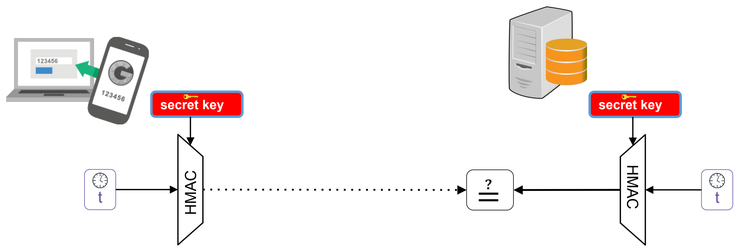
T/KEY: SECOND-FACTOR AUTHENTICATION FROM SECURE HASH CHAINS

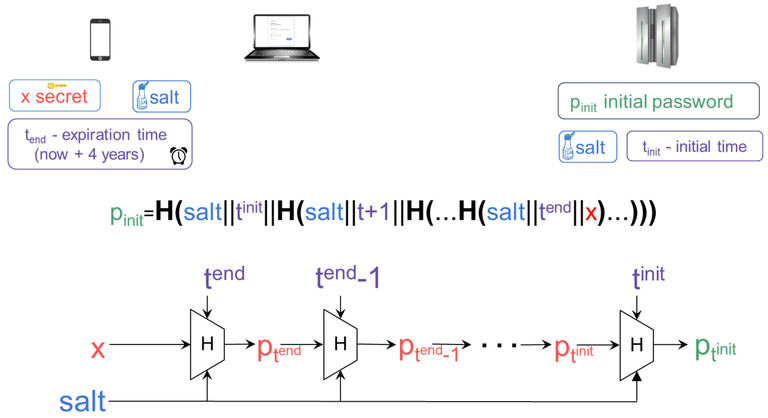
Dmitry Kogan (Stanford), Nathan Manohar (UCLA), Dan Boneh (Stanford)

T/KEY
A drop-in replacement for TOTP that **stores no secrets on the server**

THE PROBLEM

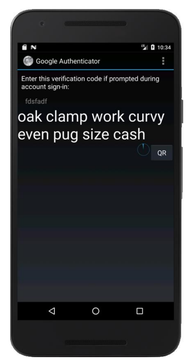
Time-Based One-Time Passwords (TOTP) rely on symmetric keys and are therefore susceptible to server side hacks





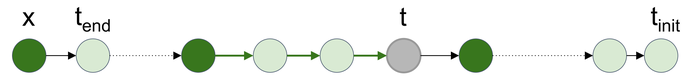
IMPLEMENTATION

We extended Google Authenticator with T/Key
Passwords are encoded as 8 short dictionary words or as QR codes



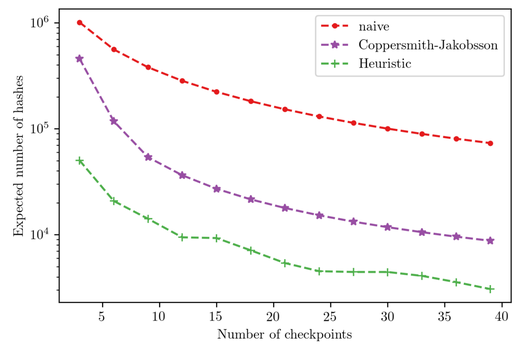
OPTIMIZING OTP-GENERATION TIME

Generating an OTP requires traversing a long hash chain
→Directly translates to login latency
Approach: store some precomputed checkpoints based on the distribution of login intervals



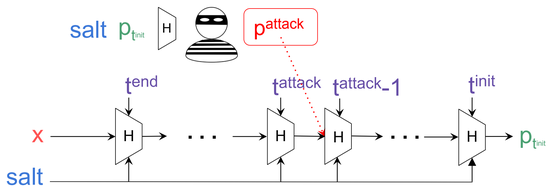
$d(t)$ - probability distribution of login intervals
Find checkpoint positions c1,...,cq to minimize:

$$\mathbb{E}[cost] = \sum_{i=0}^{q-1} \sum_{t=c_i+1}^{c_{i+1}} (c_{i+1} - t)d(t) = 0$$



SECURITY

We prove security in the Random Oracle Model



$$\Pr[A \text{ wins}] \leq \frac{2T + 2k + 1}{2^n}.$$

OUR THEORETICAL CONTRIBUTIONS

- Give a new security analysis of hash chains (iterated hash functions) against preprocessing attacks
- An optimization of hash-chain traversal for resource-constrained devices

REFERENCES

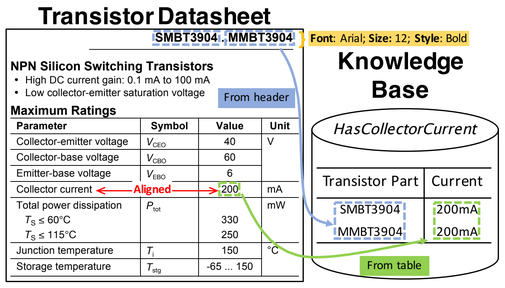
[CJ03] Coppersmith and Jakobsson. Almost Optimal Hash Sequence Traversal. Financial Cryptography.
[MMPR11] M'Raihi, Machani, Pei, and Rydell. TOTP: Time-Based OneTime Password Algorithm. RFC 6238.
[Lam81] Password Authentication with Insecure Communication. Comm ACM.

Fonduer: Knowledge Base Construction from Richly Formatted Data

Sen Wu, Luke Hsiao, Xiao Cheng, Braden Hancock, Theodoros Rekatsinas*, Philip Levis, Christopher Ré
{senwu, lwhsiao, xiao, bradenjh, pal, chrismre}@cs.stanford.edu *thodrek@wisc.edu
Stanford University *University of Wisconsin-Madison

Introduction and Background

Fonduer is a machine-learning based knowledge base construction (KBC) framework for richly formatted data, where entity relations and attributes are conveyed via structural, tabular, visual, and textual expressions.

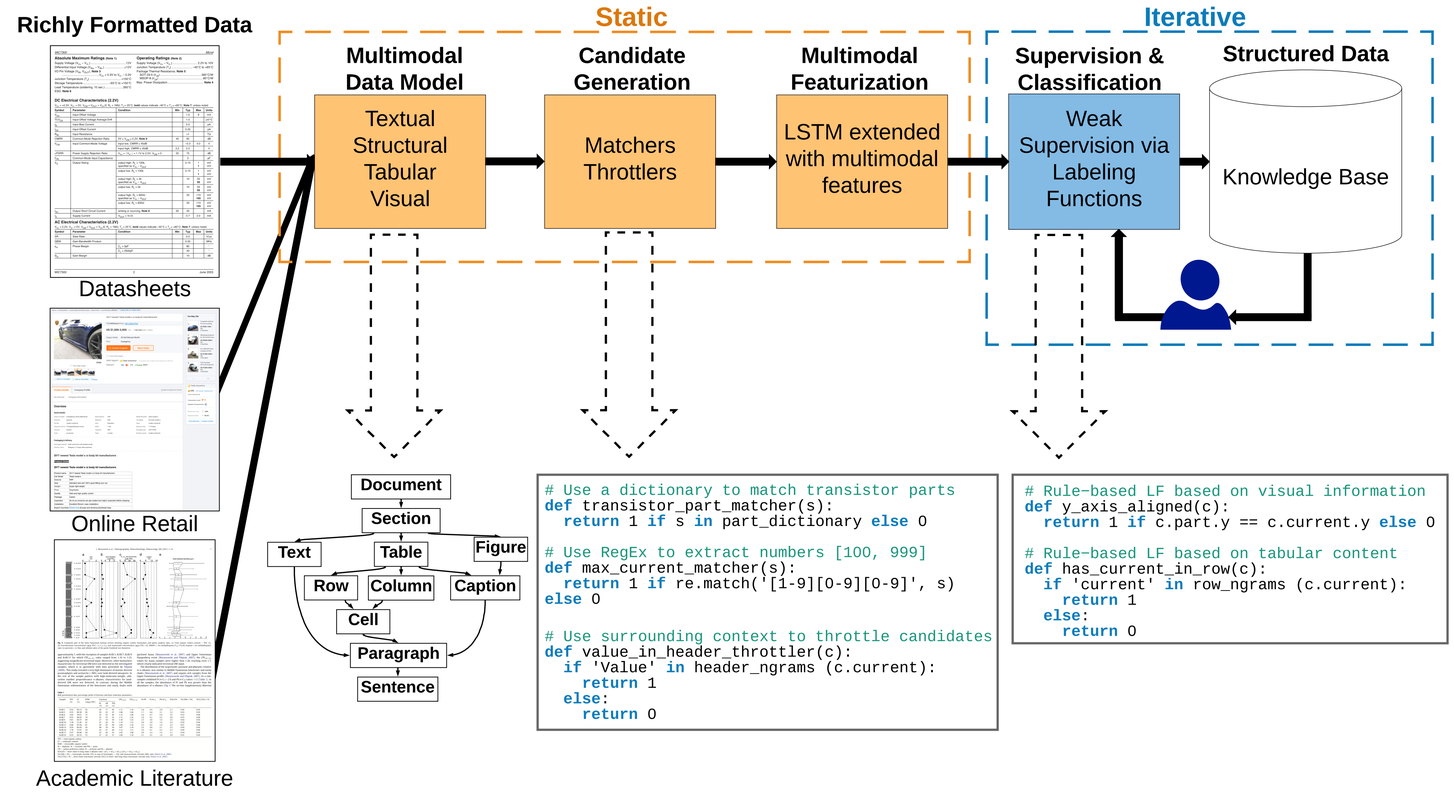


Challenges of KBC from Richly Formatted Data:

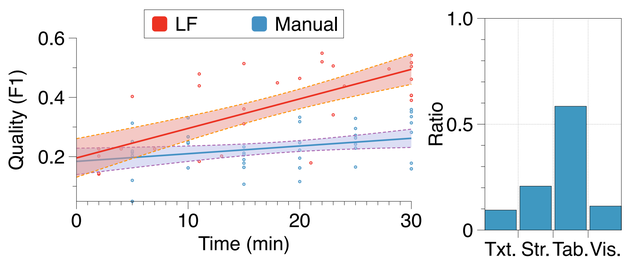
- **Prevalent Document-level Relations**: For richly formatted data, many relations rely on information from throughout the entire document to be extracted.
- **Multimodality**: Semantics are expressed as multiple modalities—textual, structural, tabular, and visual.
- **Data Variety**: The same information can be presented in many different formats and styles, in addition to linguistic variations.

Knowledge Base Construction Using Fonduer

Input: Richly formatted documents (e.g. PDF/HTML/XML/etc.) → **Output**: Structured knowledge base



User Study



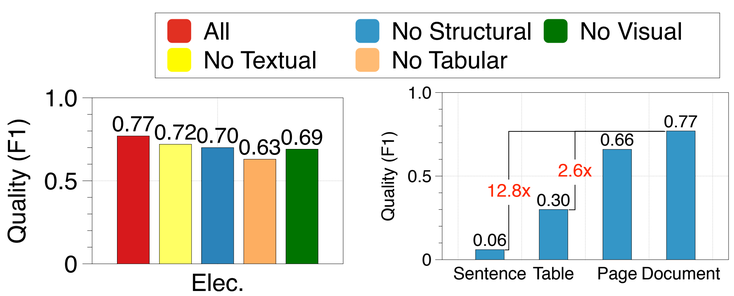
- Users relied $9\times$ more on non-textual signals than textual information alone to identify candidates and provide weak supervision.
- Leveraging multimodal supervision allowed users to create knowledge bases more effectively than traditional manual annotations alone.

Experimental Results

End-to-end Quality vs. Public Data

System	Elec.	Gen.	
Knowledge Base	Digi-Key	GWAS Central	GWAS Catalog
# Entries in KB	376	3,008	4,023
# Entries in Fonduer	447	6,420	6,420
Coverage	0.99	0.82	0.80
Accuracy	0.87	0.87	0.89
# New Correct Entries	17	3,154	2,486
Increase in Correct Entries	1.05×	1.87×	1.42×

Ablation Studies



Our Users

DARPA MEMEX · Alibaba Group · accenture · Macrostrat Lab · Stanford MEDICINE · District Attorney New York County · SING Stanford Information Networks Group

References

Blog: hazyresearch.github.io/snorkel/blog/fonduer

Paper: arxiv.org/abs/1703.05028

Code: github.com/HazyResearch/fonduer

ObliDB: An Efficient and Secure Cloud Database using Hardware Enclaves

Saba Eskandarian and Matei Zaharia
Stanford

Introduction

Motivation:

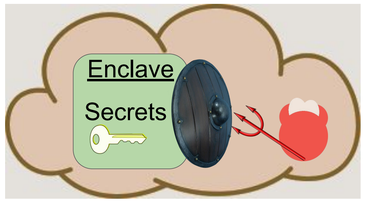
- Databases are a critical component in many applications
- Significant interest in outsourcing them securely

Our Work

- ObliDB: a secure SQL database for the cloud
- Supports both transactional and analytics workloads
- Protects against access pattern leakage

Trusted Hardware

- Trusted hardware *enclaves* provide small protected memory
- Contents hidden even from attackers with full control of the OS
- *Attestation*: Enclave proves it is running desired code
- We implement our enclaves using Intel SGX
- Enclaves still suffer from access pattern leaks



Threat Model

Powers of Adversary:

- Full control of OS
- Examine and modify unprotected memory
- Monitor network and data transfer in/out of enclave

Assumption:

Portion of enclave protected from side-channel attacks, does not leak access patterns inside protected memory

See full version of paper online at
https://arxiv.org/abs/1710.00458

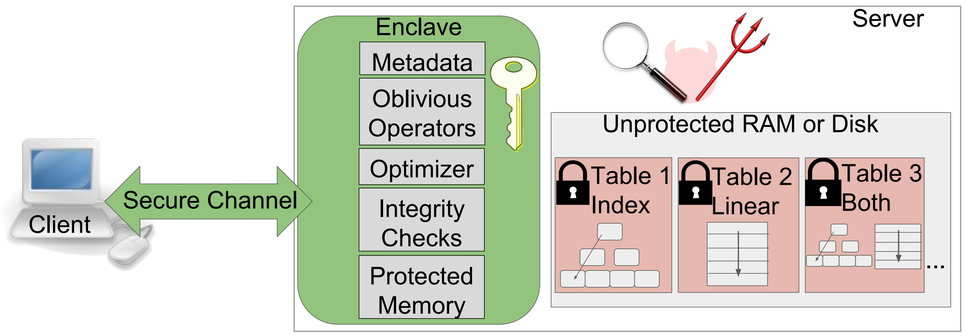
System Overview

Security Guarantees:

- Recognizes and reports attempts to tamper with data
- Leaks only query selectivity, table sizes (intermediate tables included), and query plans
- Optional padding mode hides all table sizes, leaking only query plans

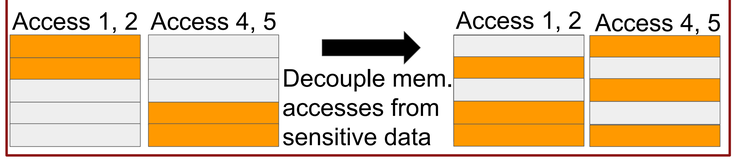
Design Overview:

- Tables stored encrypted in unprotected memory, enclave only holds metadata
- Two oblivious storage methods: linear scans and *oblivious indexes*
- Supports most SQL operations: SELECT (MAX, MIN, AVG, SUM, COUNT), INSERT, UPDATE, DELETE, GROUP BY, JOIN
- Various algorithms for SELECT - can pick best option at runtime



Leakage Attacks

Leakage attacks observe *access patterns* to encrypted memory
Problem: this leakage completely compromises security
Solution: design enclave operation to be oblivious of input data



Achieving Obliviousness

Two Storage Methods:

1. Linear Storage: access every block for every access - used for analytics
2. Oblivious Indexes: used for point queries and analytics on frequently updated tables

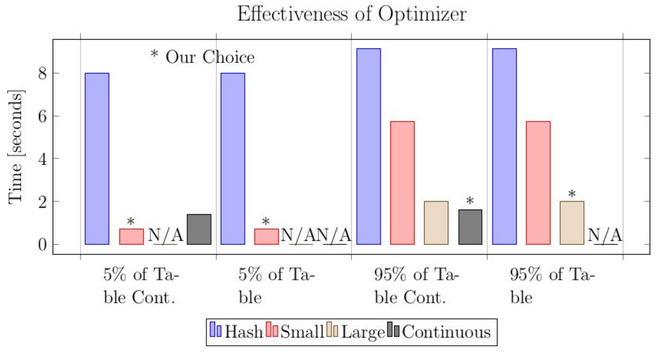
Linear Storage		Oblivious Indexes	
Point Read:	O(N)	Point Read:	$O(\log^2 N)$
Large Read:	O(N)	Large Read:	O(N)
Insertion:	O(1)	Insertion:	$O(\log^2 N)$
Deletion:	O(N)	Deletion:	$O(\log^2 N)$

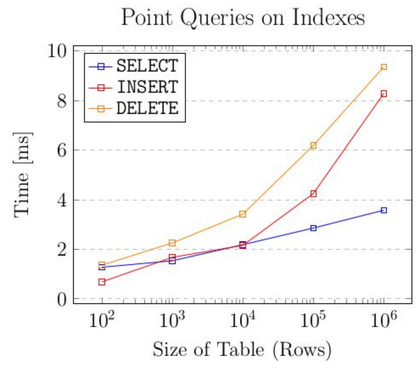
Evaluation

Performance Highlights:

- Up to 329x speedup over naive ORAM-based solution
- Comparable to 19x faster than prior work designed only for analytics
- Comes within 2.6x of Spark SQL (single node) for analytics
- Point query latency of 1-10ms
- Point queries 7-22x faster than prior work without trusted hardware enclaves

Optimizer picks the best option of our 4 oblivious SELECT algorithms (named Hash, Small, Large, and Continuous) based on the results of a fast first pass over data.





Point queries faster than typical network latencies

Comparison to Baseline	
Query Type	Speedup
Range Selection (Linear)	29.2x
Group By Aggregate (Linear)	185x
Range Selection (Index)	1.4x
Point Selection (Index)	1.5x
Insert (Index)	64x
Delete (Index)	15x

Comparison to a baseline database using a naive data structures and operators with ORAM

Enabling Safe, Low Power Networking

Jean-Luc Watson, Paul Crews, Conor McAvity, Hudson Ayers

Professor Philip Levis, Secure IoT Retreat 2018

Motivation

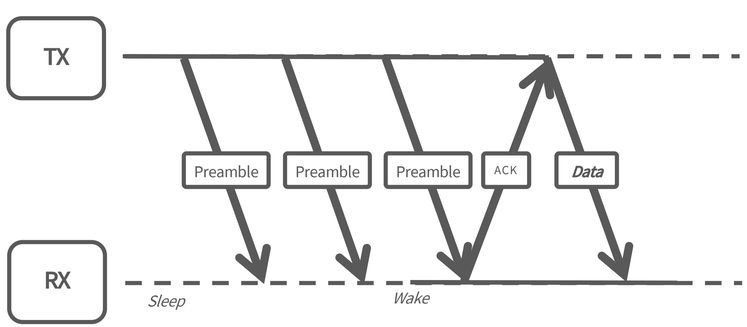
Embedded devices in an *Internet* of things require connectivity as a basic resource, yet network access poses significant security risks. **Tock**, an embedded operating system written in the Rust programming language, can help address these issues. A network stack written as a Tock capsule is memory- and type-safe: a faulty execution that can access memory it was not explicitly granted must have first subverted the Rust type system.

Building on prior work in the IP/6LoWPAN layers, we focus on 2 specific objectives:

- **Energy efficiency**, in that the power consumption of radio communication significantly impacts longevity.
- Straightforward **application networking**, which enables IP communication for multiple untrusted processes on the same platform.

X-MAC

X-MAC is a duty-cycling MAC protocol designed for a radio link layer, proposed by *Buettner et al.* A transmitter sends a sequence of extremely small "preamble" packets, and each node wakes periodically to listen for incoming traffic. Once the receiver has indicated it is awake, the transmitter will send the desired data.



Implementation

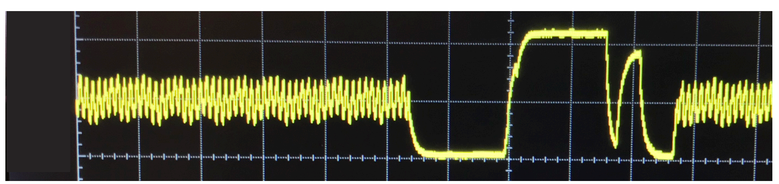
X-MAC is implemented as a Rust capsule and directly controls an 802.15.4 radio. The protocol is tested with an RF233 radio on an Imix board, and can support any hardware device implementing the Tock Radio HIL.

Observations

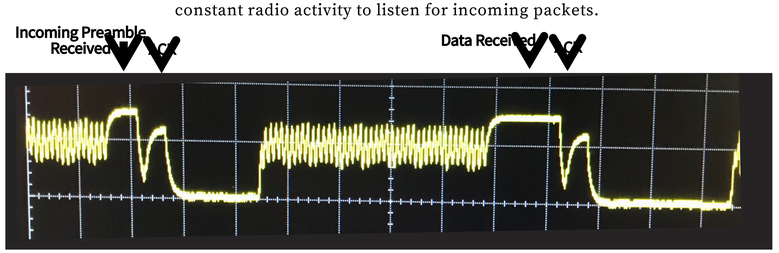
- Must retain full control of radio power management and have some knowledge of radio sleep behavior.
- Designing for the future: general purpose MAC protocol interface.
- Nodes incur significant tradeoff: additional latency.

Power Consumption Behavior

X-MAC Sender: current readings while sending a successfully ACK'd preamble, followed by data packet transmission. Scale on both axes is consistent between each figure.



Non-sleeping Sender: no preamble transmission necessary, but requires constant radio activity to listen for incoming packets.



X-MAC Receiver: preamble packet detected during wake period – ACK and wait to receive data packet.

UDP in User space

The most efficient low power MAC protocol isn't useful if it can't be used. C processes in Tock have long been able to communicate directly with other devices in a WLAN using the built-in 15.4 radio driver, but a connection to the Internet from a board running Tock is impossible. Our current work aims to bring familiar socket-based IP networking to Tock applications.

Current API

```
int udp_socket (sock_handle_t *h, sock_addr_t *a);
int udp_close (sock_handle_t *h);
ssize_t udp_send_to (sock_handle_t *h, void *buf, size_t len,
                     sock_addr_t *dest);
ssize_t udp_recv_from (sock_handle_t *h, void *buf, ssize_t len,
                       sock_addr_t *from);
```

Challenges

- Tock has no file system! Sockets aren't tied to a file descriptor, so the kernel must track registered sockets differently.
- Memory must scale with the number of processes holding socket handles. The kernel must store handle-specific information in individual application *grants*.
- Network stack sends one IPv6 packet at a time.
- Filtering received packets must occur at the driver and must consider the future addition of access control and security policies.

Design Considerations for Low-Power Internet Protocols

Hudson Ayers
Professor Philip Levis, SITP Retreat 2018

Abstract

The 6LoWPAN Internet Standard opens sensor networks up to Internet connectivity by specifying how to format IPv6 packets over low-power wireless links such as 802.15.4. Examining 6LoWPAN implementations in major embedded and sensor networking operating system, however, we observe that they do not fully interoperate. I.e., **for any pair of implementations, one implementation sends 6lowpan packets which the other fails to process and receive.**

We explore why these different implementations do not interoperate and find it is due to some of the basic design goals of 6LoWPAN. Based on these findings, we propose four principles that can be used to structure protocols for low power devices that encourage interoperability between diverse implementations. These principles are based around the importance of balancing memory usage and radio efficiency, and the importance of not relying on Postel's law when dealing with low resource devices. We evaluate and demonstrate these principles by using them to suggest changes to 6LoWPAN that would make it easier for implementations to interoperate.

6LoWPAN Interoperability Study

- 6LoWPAN RFCs define many "features" which 6LoWPAN devices must support.
- Inspecting several open source 6LoWPAN implementations, we discovered mismatched/incomplete support for these features.

Feature	Stack				
	Contiki	OpenThread	Riot	Arm Mbed	TinyOS
Uncompressed IPv6	✓		✓	✓	✓
6LoWPAN Fragmentation	✓	✓	✓	✓	✓
1280 byte packets	✓	✓	✓	✓	✓
Dispatch_IPHC header prefix	✓	✓	✓	✓	✓
IPv6 Stateless Address Compression	✓	✓	✓	✓	✓
Stateless multicast address compression	✓	✓	✓	✓	✓
802.15.4 16 bit short address support		✓	✓	✓	✓
IPv6 Address Autoconfiguration	✓	✓	✓	✓	✓
IPv6 Stateful (Context Based) Address Compression	✓	✓	✓	✓	✓
Stateful multicast address compression		✓	✓	✓	
IPv6 Traffic Class and Flow label compression	✓	✓	✓	✓	✓
IPv6 NH Compression: Ipv6 (tunneled IPv6)		✓		✓	✓
IPv6 NH Compression: UDP	✓	✓	✓	✓	✓
UDP port compression	✓	✓	✓	✓	✓
UDP checksum elision					✓
Compression + headers past first first fragment			✓	✓	
Compression of IPv6 Extension Headers		~[2]		✓	✓
Mesh Header		✓		✓	~[3]
Broadcast Header					✓
Regular IPv6 ND	✓		✓	✓	~[4]
RFC 6775 6LoWPAN ND			✓	✓	
RFC 7400 Generic Header Compression Support					

~ = Partial Support

Related Work

- 6LoWPAN specification (RFC 4944, RFC 6282, RFC 6775)
- Tock OS 6LoWPAN Stack
- Contiki OS, Riot OS, TinyOS, OpenThread, ARM 6LoWPAN stacks

IoT Platform Resources

IoT Platform	Program Memory (kB)	RAM (kB)
Tmote Sky	48	10
Zolertia Z1	92	8
Atmel RZRaven	128	8
TI CC2650	128	28
SAMR21 Xpro	256	32
Nordic nRF52 DK	512	64
Arduino Due	512	96
Nest Protect*	750+	100

- Internet of Things space is composed of devices with varied capabilities.
- Code size and RAM can vary by an order of magnitude
- Application code size and RAM requirements vary as well

Code Size / Radio Energy Tradeoff

- Techniques such as advanced MAC and physical layers, and tracking network state can reduce packet overhead and, thus, radio energy consumption.
- These techniques require larger and more complex implementations.
- Too much emphasis on saving radio energy through complex techniques → force requirement for more expensive, power hungry microcontrollers.
- Different applications and platforms will be better served by different positions of this "slider"
- Specifications should not force any particular balance between code size and radio energy.
- **The table below reveals that much of the complexity of 6LoWPAN is associated with complex compression techniques used to reduce radio energy**

Stack	Code Size Measurements (Bytes)				
	Full IP stack	6LoWPAN-All	Compression	Fragmentation	Mesh/Broadcast Headers
Contiki	37538	11262	5952	3319	N/A
OpenThread	42262	26375	4146-20000	1310	4500
Riot	30942	7500	>4712	1514	N/A
Arm Mbed	46030	22061	17900	3104	1331
TinyOS	37312	16174	– –	– –	600

* TinyOS inlines compression code into fragmentation code, and does not completely implement the mesh header

Design Principles

1. **Capability Spectrum:** A protocol should support a spectrum of device capabilities. This spectrum defines a clear ordering via which especially resource constrained devices can reduce code size or RAM use by eliding features. Such a spectrum makes a protocol usable by extremely low resource devices without forcing more resourceful devices to communicate inefficiently.
2. **Capability Negotiation:** There should be an explicit mechanism by which two devices can efficiently negotiate what level to use when they communicate If two devices wish to communicate, they default to the lower of their supported capability levels.
3. **Provide Reasonable Bounds:** Specifications should specify reasonable bounds on recursive or variable features so implementations can bound RAM use. This allows implementations to safely limit their RAM use without silent interoperability failures.
4. **Don't Break Layering:** Energy-saving optimizations should not make assumptions about the rest of the stack despite the appeal of cross-layer optimization in embedded systems. Long-lived IoT systems will evolve and change, and systems use and draw on existing operating systems as well as libraries. Enforcing layering ensures developers need not own and customize the entire software stack.

WiFröst

Bridging the Information Gap for Debugging IoT Embedded Systems

Will McGrath, Jeremy Warner, Björn Hartmann, et al.

Introduction

The rise in prevalence of IoT technologies has encouraged more people to prototype and build custom internet connected devices based on low power microcontrollers. While well-developed tools exist for debugging network communication for desktop and web applications, it can be difficult for developers of networked embedded systems to figure out why their network code is failing due to the limited output affordances of embedded devices. WiFröst is a tool for debugging these systems using instrumentation that spans from the device itself, to its communication API, to the wireless router and back-end server. WiFröst automatically collects this data, displays it in a web-based visualization, and highlights likely issues with an extensible suite of checks based on analysis of recorded execution traces.

Motivation

We believe that providing the information relevant to understand the behavior of a networked embedded system in a single linked environment can allow users to debug more efficiently through holistic methods (e.g. pattern recognition) and also help preemptively identify problem areas in system behavior across domain boundaries. WiFröst combines information from different domains (from code to network packets and server events) in a joint visualization that allows users to traverse domain boundaries as they seek to understand problems. A key insight is that instrumentation of the network gateway device provides a good deal of both control flow information as well as specific error information (e.g. through remote API call returns).

System Overview

We instrument the code of a microcontroller under test with an ANTLR parser and configure a Rapberry Pi linux computer as a software router that logs communications with mitmproxy. Data from the device and router are captured with an off-the-shelf logic analyzer. We also optionally run an open-source IoT backend that we have modified to log and transmit API accesses. We aggregate these streams in a local database and display them in a visualization.

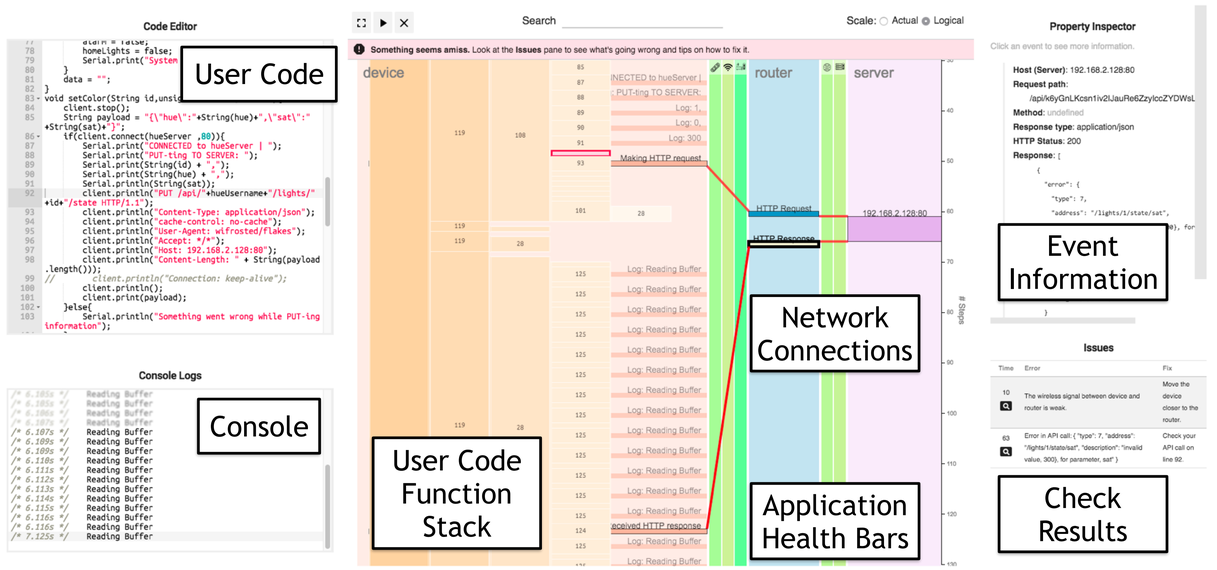


Figure 1: An overview of WiFröst's UI: an integrated code editor, serial console, and application network trace visualization that includes the device's program and communication activity.

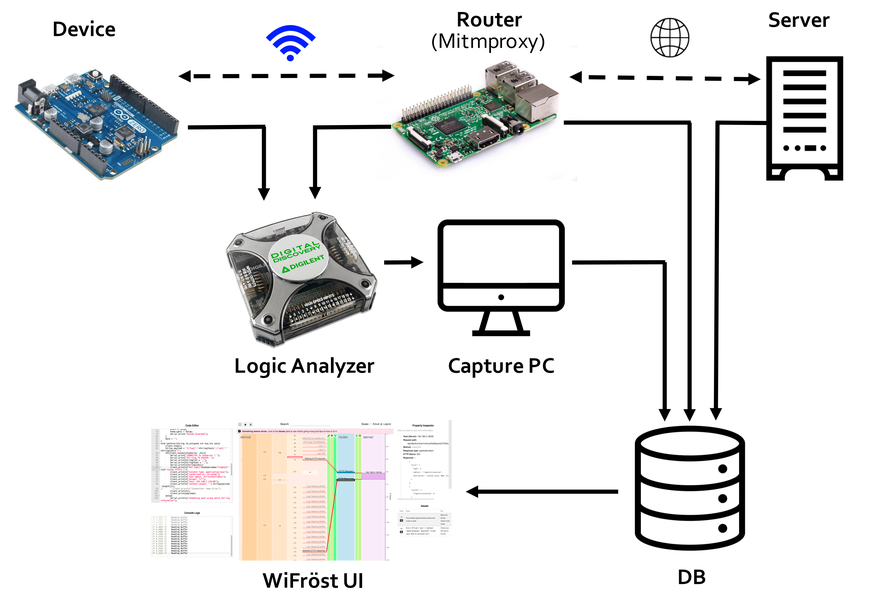


Figure 2: An overview of the hardware components of WiFröst's architecture.

Checks

WiFröst helps users locate the causes of unexpected application behavior by detecting and flagging unexpected behaviors from the trace logs and offering concrete tips on how to fix these behaviors. The checks have access to all of the information collected by the system including a user's code execution, the communication health, and network activity.

	Location of Error:					
		API()				
Device out of memory	√					
WiFi chip connection broken		√				
Incorrect WiFi SSID		√				
Incorrect WiFi password		√		√		
Bad host name		√				
Weak connection to router			√			
Buggy web API usage					√	
Can't access Internet					√	
Incorrect authentication					√	
Server CPU load high						√
Source of information:		API()				

Figure 3: WiFröst's currently supported checks and where they collect their information.

WiFröst GUI

WiFröst focuses on the common pattern of IoT devices communicating with HTTP(S) REST interfaces. WiFröst provides a unified visualization and exploration environment for measurements taken through instrumentation of IoT stacks (device, router, server), so that users can localize errors and identify failure modes based on pattern recognition and situational context across levels of application and communication infrastructure. WiFröst preemptively checks for common errors and as provides in-situ explanations for error types at different levels (e.g., WiFi connection errors, HTTP error codes, API usage errors) in order to decrease the knowledge required for both localizing and interpreting bugs.

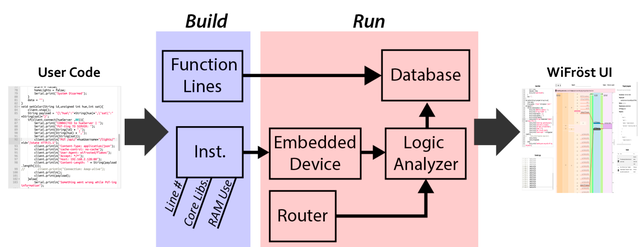


Figure 4: WiFröst's compilation and instrumentation process.

Experiences with Using Bluetooth-Based Tools for Internet Connectivity in the Web of Things

Thomas Zachariah and Prabal Dutta <tzachari,prabal@berkeley.edu>

Secure Internet of Things Project - June 2018

Gateway on a Diet

As more low power sensors and actuators are deployed, the need for an affordable and ubiquitous gateway solution grows more pressing.

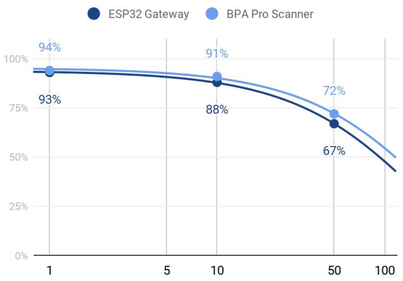
New cheap and efficient components present promising possibilities in creating hardware infrastructure that connect such devices to the Internet while maintaining a miniscule footprint in terms of physical space and energy.

In particular, we look at the **ESP32**, a $4 microcontroller component from Espressif with integrated Wi-Fi and Bluetooth Low Energy (BLE) radios [1].

The provided software environment and drivers for the ESP32 are based on FreeRTOS, and contains built-in libraries for BLE and Wi-Fi.

Espressif ESP32 SoC

Sparkfun ESP32 Thing



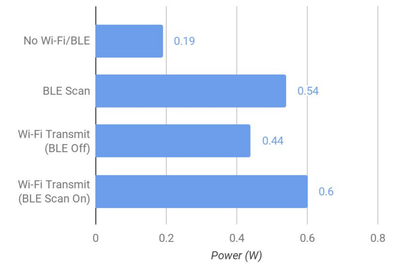
BLUETOOTH PACKET RECEPTION

To test the ESP32's ability to receiving BLE data, we ran a series of 10-min scans in an isolated environment, with no external BLE interference.

We compared with results of a $1000 professional Teledyne BPA scanner.

With devices sending a unique packets every 100ms, the ESP32 achieved packet reception rates of:

- ~93% PRR with 1 advertising devices (vs ~94% on BPA)
- ~88% PRR with 10 advertising devices (vs ~91% on BPA)
- ~67% PRR with 50 advertising devices (vs ~72% on BPA)



POWER CONSUMPTION

To test power consumption, we created a simple Gateway application that performs a BLE scan and sends raw data via HTTP request over Wi-Fi. Power was recorded in each state:

- No Wi-Fi/Bluetooth: ~.19W
- BLE scanning: ~.54W
- Data send over Wi-Fi (while BLE off): ~.44W
- Data send over Wi-Fi (while BLE scan on): ~.6W

These preliminary results indicate that the BLE performance of the ESP32 is comparable to professional BLE scanning equipment, & proves promising as a major gateway component.

Initial experience with the ESP32 is encouraging & we plan to test it for large scale deployments.

Our applications for the ESP32 are available on GitHub [2].

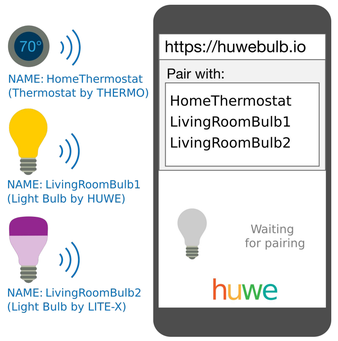
[1] ESP32 Overview | Espressif Systems. 2018. https://www.espressif.com/en/products/hardware/esp32/overview
[2] ESP32-Apps. Lab11. 2017. https://github.com/lab11/esp32-apps/

Bluetooth as a Web Standard

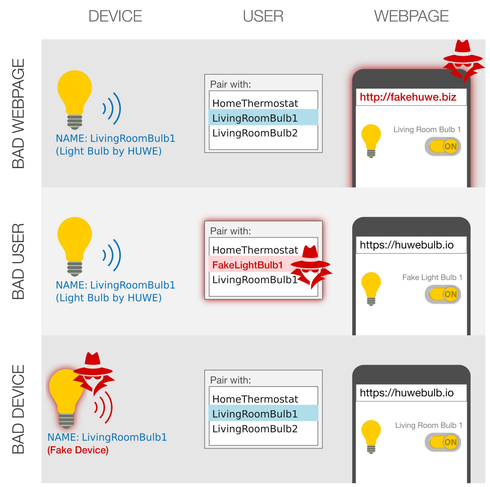
WEB BLUETOOTH is a new W3 standard & JavaScript API [3], enabling connection with BLE devices from websites, implemented in Chrome on Android & desktop.

The current spec requires users to have prior knowledge of the device and its associated website prior to manually navigating to that particular site.

Once the page is open, the user chooses the appropriate device to connect to the page from a list of nearby peripherals, emulating legacy Bluetooth pairing.



Web Bluetooth Model



Potential Threats with Web Bluetooth

VULNERABILITIES WITH WEB BLUETOOTH

This model places the burden of accuracy & authentication on users, disrupts discovery & interaction, and exposes risks from malicious or careless actors:

- ***Malicious webpages*** may spoof pages for real devices, tricking users into pairing with them. Careless pages may allow unknown devices to connect.
- ***Malicious users*** may purposefully initiate an inappropriate pairing of device and webpage, or careless and uninformed users may do so accidentally.
- ***Malicious devices*** can simply use the same name or service ID as a real device and easily trick users into pairing it with legitimate webpages.

The current model is too restrictive for usability, and too permissive for security.

INTEGRATING WITH DISCOVERY SERVICES

The usage model can improve by allowing more casual discovery of devices & interfaces. This can be enabled with a service like **PHYSICAL WEB** [4] — devices broadcast a link that points the browser to a Web Bluetooth webpage.

However, Physical Web obfuscates device information, and the user must guess the appropriate device to connect to the page based on device name.

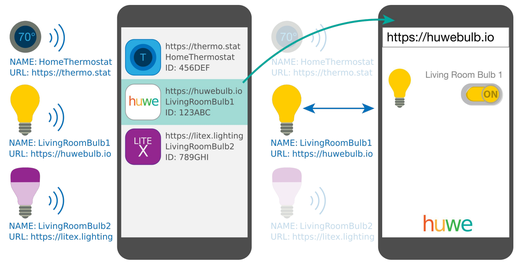
THE DEVICE AS A WEB RESOURCE

To further improve both seamlessness and safety, we can treat the Bluetooth devices as resources of the websites themselves, by allowing them to declare the sites to which they belong or from which they can be accessed.

We have previously introduced a browser, Summon, which employs a form of this strategy to enable seamless discovery & interaction [5].

In this model, the devices broadcast their website's location for discovery, and the browser transparently lists devices with links to their websites.

When opened, the page automatically obtains rights to access only the devices that have declared their association to the site.



Seamless Discovery & Interaction Model

[3] Web Bluetooth Draft Community Report. Web Bluetooth W3C Community Group. 2018. https://webbluetoothcg.github.io/ web-bluetooth/
[4] The Physical Web. Google. 2017. https://google.github.io/ physical- web/
[5] Summon. Lab11. 2017. https://github.com/lab11/summon/

EMBEDDED SYSTEMS RESEARCH
UNIVERSITY OF CALIFORNIA, BERKELEY
lab11.eecs.berkeley.edu

Berkeley | EECS
Electrical Engineering and Computer Sciences

Pantheon: the training ground for Internet congestion-control research

https://pantheon.stanford.edu

Francis Y. Yan[†], Jestin Ma[†], Greg D. Hill[†], Deepti Raghavan[¶], Riad S. Wahby[†], Philip Levis[†], Keith Winstein[†]

[†]Stanford University, [¶]Massachusetts Institute of Technology

Introduction

- congestion control is a cornerstone problem
- no community benchmarks
- inconsistent behaviors of congestion-control algorithms

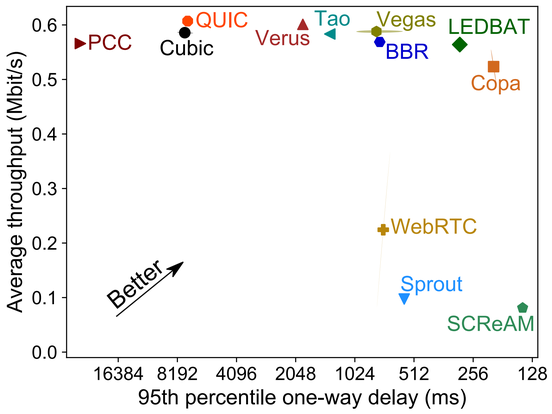


Figure 1: Many algorithms perform differently from how they were intended and documented. Colombia to AWS Brazil (cellular, 1 flow, 3 trials, P1391).

Pantheon: a benchmark platform for congestion control

- a software library containing 15+ congestion-control algorithms
- a diverse testbed in 10+ countries on wireless and wired networks
- a collection of *calibrated emulators* and pathological emulated networks
- a continuous-testing system
- a public archive of searchable results at https://pantheon.stanford.edu

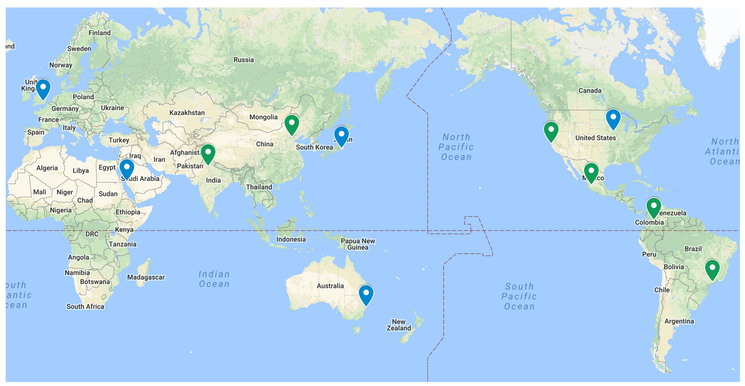


Figure 2: Pantheon's measurement nodes (green pins: wired and cellular; blue pins: wired only).

Measurement study:

- performance of congestion-control algorithms is highly variable across the type of network path, bottleneck network and time
- no single existing algorithm performs well in all settings

Calibrated emulators

- simulation or emulation: reproducible and allows raplid experimentation
- traditional view: the more fine-grained and detailed, the better
- open problem: how to choose parameter values to faithfully emulate a network

New figure of merit: replication error

Average difference of the performance of a set of transport algorithms run over the emulator compared with over the target real network path.

Approach:

- collect a set of results over a particular network path on Pantheon
- run Bayesian optimization to minimize replication error
- parameter space: link rate, propagation delay, sender's queue size, loss rate, constant or Poisson-governed rate

Evaluation:

Path	Replication error (%)
Nepal to AWS India (Wi-Fi, 1 flow, P188)	19.1
AWS Brazil to Colombia (cellular, 1 flow, P339)	13.0
Mexico to AWS California (cellular, 1 flow, P196)	25.1
AWS Korea to China (wired, 1 flow, P361)	17.7
India to AWS India (wired, 1 flow, P251)	15.6
AWS California to Mexico (wired, 1 flow, P353)	12.7
AWS California to Mexico (wired, 3 flows, P1237)	14.4

Figure 3: Replication error of calibrated emulators on real-world paths. 16.8% on average.

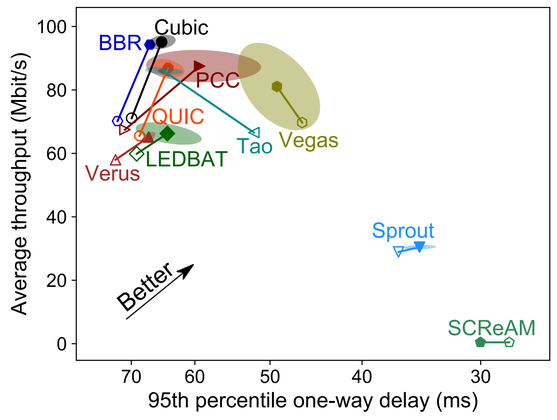


Figure 4: Calibrated emulators are able to match the performance of transport algorithms. AWS California to Mexico (wired, 3 flows, 10 trials, P1237). Mean replication error: 14.4%.

Pantheon use cases

- Vivace (NSDI 2018): contributed 3 variants to Pantheon
- Copa (NSDI 2018): deployed a series of 6 prototypes and used Pantheon's measurements to inform each iteration
- Indigo: a new congestion-control design trained on Pantheon's data

Indigo: a machine-learned congestion control

- learns to map from *states* to *actions* using a recurrent neural network
- *state*: congestion window size, previous action, and exponentially-weighted moving average (EWMA) of queueing delay, sending rate, and receiving rate
- *action*: adjustment to congestion window ($\div 2, -10, +0, +10, \times 2$)
- imitation learning: imitates actions labelled by a *congestion-control oracle*
- *congestion-control oracle*: outputs the action that brings the current congestion window closest to the ideal size—around bandwidth-delay product (BDP) in emulators

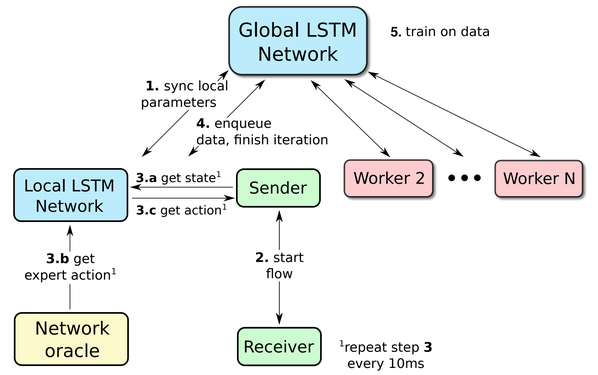


Figure 5: Schematic of Indigo's distributed training system.

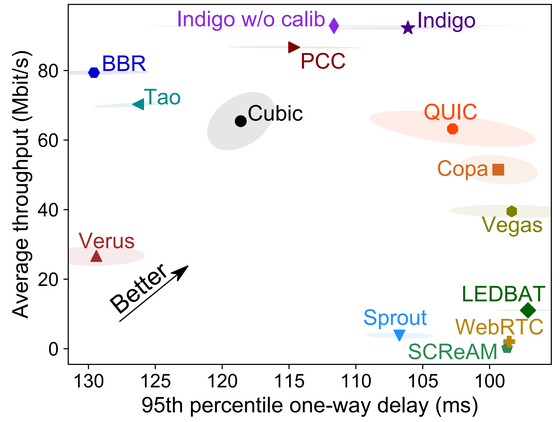


Figure 6: Indigo's performance is at the throughput/delay tradeoff frontier. AWS Brazil to Colombia (wired, 1 flow, 10 trials, P1439).

Conclusion

Pantheon has assisted in the development of two recently-published congestion-control algorithms, and has supported our own data-driven approach to protocol design. Motivated by the success of ImageNet in the computer-vision community, we believe Pantheon will enable faster innovation and more reproducible research.

Power clocks: Dynamic Multi-Clock Management for Embedded Systems

Holly Chiang, Daniel Giffin, Amit Levy, Philip Levis

Introduction

Modern microcontrollers can be reconfigured on the fly with different clock sources and frequencies, and these may differ radically in their power consumption. For applications whose workloads change over time, dynamic management of clocks is critical to conserving energy. While this task can be integrated into application logic, the varying constraints of each embedded hardware environment together with the complex interactions of multi-application systems can make this approach unacceptably burdensome. Power Clocks orchestrates clock management in the kernel to optimize energy consumption, thus obviating the need for application involvement and still achieving acceptable performance for typical workloads. In our example sensor application, Power Clocks is able to demonstrate up to 15.2% in energy savings over a statically chosen energy-optimizing clock and up to in 31.2% energy savings over a static general purpose clock.

Clock	Frequency	Current	Startup
RCSYS	113600 Hz	12 µA	38 µs
RC1M	1 MHz	35 µA	-
RCFAST	4.3/8.2/12 MHz	90/130/180 µA	0.31 µs
OSC0	16 MHz	-	-
RC80M	80 MHz	300 µA	1.72 µs
PLL	48-240 MHz	120-500 µA	30 µs
DFLL	20-150 MHz	122-1919 µA	100 µs

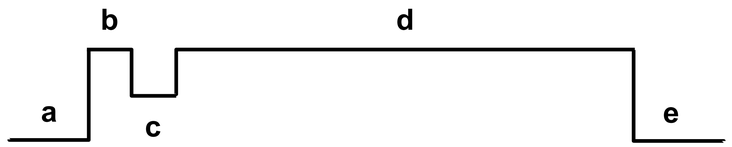
System Design

What information is needed to change the clock?

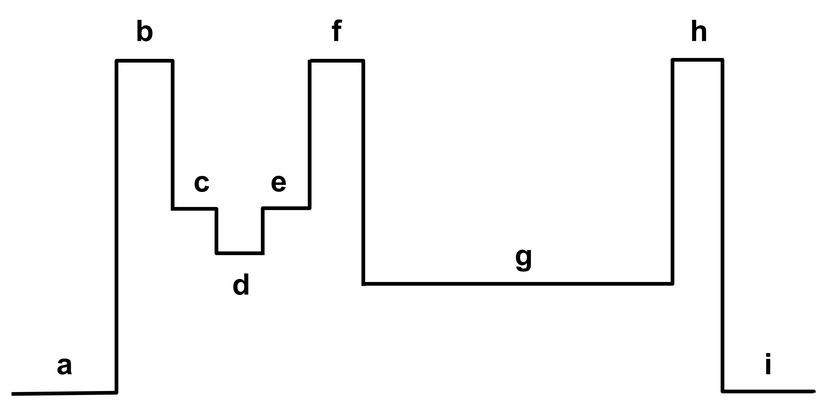
Modern microcontrollers can have dozens of peripherals, so requiring the controller to keep track of the state of each can be too resource-intensive. Instead, it makes more sense for each peripheral to keep track of its own state and notify the manager when changes occur. Rather than having the controller figure out what clocks can be used for a given peripheral state, each peripheral can directly report a list of clocks compatible with its current state.

When can the clock be changed?

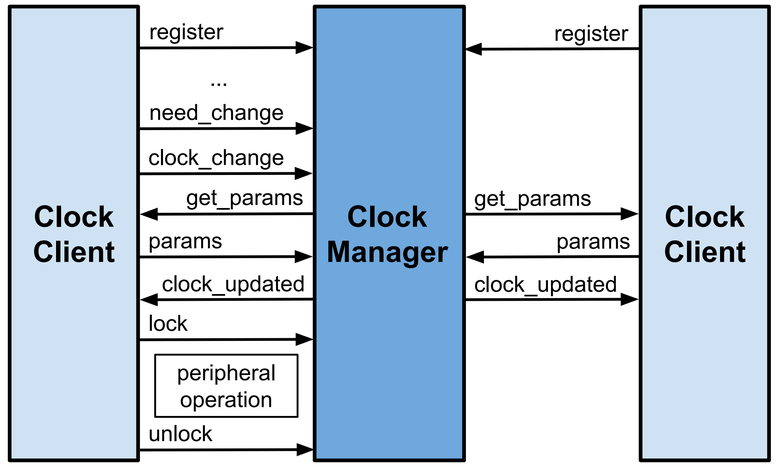
Although immediately servicing a clock change request would allow better real-time guarantees, correctness requires that some peripheral operations be allowed to complete first. In order to determine when such operations are in progress, the manager could periodically scan peripheral devices. But this can be expensive if there are many peripherals, and the scan has to be repeated multiple times. It is more efficient for peripherals to report to the manager when their clock-sensitive operations begin and end. Another issue is preventing blocking caused by new peripheral operations starting while the system is waiting for a clock change to occur. Both of these goals can be solved by using a locking mechanism.



Label	Mode	Clock	Description
a	wait	RC32K	The processor is in deep sleep
b	run	RCFAST	The processor wakes up, the ADC requests a clock change
c	sleep	RCFAST	The processor goes to sleep
d	run	RCFAST	The ADC's DMA transfer completes, ADC samples are converted, flash is written
e	wait	RC32K	The processor goes to deep sleep



Label	Mode	Clock	Description
a	wait	RC32K	The processor is in deep sleep
b	run	DFLL	The processor wakes up
c	run	RCFAST	The ADC starts sampling with DMA transfer
d	sleep	RCFAST	The processor goes to sleep
e	run	RCFAST	The ADC's DMA transfer completes
f	run	DFLL	ADC samples are converted
g	run	RCSYS	The flash is written
h	run	DFLL	The processor switches to its default clock
i	wait	RC32K	The processor goes to deep sleep



Implementation

ClockManager and **ClockClient** are Rust traits. The central clock controller implements ClockManager. Every peripheral that has clock dependent operations implements ClockClient.

During initialization, each peripheral calls **register**, allowing the ClockManager to store a reference to itself.

Before beginning a operation that needs the clock, a peripheral calls **need_change** to check if the current clock is compatible with the its clock requirements. If the current clock is not compatible with the peripheral's needs, the peripheral calls **clock_change** to request a change in clock. To choose a clock that meets the requirements of all peripherals, the ClockManager queries all registered peripherals for their clock requirements by calling **get_params**. The ClockManager then chooses the lowest power clock that meets all requirements and informs each registered ClockClient that the system clock has been changed by calling the **clock_updated** function on each ClockClient.

If a ClockClient has a clock dependent operation, it calls **lock** before it begins its operation. Calling lock prevents the ClockManager from changing the clock while the peripheral's operation is ongoing. lock returns a boolean indicating whether or not the peripheral obtained a lock. It will always return true unless a clock change is pending, during which locking is prevented. Once a peripheral's operation finishes, the peripheral calls **unlock** . If the unlock causes the lock count to drop to zero, a clock_change is triggered.

ADC sample rate	Static DFLL (µJ)	Static RCFAST (µJ)	Hand Code (µJ)	Power Clocks (µJ)
125 ksps	719.4	574.2	491.7	528.0
250 ksps	907.5	739.2	633.6	650.1
300ksps	973.5	801.9	693.0	679.8

Stanford Information Networks Group (SING)

Computer Science Department
353 Serra Mall
Stanford University
Stanford, CA 94305-9025, USA

Examining PCB Design Practices

setting the foundation for better board design tools

R. Lin, R. Ramesh, A. Iannopollo, A. Sangiovanni-Vincentelli, P. Dutta, B. Hartmann

work in progress

Background

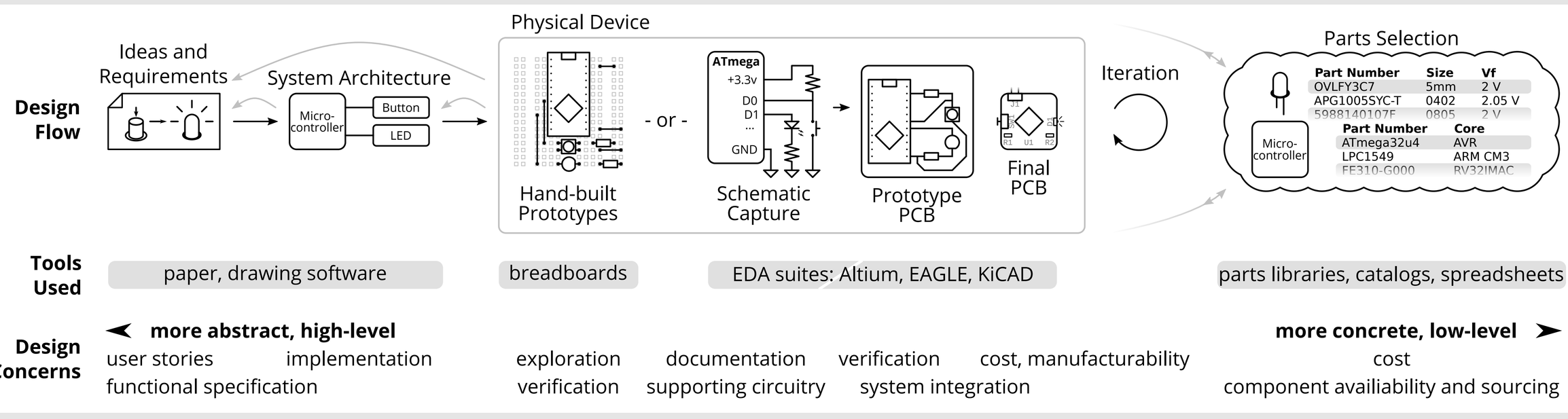
Why do we care?
- Electronics are everywhere, e.g. for interactivity or IoT
- PCBs needed for nontrivial projects, to work with small (non-breadboardable) parts, or for mechanical stability

The study
- Interviewed 15 people with PCB design experience, from beginning novice to intermediate professional
- Asked about design flows as well as details of each step

Modern Practice
- Current design paradigm (graphical schematic capture followed by place and route) **established in the 1980s**

What's new?
- **Designs are different**: self-contained, easy-to-use parts
- **Designers are different**: lower barrier to entry (Arduino) enables creative and non-professional designers
- Orders of magnitude **more compute**: smarter tools possible

Observed Design Flows



Overall Observations
- Overall, a strategy of **iterative refinement**
  - Start with high-level, abstract designs
  - Refine with system diagrams, prototype iterations

High Level Design
- Functional requirements: ideas, interfaces, specifications
- If written down: generally a living, evolving document

System Architecture (Block Diagrams)
- Maps requirements down to hardware
- **Mixed levels of abstraction**: blocks may be generic parts (e.g. "microcontroller"), or may specify implementations
- Paper popular for freedom, but digital has other advantages

Prototyping
- Breadboards, protoboards, non-form-factor PCBs
- Verify design works before committing

Board Design (Schematic Capture, Place and Route)
- Schematic **mostly data entry**, 'creative' work done outside
- Manually copy-paste (transcribe) part datasheet circuits
- ERC sometimes used to catch limited classes of errors

Parts Selection
- Happens throughout the design process, at different levels
- Considerations: function, cost, ease-of-use

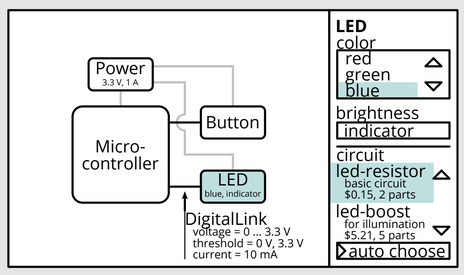
Assembly and Test
- Visual inspections for assembly defects
- Electrical bring-up: connectivity check, power-up smoke test

Concepts for New Tools

Significant oppurtunity for improved schematic entry

Goals
- Reduce schematic design time by:
  - Allow higher-level design input, design space exploration
  - Encourage re-use of circuit blocks
  - Better automated checks, reduce manual verification effort

Underlying Data Model
- Block diagram model is widespread but powerful
  - Support building reusable generators with parameterization examples: LED color, operating voltage, maximum ratings
  - Type system on blocks and ports, allowing auto refinement



◄ Block Diagram GUI
- Users reluctant to use new tools must provide **familiar interfaces**
- Hierarchical design abstraction allowing underconstrained design

```
class IndicatorLed(SubcircuitPart):
  def __init__(s, color=None):
    io = s.Port(DigitalSink())
    led = DiscreteLed(
      color=color, i=10*mA)
    res = DiscreteRes(res=
      (io.pos.v-led.vf-io.neg.v)/10mA)

    io.pos << led.pos
    led.neg << res.a
    res.b << io.neg
```

◄ Hardware Construction EDSL
- EDSLs are highly efficient for advanced users
- Allows **building generators:** methodologies not instances

Berkeley | EECS
Electrical Engineering and Computer Sciences

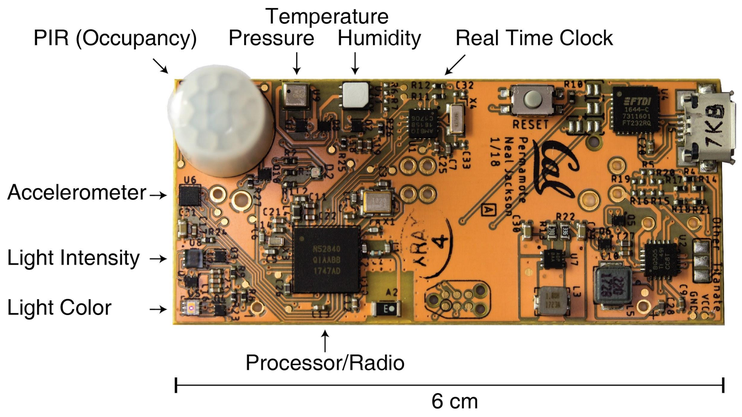
For further information, contact:
richard.lin@berkeley.edu
rkr@berkeley.edu

Permamote: A Long-Lifetime Sensor Platform for a Reliable Internet of Things

Neal Jackson, Joshua Adkins, and Prabal Dutta <neal.jackson, adkins, prabal>@berkeley.edu

SITP - JUNE 2018

Permamote combines energy-harvesting, a backing non-rechargeable battery, and the newest, lowest power components to achieve consistent operation over a greater than 10 year lifetime when performing common sensor mote workloads.



Permamote Features

Built from the lowest power components currently available.

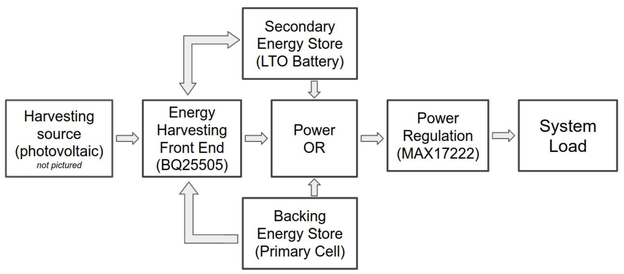
Permamote has an integrated processor and BLE/802.15.4(Thread) radio. It includes a variety of environmental sensors including temperature, pressure, humidity, accelerometer, and light intensity and color.

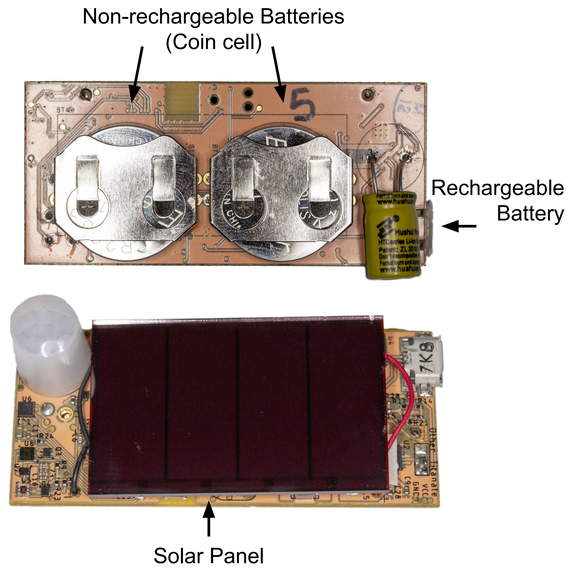
Additionally, an ultra low power (50nA) real time clock allows the platform to keep independent, accurate time

Permamote Power Supply

The Permamote power supply architecture consists of an LTO battery that is recharged with a solar panel. When this battery is depleted, it automatically switches over to a couple of non-rechargeable coin cell batteries.

The hardware designs for Permamote are open source. This power supply can be reused for any kind of sensor.

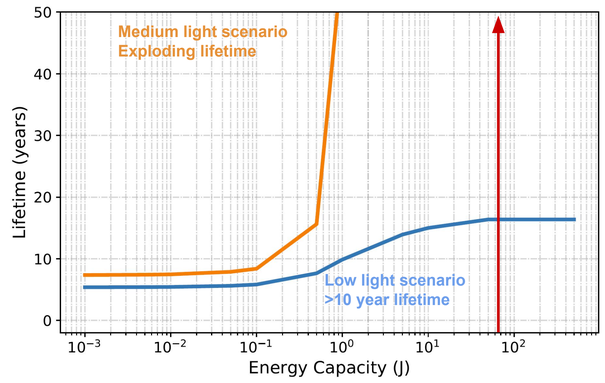




https://github.com/lab11/permamote

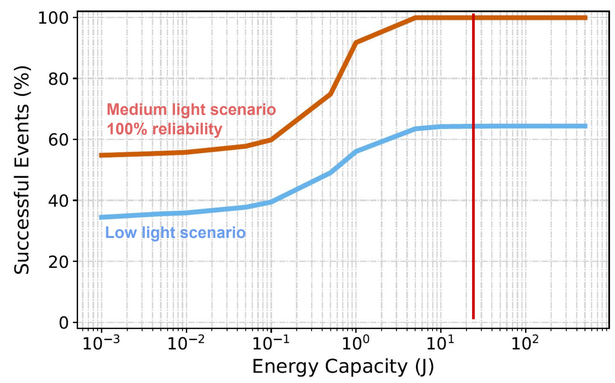
Long Lifetime

With its current storage configuration, Permamote is expected to live **"indefinitely"** in normal light conditions, and greater than a decade in darker conditions.



Reliability

In addition to a long expected lifetime, Permamote is also expected to have **near perfect reliability**, even without backup non-rechargeable energy storage.



Envisioned Applications

- Fine grained occupancy sensing
- Dynamic lighting, color, and shade control
- Building heating and cooling tracking
- Plant water monitoring
- Distributed glare detection
- Solar energy trace generation

Future Work

Explore performing more general computation on devices like Permamote and pushing down tasks like machine learning and DSP to the edge. Additionally, consider dynamic workload adjustment based on available energy and lifetime constraints.

Use Permamote as a testbed for **autonomously localizing** sensor deployments. When deployed, sensors perform both relative and semantic localization, where relative means discovering coordinates in 3D space in relation to other sensors, and semantic means discovering which sensors **share the same surface, desk, wall, or room**. From this we can better construct maps of spaces, as well as automatically discover context about the deployment environment instead of needing to write it down.

EMBEDDED SYSTEMS RESEARCH
UNIVERSITY OF CALIFORNIA, BERKELEY
lab11.eecs.berkeley.edu

Berkeley | EECS
Electrical Engineering and Computer Sciences