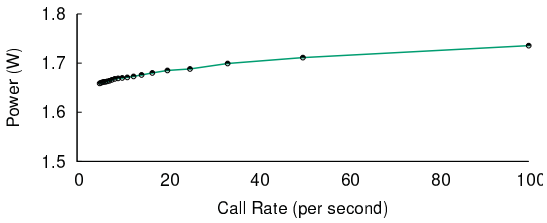


Figure 4: Power consumption vs call rate per second

Variable	Description
P_S^{limit}	solar power limit
P_S^{weight}	solar power weight
P_S^{max}	maximum solar power
$P_D(t)$	grid demand at time t
P_D^{limit}	grid demand limit
α	smoothing factor for EWMA
P_S^T	solar power data over T
P_H	helios power output
Δ	time between power updates

Table 4: Variable definitions for Algorithms 1-4

than the most agile conventional power generators, which take on the order of minutes to ramp up.

Power Consumption. Helios' power consumption ranges between 0.9W and 1.6W with the Raspberry Pi's consumption alone ranging between 0.7W and 0.8W. By contrast, the MPPT chip's power is insignificant, ranging from 0.08W to 0.1W depending on its temperature. Figure 4 plots the power consumption as a function of the function call rate to alter the power, which shows that the power consumption is steady even under intensive use of the platform and the MPPT chip, e.g., when changing power levels every 10ms. Since Helios is designed to support modules with capacity of ~300W, its power consumption is insignificant. In addition, the low power consumption should enable Helios to run for the entire day.

5.2 Case Studies

In our case studies, we use Helios to implement a wide range of solar control algorithms. In each case, we list the algorithm's pseudocode when implemented using Helios. The variable definitions for this pseudocode are shown in Table 4. Note that for our emulation experiments, we use traces that report solar radiation every five minutes. We accelerate our emulation experiments by 300× by changing power every second based on the five minute values. Note that accelerating the experiments stresses our prototype much more than in practice by increasing the frequency of power variations. We quantify accuracy using the Mean Absolute Percentage Error (MAPE) between the limit and the actual generation, as below. A MAPE near 0% is better, as it indicates better accuracy.

$$MAPE = \frac{100}{T}\sum_{t=0}^{T}\left|\frac{P_S^{limit}(t) - P_H(t)}{P_S^{limit}(t)}\right| \quad (1)$$

5.2.1 Constant Power Generation. The simplest possible "algorithm" is to set and maintain a constant power output. Recent work has proposed different various approaches to constant power generation [13, 14]. These approaches are similar to the algorithmic variants of MPPT in that, while interesting, simple approaches, such as P&O, are quite effective. Since our MPPT chip natively supports

Algorithm 1 Constant Power Generation

Require: P_S^{limit}
1: $Helios.init()$
2: $Helios.setPowerLimit(P_S^{limit})$

Algorithm 2 Constant Net Metering

Require: $P_D^{limit}, P_D(t)$
1: $Helios.init()$
2: **while** (1) **do**
3: **if** $(P_D(t) > P_D^{limit})$ **then**
4: $P_S^{limit} = P_D(t) - P_D^{limit}$
5: **else**
6: $P_S^{limit} = 0$
7: $Helios.setPowerLimit(P_S^{limit})$
8: $sleep(\Delta)$

setting power limits using a modified P&O algorithm, and Helios' API directly exposes this functionality, the Helios pseudocode in Algorithm 1 for constant power generation is straightforward. Figure 5 demonstrates our constant power generation algorithm on a sunny (a) and cloudy (b) day in emulation. In this case, we set Helios to maintain a fixed power of 50W on a 100W solar panel's output. We also ran the algorithm outdoors on a real 100W panel on a sunny day as well, although we only ran it for an hour, during which solar radiation was relatively constant. These experiments demonstrate that Helios, and its underlying MPPT chip's algorithm, are nearly perfect at maintaining constant power generation as the MAPE is 1.06%, 1.13%, and 0.83% for the sunny day, cloudy day, and outdoors, respectively.

5.2.2 Constant Net Metering. A slightly more complex algorithm is constant net metering, which attempts to maintain constant net power for a solar-powered building or home. Constant net metering has multiple potential benefits. For example, while current policies regulate grid-tied solar capacity to control its effect on the grid, they are indirect. In contrast, regulating users' net meter demand would directly control users' grid impact. In particular, by regulating users' net demand, governments could incentivize users to alter their demand in response to their solar to minimize their aggregate impact on the grid. Our constant net meter generation algorithm (Algorithm 2) regulates a home's net demand to maintain it at a constant level and remove all variations in power. Prior work proposes a similar approach to flattening a home's net demand to preserve user privacy by preventing utilities from applying energy analytics algorithms, which analyze changes in a building's power to infer a range of behaviors, such as appliance usage and occupancy [6, 9]. The primary difference here is that prior algorithms use batteries or thermal energy storage, e.g., in water heaters, to flatten demand, while Helios controls solar power.

Figure 6 demonstrates Helios maintaining a constant net meter power. Here, the demand and net lines in the graph are negative and we translate them up on the graph by 100W so that they do not overlap with the positive solar generation. In this case, we configure our algorithm to keep the net demand constant at 0W for the home, such that it has zero impact on the grid. Note that we have scaled down a real home's demand by 10× to match it to the solar emulator's capacity. The dotted line shows how Helios varies solar power to mirror demand and maintain the flat net demand line in the middle. On the sunny day (a), Helios maintains a nearly perfect net demand, while on the cloudy day (b), there are a few periods where there is not enough solar power available to completely flatten demand, resulting in some fluctuations in net demand and a higher MAPE value, as shown in Table 5. We also ran a proof-of-concept experiment outdoors over an hour on a sunny day, where Helios was able to maintain a flat net demand with a low MAPE, as there were no fluctuations in solar power.

Algorithm 3 Weighted Power Point Tracking (WPPT)

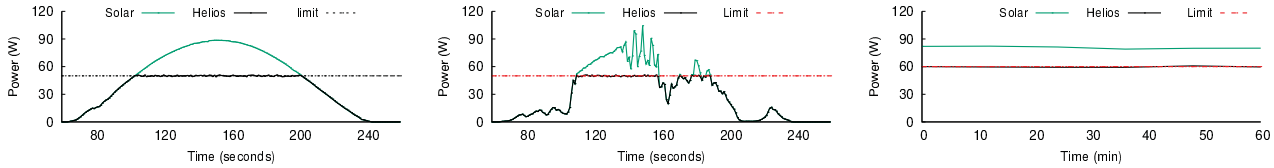
Require: P_S^{weight}
1: $Helios.init()$
2: **while** (1) **do**
3: $P_S^{max} = Helios.getPowerMax()$
4: $sleep(20ms)$
5: $P_S^{limit} = P_S^{weight} \times P_S$
6: $Helios.setPowerLimit(P_S^{limit})$
7: $sleep(\Delta)$

Algorithm 4 Solar Ramp Rate Control (SRRC)

Require: α, P_S^T
1: $Helios.init()$
2: **while** (1) **do**
3: $P_S^{limit} = ewma(P_S^T, \alpha)$
4: $Helios.setPowerLimit(P_S^{limit})$
5: $sleep(\Delta)$

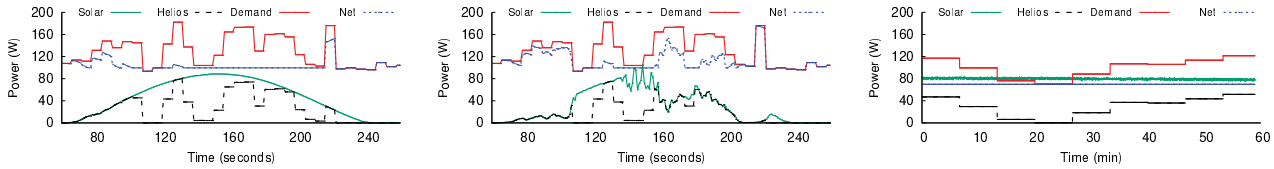
5.2.3 Weighted Power Point Tracking. WPPT is a recently proposed solar control algorithm that enables users to apply a percentage weight to their output, such that their solar output is regulated to be a specified percentage of their maximum output [13]. WPPT is inspired by similar approaches to proportional sharing in computer systems and networks, which allocate users a fraction of the available resources. Similarly, WPPT enables higher-level algorithms that both regulate aggregate solar power and ensure all users are able to generate the same fraction of power relative to their maximum capacity [3]. Since WPPT requires knowing the actual MPP and then backing off by a certain fraction, Helios' algorithm periodically finds the MPP, multiplies it by the fractional weight, and then sets the appropriate limit. Algorithm 3 shows the pseudocode for WPPT in Helios. Figure 7 shows the results of WPPT. The periodic spikes upward represent points where Helios searches for an updated MPP before scaling back. In this case, we set the weight equal to 50% and search every five seconds for a new MPP. As the graphs show, on a cloudy day (b) with frequent variations, WPPT is less accurate at maintaining its limit than on sunny days and has a higher MAPE, as in (a) and (c), with fewer variations and lower MAPE, as given in Table 5.

5.2.4 Solar Ramp Rate Control. Helios also enables new algorithms. Here, we describe an algorithm we developed to dampen



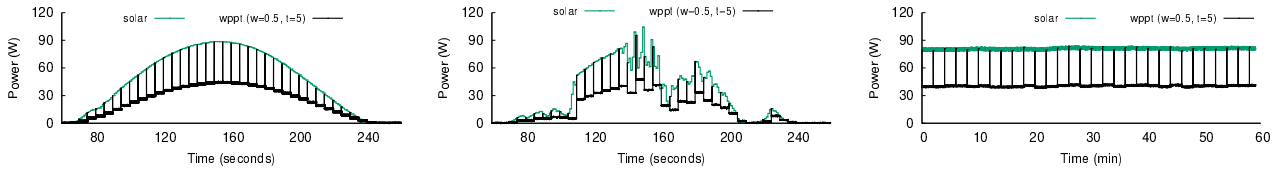
(a) Sunny Day (Emulation) (b) Cloudy Day (Emulation) (c) Sunny Day (Outdoors)

Figure 5: Helios is nearly perfect at maintaining a constant power, in this case, of 50W in various scenarios.



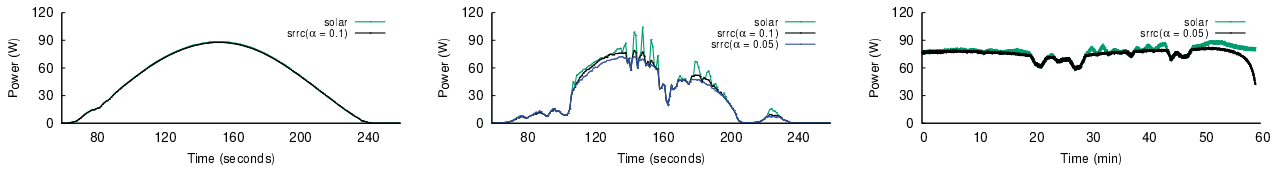
(a) Sunny Day (Emulation) (b) Cloudy Day (Emulation) (c) Sunny Day (Outdoors)

Figure 6: Helios maintaining a constant net power at 0W.



(a) Sunny Day (Emulation) (b) Cloudy Day (Emulation) (c) Sunny Day (Outdoors)

Figure 7: Helios performing Weighted Power Point Tracking (WPPT).



(a) Sunny Day (Emulation) (b) Cloudy Day (Emulation) (c) Sunny Day (Outdoors)

Figure 8: Helios executing an algorithm to dampen the ramp rates of solar power generation.

solar ramp rates based on an Exponentially Weighted Moving Average (EWMA). The algorithm simply sets the solar power limit equal to an EWMA of solar output. EWMA defines a history parameter α, which dictates the weight the moving average applies to previous values when updating the moving average, enabling us to control the level of dampening using this parameter. Note that the Helios pseudocode in Algorithm 4 is only able to dampen the rate of *increase* in solar power output, as sharp decreases are not incorporated into a moving average based on previous values.

Enabling dampening of solar power decreases would require additional logic to track the time and nearby weather conditions, e.g., such as with sky imagers, and respond to short-term predictions of solar output. We could also modify this algorithm to enforce a limit on the ramp rate, rather than simply dampening it using a moving average. Dampening the rate of increase in solar output will be increasingly important as solar penetration grows in providing time to ramp up/down mechanical generators with potentially slow ramp rates to balance fluctuations in solar power. Figure 8 shows the results of dampening the solar ramp rates. As expected, on sunny days that are already smooth, the algorithm does not alter solar power and thus has a low MAPE. However, on the cloudy day, the algorithm smooths the increases in solar power and reduces the sharp spikes in output. As there are large and frequent variations, Helios is not able to satisfy the limit perfectly, which is reflected in its higher MAPE of 4.79%, as shown in Table 5.

5.2.5 Load Imitation. Another recently proposed solar control algorithm focuses on privacy [11] by proposing to imitate the presence of loads in net meter data, rather than flatten or hide the loads. The idea is to inject fake load signatures or noise to fool energy analytics algorithms, such as Non-Intrusive Load Monitoring (NILM) or Occupancy Detecion (NIOM), into returning incorrect results. Our load imitation algorithm simply applies a limit to solar power to mirror the demand imposed by a particular load's power usage taken from a trace. That is, the algorithm reduces solar output by the demand specified in the trace, which has the same effect that the load would have on solar if it were actually present. To demonstrate this capability, we re-play a highly complex load trace for a washing machine trace from recent work [2]. Figure 9 shows the results, which depict the steady solar output and the washing machine demand on the top graph, and the resulting solar trace

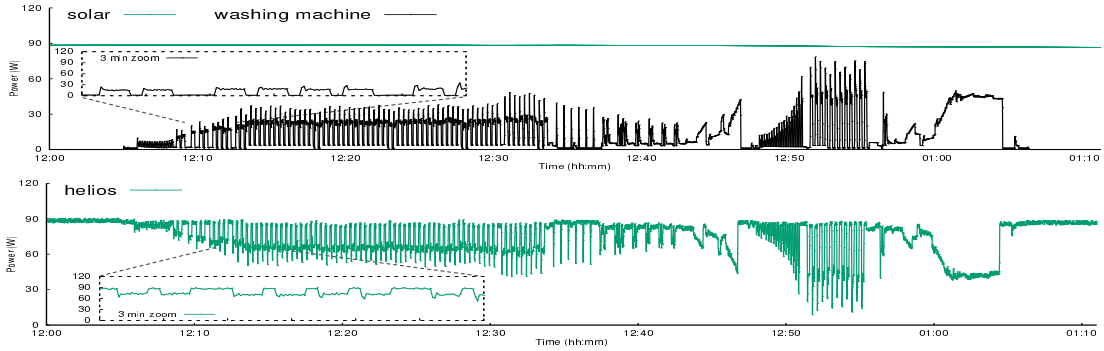


Figure 9: The top graph shows raw solar power and a washing machine load trace, while the bottom graph shows Helios modifying the solar power output to mimic the washing machine load.

Case Study	Sunny	Cloudy	Outdoor
Constant Power Generation	1.06	1.13	0.83
Constant Net Metering	5.43	11.90	0.16
Weighted Power Point	4.16	19.12	2.02
Solar Ramp Rate Control	0.67	4.79	2.96
Load Imitation	2.51		

Table 5: Fidelity, in terms of MAPE, for different case studies

when imitating the load in the bottom graph. The graph demonstrates that Helios is able to mimic arbitrarily complex load power traces to within a MAPE of 2.5%, as given in Table 5.

6 RELATED WORK

Helios is a platform that enables rapid and fine-grained control of solar power output. In effect, Helios transforms a solar module into a network-connected IoT device. As we discuss, there has been significant prior research on algorithms for controlling solar output, many of which we implement in the previous section using Helios [8, 11–14]. However, since there is currently no platform that enables programmatic solar control, existing algorithms have only been evaluated in simulation and never been implemented. In enabling control of solar power, Helios enables solar to be used as a demand response resource, similar to other thermostatic loads, such as air conditioners and heaters, which are currently the focus of demand response programs. However, unlike these loads, altering solar power has no impact on either user comfort or food temperature. As we show, since solar modules have no moving parts, Helios enables much finer-grained and rapid power control compared to thermostatic loads with mechanical parts. Recent work on smart inverters is most closely related to Helios. However, as discussed in §2, smart inverters focus on specific operating modes that provide grid power quality support. While these modes implement useful algorithms, they are not programmable. Instead, Helios is a programmable DC power optimizer that operates separate from the inverter, and focuses on enabling control of real power, which has numerous applications, as we show in the previous section.

7 CONCLUSIONS

This paper presents the design and implementation of a "smart" software-defined solar module, called Helios, which enables software control of solar power output. While recent work has proposed various algorithms to control solar power to optimize grid operations, there is currently no way to implement them. Helios addresses the problem by exposing a high-level programmatic interface to a DC-DC power optimizer, which enables higher-level software to remotely control solar power output in real-time. We evaluate Helios in the lab (using a solar array simulator) and outdoors, and show that i) it enables precise and rapid solar control and ii) its simple API supports multiple solar control algorithms proposed in recent work. Our work effectively transforms solar from an intermittent power source into one that is highly controllable.

Acknowledgements. This research is supported by NSF grants CNS-1645952, IIP-1534080, CNS-1405826, CNS-1253063, CNS-1505422, and the Massachusetts Department of Energy Resources.

REFERENCES

[1] 2018. IEEE Standard for Interconnection and Interoperability of Distributed Energy Resources with Associated Electric Power Systems Interfaces. http://standards.ieee.org/findstds/standard/1547-2018.html. (June 2018).
[2] S. Barker, S. Kalra, D. Irwin, and P. Shenoy. 2013. Empirical Characterization and Modeling of Electrical Loads in Smart Homes. In *IGSC*.
[3] N. Bashir, D. Irwin, P. Shenoy, and J. Taneja. 2017. Enforcing Fair Grid Energy Access for Controllable Distributed Solar Capacity. In *ACM BuildSys*.
[4] N. Bashir, D. Irwin, P. Shenoy, and J. Taneja. 2018. Mechanisms and Policies for Controlling Distributed Solar Capacity. In *ACM TOSN*.
[5] D. Cardwell. 2015. New York Times, Solar Power Battle Puts Hawaii at Forefront of Worldwide Changes. (April 18th 2015).
[6] D. Chen, D. Irwin, P. Shenoy, and J. Albrecht. 2014. Combined Heat and Privacy: Preventing Occupancy Detection from Smart Meters. In *IEEE PerCom*.
[7] R. Fares. 2016. Scientific American, The Price of Solar Is Declining to Unprecedented Lows. (August 27th 2016).
[8] S. Lee, S. Iyengar, D. Irwin, and P. Shenoy. 2017. Distributed Rate Control for Smart Solar-powered Systems. In *ACM e-Energy*.
[9] S. McLaughlin, P. McDaniel, and W. Aiello. 2011. Protecting Consumer Privacy from Electric Load Monitoring. In *ACM CCS*.
[10] A. Mulkern. 2013. Scientific American, A Solar Boom So Successful, It's Been Halted. (December 20th 2013).
[11] A. Reinhardt, D. Egarter, G. Konstantinou, and D. Reinhardt. 2015. Worried about privacy? Let your PV converter cover your electricity consumption fingerprints. In *IEEE SmartGridComm*.
[12] S. Rongali, T. Ganuy, M. Padmanabhan, V. Arya, S. Kalyanaraman, and M. Petra. 2016. iPlug: Decentralised Dispatch of Distributed Generation. In *IEEE COMSNETS*.
[13] A. Singh, S. Lee, D. Irwin, and P. Shenoy. 2017. SunShade: Enabling Software-defined Solar-powered Systems. In *ACM ICCPS*.
[14] H. Tafti, A. Maswood, G. Konstantinou, J. Pou, and F. Blaabjerg. 2018. A General Constant Power Generation Algorithm for Photovoltaic Systems. *IEEE Transactions on Power Electronics* 33, 5 (May 2018).
[15] H. Tafti, A. Maswood, J. Pou, G. Konstantinou, and V. Agelidis. 2016. An Algorithm for Reduction of Extracted Power from Photovoltaic Strings in Grid-tied Photovoltaic Power Plants during Voltage Sags. In *IEEE IECON*.
[16] A. Vaughan. 2017. Solar Power Growth Leaps by 50% Worldwide Thanks to US and China. (March 7th 2017).