OpenBSD Hardware Sensors: Environmental Monitoring and Fan Control

by

Constantine A. Murenin

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of

Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2010

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Constantine A. Murenin

Abstract

This thesis discusses the motivation, origin, history, design guidelines, API, the device drivers and userland utilities of the hardware sensors framework available in OpenBSD. The framework spans multiple utilities in the base system and the ports tree, is utilised by over 75 drivers, and is considered to be a distinctive and ready-to-use feature that sets OpenBSD apart from many other operating systems, and in its root is inseparable from the OpenBSD experience.

The present framework, however, is missing the functionality that would allow the user to interface with the fan-controlling part of the hardware monitors. We therefore discuss the topic of fan control and introduce sysctl-based interfacing with the fan-controlling capabilities of microprocessor system hardware monitors. The discussed prototype implementation reduces the noise and power-consumption characteristics in fans of personal computers, especially of those PCs that are designed from off-the-shelf components. We further argue that our prototype is easier, more intuitive and robust compared to solutions available elsewhere.

Acknowledgements

I would like to thank my supervisor, Professor *Raouf Boutaba,* for his guidance and support in my study at the University of Waterloo. I would also like to thank my readers, Associate Professor *Richard Trefler* and Associate Professor *Martin Karsten,* for their encouragement and comments. Special thanks goes to my friends and colleagues that have made my experience at the university a time to miss and remember.

To my family.

Table of Contents

1. *Introduction* 1

   1.1. *Motivation* 1

   1.2. *Design decisions* 4

2. *Framework API* 6

   2.1. *Adding sensors in attach()* 8

   2.2. *Sensor task refresh procedure* 9

3. *Sensor Tools* 11

   3.1. *sysctl hw.sensors* 11

   3.2. *sensorsd* 12

4. *Sensor Drivers* 14

5. *I^2C Sensors and Bus Scan* 20

   5.1. *Open Firmware and I^2C* 20

   5.2. *I^2C bus scan through i2c_scan.c* 21

   5.3. *I^2C drivers* 22

   5.4. *I^2C register dumps* 22

   5.5. *I^2C sandboxing for driver development* 23

6. *Evolution of the Framework* 26

   6.1. *Framework timeline* 26

6.2. Evolution of drivers 28

7. Related Frameworks 30

7.1. NetBSD envsys / sysmon 30

7.2. FreeBSD general sysctl tree 34

7.3. lm_sensors 35

8. Port to FreeBSD / DragonFly BSD 36

8.1. Summer of Code 2007 36

8.2. Sensors framework in DragonFly 37

8.3. Sensors framework in FreeBSD CVS 37

9. Availability 39

10. Introduction and Motivation for Fan Control 40

11. Related Work on Fan Control 42

11.1. Interfacing from the BIOS 42

11.2. ACPI 43

11.3. SpeedFan on Windows 45

11.4. lm_sensors on Linux 46

12. Hardware Monitoring Chips 47

12.1. Shortcomings with general-purpose fan-control software 48

13. OpenBSD sysctl hw.sensors Fan-Control 49

13.1. New upvalue field and new flags 49

13.2. Sensor types 51
13.3. Dynamic sensor descriptions 52
13.4. The lm(4) driver 52
14. Demonstration 54
15. Conclusion 57
16. Future Projects 58
16.1. Hardware support 58
16.2. Port sensor-detect.pl from lm_sensors 58
16.3. Port i2c_scan.c to other BSDs 58
16.4. Further improve sensorsd 59
16.5. Interfacing for fan-speed controlling 59
16.6. Possible race conditions between user software and the BIOS 60
16.7. Fan-controlling through sensorsd 60
References 63

1. Introduction

In the first part of the thesis, we start by investigating the matter of what hardware monitoring sensors represent, how common is it for them to appear in general-purpose computer hardware that has been available on the market in the last decade or so, and what benefits can be gained by having a unified, simple and straightforward interface for getting data out of these sensors.

In the second part of this work, we investigate the possibilities of controlling the speed of the fans for purposes of reduced acoustics and power-consumption, and discuss the prototype that implements said functionalities.

1.1. Motivation

Although it may come as a surprise to some users, the majority of personal computers that have been available on the market in the last decade have an integrated hardware monitoring circuitry, the main purpose of which is delivering a plethora of functionalities regarding temperature management and environmental conditions. For example, many Super I/O chips that can be found on popular motherboards contain a logical device known as Microprocessor System Hardware Monitor. [lm.4] [it.4] [viasio.4] [nsclpcsio.4] [fins.4] [schsio.4] These hardware monitors can be interfaced through I^2C/SMBus or ISA/LPC, or both, and provide information regarding the rotational speed of cpu and system fans, temperature readings from internal and external sensors, as well as the voltage that is supplied to the motherboard by the power supply unit. As a matter of fact, these hardware monitors can also allow one to control the voltage that is given to the fans that are connected to the motherboard, and have a closed-loop circuitry that, once programmed, can vary the voltage, and thus the speed, of certain fans based on the changes in sensory data.

Another trend that has been particularly common in the recent years is the availability of defined interfaces for software-based temperature readout from individual components of personal computers, such as the CPU, or the add-on cards, such as

those implementing the 802.11 wireless functionality or 10 Gigabit Ethernet. Popular examples include recent Intel Xeon and Core series of processors (as well as budget models that are marketed under different brands) [admtemp.4] [cpu.4]; all AMD64 processors from AMD (Families 0Fh, 10h, 11h) [kate.4] [km.4]; Intel WiFi Link 4965/5100/5300 wireless network devices [iwn.4].

When it comes to high-end server and workstation equipment, then there are even more possibilities for additional sensors, from Intelligent Platform Management Interface (IPMI) [ipmi.4] and Dell Embedded Server Management [esm.4] to SCSI Accessed Fault-Tolerant Enclosure [safte.4] and SCSI Enclosure Services [ses.4]. [Gwynne.Open06]

Certain brand-name laptops feature additional opportunities for hardware monitoring, too. These can include IBM/Lenovo ThinkPad Active Protection System [aps.4] and Apple Sudden Motion Sensor [asms.4], which provide information regarding acceleration in a 2- and 3-D plane, respectively. Newer laptops with Advanced Configuration and Power Interface (ACPI) [acpi.4 et al] may additionally provide information regarding the thermal zones [acpitz.4] and battery status [acpibat.4], as well as a boolean state of whether the power supply is currently connected [acpiac.4].

Some vendors, moreover, include custom ACPI devices in the Differentiated System Description Tables (DSDTs) that specify ACPI methods and operation regions for enquiring certain environmental information from the underlying hardware. Examples feature IBM/Lenovo ThinkPad laptops with IBM0068 ACPI device [acpithinkpad.4], ASUSTeK desktop motherboards that are advertised with features such as AI Booster through the underlying ATK0110 ACPI device [aibs.4] and ABIT uGuru.

Another variant of sensor data that has been found necessary for system administrators to be aware of is that of the status of logical disc drives from the utilisation of the Redundant Array of Inexpensive Discs (RAID) technology. [esm.4] [ami.4]

[ciss.4] [mfi.4] [arc.4] [softraid.4] [cac.4] [mpi.4] [ips.4] This includes information regarding which logical drives may be affected by what kinds of problems (e.g. a state of 'online', 'degraded', 'failed' etc). [Gwynne.Open06]

The latest type of sensors that was introduced in OpenBSD is the timedelta type, which provides data regarding the offset of the local clock versus some kind of a much more reliable timesource (e.g. a GPS source [nmea.4] or a low-frequency radio receiver [udcf.4] [mbg.4] [umbg.4] [gpiodcf.4]). The ntpd(8) daemon uses these timedelta sensors to correct the local clock, with the aim of ensuring that they are as close to zero as possible (i.e. the local clock within the OS is as close as possible to that of some trusted external time source). [Balmer.Asia07] [Balmer.Euro07]

Type	Unit
Temperature	K
Fan speed	RPM
Voltage	V DC, V AC
Resistance	Ω
Power	W
Current	A
Power capacity	W/h, A/h
Indicator	boolean
Raw number	integer
Percentage	%
Illuminance	lx
Logical disc drive	enumeration
Local clock offset	s

Table I. List of sensor types.

As a summary, it has been found that all of these sensor-like data (see Table I for the complete list) can be aggregated in a single and straightforward interface, which we describe in the following sections. Needless to say, the monitoring of many of these

environmental sensors can predict and diagnose system failure. [Gwynne.Open06] We argue that OpenBSD's interface is easier to use and is more effective than comparable interfaces in other systems, since it requires no manual configuration on the part of the user or system administrator (unlike most competing solutions). [deRaadt.zdnet06] Granted, due to the fact that no manual configuration is required, and little user-visible options are available for such configuration, the framework may not fit everyone's needs; however, when one has dozens of distinct machines across the network, one starts appreciating the fact that the framework is so easy to use, and that, for the most part, it comes preconfigured right from the time of the first boot of one's copy of OpenBSD.

Since many modern operating systems still lack unified hardware monitoring frameworks, our presentation of the framework developed in OpenBSD will be helpful for any future endeavours regarding the design and development of any comparable framework and the kinds of devices and features it has to support, and which framework features may be considered optional.

1.2. Design decisions

Following the standard OpenBSD philosophy regarding operating system features, the framework has been designed with the goal of being simple, secure and usable by default, right out-of-the-box. [deRaadt.priv06] We should note that in many cases overengineering would not have been useful anyhow, since many devices have incomplete specifications, and supporting too many extended features at the price of a bloated kernel is judged as not being a positive approach.

To illustrate this example of lacking or incomplete documentation, consider how voltage sensors work. When reading a value from one of the popular hardware monitoring chips (like those from Winbond or ITE Tech), the sensed value represents a voltage in range between 0 and around 2 or 4 V. I.e. some resistors must be in place between a 12 V power line and the sensor input on the chip. Subsequently,

the read value is scaled based on the resistor factors (see Table II for the illustration), where the resistor recommendations are expected to have been supplied by the chip manufacturer to the motherboard manufacturer, where the latter must have adhered to such recommendations when building their products.

Function	Maths	Result
original reading	0xcb	203
sensor voltage	203 * 16 mV	3,24 V
scale for +5 V	3,24 V * 1,68	5,44 V
scale for +12 V	3,24 V * 3,80	12,31 V

Table II. Voltage example for Winbond W83627HF.

In practice, it has been noted that such recommendations may sometimes be missing or contradict each other from much of Winbond documentation, whereas at other times, the motherboard manufacturer might have decided to go with some microsaving by not following certain parts of the recommendation. In turn, all of this results in situations where it is not at all clear what voltage sensor monitors which power line and whether the readings can be trusted; in a sense, voltage doesn't scale, so to speak; thus we have to do the best with what we have. [Murenin.IEEE07]

In essence, design decisions included keeping the framework simple, secure and usable, with minimal user-configuration and sane default settings, since the framework is enabled by default on all installations and must be useful, reliable and stable out-of-the-box.

2. Framework API

The API of the framework with relevant datastructures is defined in /sys/sys/sensors.h. The framework was originally introduced in 2003 and first appeared in OpenBSD 3.4, but was redesigned on several occasions afterwards. As of this writing (March 2010), the userland API has been stabilised in 2006 and has been stable since OpenBSD 4.1, whereas the kernel API and the overall ABI has suffered some minor changes in 2007 with OpenBSD 4.2. Since then, many new and interesting drivers and userland tools have been introduced, but the underlying framework itself has remained stable. This section describes this latest revision of the framework [Murenin.IEEE07] in some detail; details on what major changes were made in 2006 are available in an article in The OpenBSD Journal [Murenin.TOJ06].

The sysctl mechanism is used as a transport layer between the kernel and the userland. [sysctl.3] [sysctl.8] This has an advantage of making the interface rather familiar to many end users, as well as programmers, due to the wide familiarity with sysctl amongst BSD users.

Two main datastructures are used: *struct sensordev* and *struct sensor*. The former holds some information about the sensor device as a whole (relevant MIB element in the sysctl tree, the unix name of the device, the most number of sensors of each type and the actual total number of sensors), whereas the latter holds the information about each individual sensor.

```
struct sensordev {
      int   num;
      char  xname[16];
      int   maxnumt[SENSOR_MAX_TYPES];
      int   sensors_count;
};
```

Each sensor may include an optional 31-character description, an optional time when the value of the sensor was last changed, the actual value of the sensor, the type of

the sensor, an ordinal sensor number within sensors of this type on this device, an optional sensor status and a field for some sensor flags.

```
struct sensor {
        char                    desc[32];
        struct timeval          tv;
        int64_t                 value;
        enum sensor_type        type;
        enum sensor_status      status;
        int                     numt;
        int                     flags;
};
```

Sensor description field should be used wisely: there is absolutely no need to duplicate sensor type in sensor description, nor is there any need to duplicate *numt* in the description; thus descriptions like "Fan1", "Local Temperature 1" and "Local Temperature 2" should be avoided if at all possible, and an empty string "", "Local" and "Local", respectively, should be used in their place.

Sensor state is optional, and should only be used by those drivers that are actually able to query significant amount of state information from the hardware to have the ability to meaningfully change the state from one to another.

```
enum sensor_status {
        SENSOR_S_UNSPEC,
        SENSOR_S_OK,
        SENSOR_S_WARN,
        SENSOR_S_CRIT,
        SENSOR_S_UNKNOWN
};
```

The default state value is *UNSPEC*, which signifies that the state information will never be updated, and thus can be safely ignored by userland utilities such as sysctl(8) and systat(1) in order to avoid providing the user with meaningless information. For example, the majority of the I^2C and Super I/O hardware monitors should not populate the state field, since they have not much certainty in the validity of the readings they acquire. On the other hand, if the driver does know the state of its sensors (for example, as is the case with IPMI [ipmi.4] or ATK0110 [aibs.4]), then those states may be one of *OK*, *WARN*, *CRIT* or *UNKNOWN*.

For the list of sensor types, please refer to Table I, or the source code definitions in /sys/sys/sensors.h.

Since OpenBSD 4.2, the *sensor* and *sensordev* datastructures were renamed to *ksensor* and *ksensordev* in the kernel, and some irrelevant bookkeeping fields were removed from the userland *sensor* and *sensordev* structures; for simplicity, we may continue to refer to these structures from the userland perspective of *sensor* and *sensordev*, even if the kernel structures are the ones being discussed.

2.1. Adding sensors in attach()

Writing drivers that utilise the framework is very straightforward. In the attach procedure of the driver, the first step that can be taken is the initialisation of the *xname* field of *struct sensordev*. Subsequently, each member of the *struct sensor* array should have its *type* field initialised (the only field that requires explicit initialisation), and *sensor_attach()* should be called (which will set the *numt* field appropriately, amongst some other bookkeeping).

Subsequently, *sensor_task_register()* can be used to register the periodic update task.

```
void
drv_attach(struct device *parent,
    struct device *self, void *aux)
{
      ...

      strlcpy(sc->sc_sensordev.xname,
          sc->sc_dev.dv_xname,
          sizeof(sc->sc_sensordev.xname));

      for (i = 0; i < n; i++) {
            sc->sc_sensors[i].type =
                SENSOR_TEMP;
            sensor_attach(&sc->sc_sensordev,
                &sc->sc_sensors[i]);
      }

      if (sensor_task_register(sc,
          drv_refresh, 5) == NULL) {
            printf(": unable to register "
                "update task\n");
            return;
      }

      sensordev_install(&sc->sc_sensordev);

      printf("\n");
}
```

The final action that the driver must perform to make its whole tree of sensors available system-wide is the call to *sensordev_install()*. The driver may abort at any time before calling *sensordev_install()*, but once the call is made, a *sensordev_deinstall()* must be called before the driver can safely detach itself or cancel the attach procedure due to some other error.

2.2. Sensor task refresh procedure

In the refresh procedure of a minimal driver, all that needs to be done is the *value* field of each sensor to be updated.

```
void
drv_refresh(void *arg)
{
      struct drv_softc *sc = arg;
      struct ksensor *s = sc->sc_sensors;
      ...

      for (i = 0; i < n; i++)
            s[i].value = ...;
}
```

Of course, those drivers that keep state or use other fields of the *sensor* structure must update them, too (presumably, during each update cycle).

Our minimal examples are intended to be illustrative of the main ideas behind the framework, where the drivers are not required to update any fields which they do not specifically use, therefore, simplifying the logic of the driver code and the overall complexity of the system (it is in fact known that in general the driver code amounts for around 70% of the kernel code in an operating system, moreover, the driver code is known to have error rates up to 7 times higher than the rest of the kernel [Tanenbaum06] [Chou01]).

3. Sensor Tools

The sensors framework spans multiple user interfaces in OpenBSD. These include **sysctl**(3) *HW_SENSORS* for C/C++ programmes; **sysctl**(8) *hw.sensors* for instantaneous readings or usage from shell scripts; **systat**(1) *sensors* view for semi-realtime sensor monitoring; **sensorsd**(8) for filling in the log files with relevant changes in sensor data, as well as user-configured alerts; **ntpd**(8), which, effectively, acts as a timedelta minimiser; and **snmpd**(8), the SNMP daemon. Some interesting tools are available in the ports tree, too; these include sysutils/**symon** for remote monitoring, and sysutils/**gkrellm** for some GUI monitoring.

3.1. sysctl hw.sensors

The following is a sample output from running `sysctl hw.sensors` on an AMD Phenom X4 9850 box, where one can see the sensor trees from two drivers, km(4), the embedded temperature sensor in the CPU, and it(4), the Hardware Monitor from ITE Tech's Super I/O chip.

```
hw.sensors.km0.temp0=50.50 degC
hw.sensors.it0.temp0=32.00 degC
hw.sensors.it0.temp1=45.00 degC
hw.sensors.it0.temp2=92.00 degC
hw.sensors.it0.fan0=2528 RPM
hw.sensors.it0.volt0=1.34 VDC (VCORE_A)
hw.sensors.it0.volt1=1.92 VDC (VCORE_B)
hw.sensors.it0.volt2=3.42 VDC (+3.3V)
hw.sensors.it0.volt3=5.21 VDC (+5V)
hw.sensors.it0.volt4=12.54 VDC (+12V)
hw.sensors.it0.volt5=1.62 VDC (-5V)
hw.sensors.it0.volt6=4.01 VDC (-12V)
hw.sensors.it0.volt7=5.75 VDC (+5VSB)
hw.sensors.it0.volt8=3.23 VDC (VBAT)
```

The first part of each line (before the equal sign, "=") represents the sysctl MIB for the sensor in question, whereas the second part of each line represents the decoding of the *struct sensor* datastructure by sysctl(8).

The following is an example of running `sysctl hw.sensors.aibs0` on a system with Asus Striker Extreme motherboard. The motherboard's ACPI DSDT includes an ASOC ATK0110 device, which provides not only the ability to gather sensor input, but also describes what each sensor is specifically used for, as well as supplying the information regarding the applicable range or ranges of alert values for each sensor. The ATK0110 device is supported by the aibs(4) driver, which automatically populates the sensor description and sensor state fields of each sensor.

```
hw.sensors.aibs0.temp0=31.00 degC (CPU Temperature), OK
hw.sensors.aibs0.temp1=43.00 degC (MB Temperature), OK
hw.sensors.aibs0.fan0=2490 RPM (CPU FAN Speed), OK
hw.sensors.aibs0.fan1=0 RPM (CHASSIS FAN Speed), WARNING
hw.sensors.aibs0.fan2=0 RPM (OPT1 FAN Speed), WARNING
hw.sensors.aibs0.fan3=0 RPM (OPT2 FAN Speed), WARNING
hw.sensors.aibs0.fan4=0 RPM (OPT3 FAN Speed), WARNING
hw.sensors.aibs0.fan5=0 RPM (OPT4 FAN Speed), WARNING
hw.sensors.aibs0.fan6=0 RPM (OPT5 FAN Speed), WARNING
hw.sensors.aibs0.fan7=0 RPM (PWR FAN Speed), WARNING
hw.sensors.aibs0.volt0=1.26 VDC (Vcore Voltage), OK
hw.sensors.aibs0.volt1=3.25 VDC ( +3.3 Voltage), OK
hw.sensors.aibs0.volt2=4.95 VDC ( +5.0 Voltage), OK
hw.sensors.aibs0.volt3=11.78 VDC (+12.0 Voltage), OK
hw.sensors.aibs0.volt4=1.23 VDC (1.2VHT Voltage), OK
hw.sensors.aibs0.volt5=1.50 VDC (SB CORE Voltage), OK
hw.sensors.aibs0.volt6=1.25 VDC (CPU VTT Voltage), OK
hw.sensors.aibs0.volt7=0.93 VDC (DDR2 TERM Voltage), OK
hw.sensors.aibs0.volt8=1.23 VDC (NB CORE Voltage), OK
hw.sensors.aibs0.volt9=1.87 VDC (MEMORY Voltage), OK
```

For more details regarding the possible output of sysctl(8), see the relevant source code in src/sbin/sysctl/.

3.2. sensorsd

The sensorsd(8) sensor monitoring daemon allows the user to monitor all sensors and send alerts if certain states of the sensors change. [deRaadt.zdnet06]

Since c2k7 (the general OpenBSD hackathon in 2007) and OpenBSD 4.2, sensorsd can automatically monitor and report the changes in sensor states on those sensors that keep their state (for example, as is the case with IPMI, ESM, ACPI ATK0110

and the *drive* and *timedelta* type of sensors). [Biancuzzi.42] Moreover, for any sensor, no matter whether its driver keeps its state or not, the monitoring of manually specified upper and lower boundaries can be performed. When any monitored sensor state changes, the change is logged with syslog(3) and a command, if specified, is executed. For more details, see the source code and documentation of src/usr.sbin/sensorsd/.

We would like to point out an interesting observation in regards to the temperature monitoring that may often be overlooked — it is generally assumed that only the upper temperature limits are of any interest to the users, whereas in reality, monitoring of the lower limit may also be of value in certain hostile environments, where the attacker may cool the memory modules down with liquid nitrogen, and subsequently remove the modules from the system to gain unauthorised access to the data they will continue to contain. [Halderman.Lest] Therefore, regardless of whether the driver automatically keeps the state of its sensors, we advise the users to consider the manual monitoring of the lower limit of any temperature sensor they deem important. This monitoring of the lower limit of all temperature sensors from any driver can be easily accomplished with a sensorsd.conf rule similar to "temp:low=18C".

4. Sensor Drivers

In this section, we provide an overview of the kernel device drivers that utilise the framework. For the most part, the statistics in this section are based on the source code, as opposed to the binaries of the actual kernels for i386/amd64/macppc/sparc64 etc. However, since the majority of the drivers are actually enabled in most GENERIC kernels in OpenBSD, we deem that such a comparison is still appropriate and reasonable.

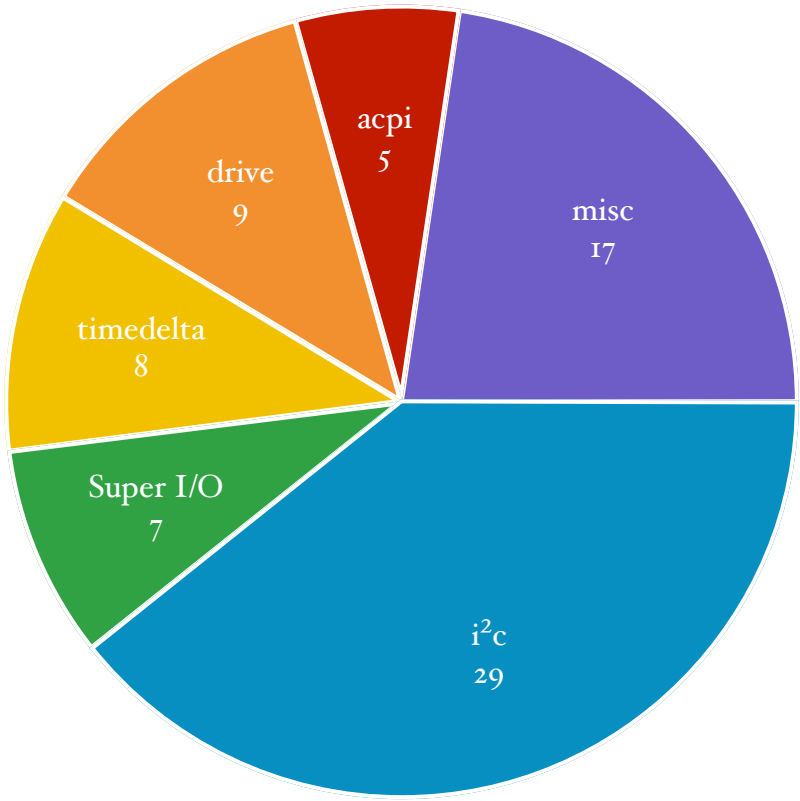

Chart I. Number of sensor device drivers in OpenBSD 4.6 by primary category.

In general, the drivers can be roughly divided into the following categories: Super I/O hardware monitors, I²C/SMBus sensors, embedded temperature sensors, SCSI enclosures and IPMI, ACPI sensors, as well as RAID logical drive status sensors and

time offset sensors. The I²C drivers, by far, form the majority, as is evidenced from Chart I. Note that in this chart some drivers have been placed into the general miscellaneous category for simplicity, which include IPMI and various other embedded sensors. Moreover, the above grouping is not strictly unambiguous — some I²C devices, for example, may actually be part of a Super I/O solution (which would not make much difference from the perspective of a software developer, so we deem our grouping to be reasonable where the Super I/O category does not include any I²C-only devices).

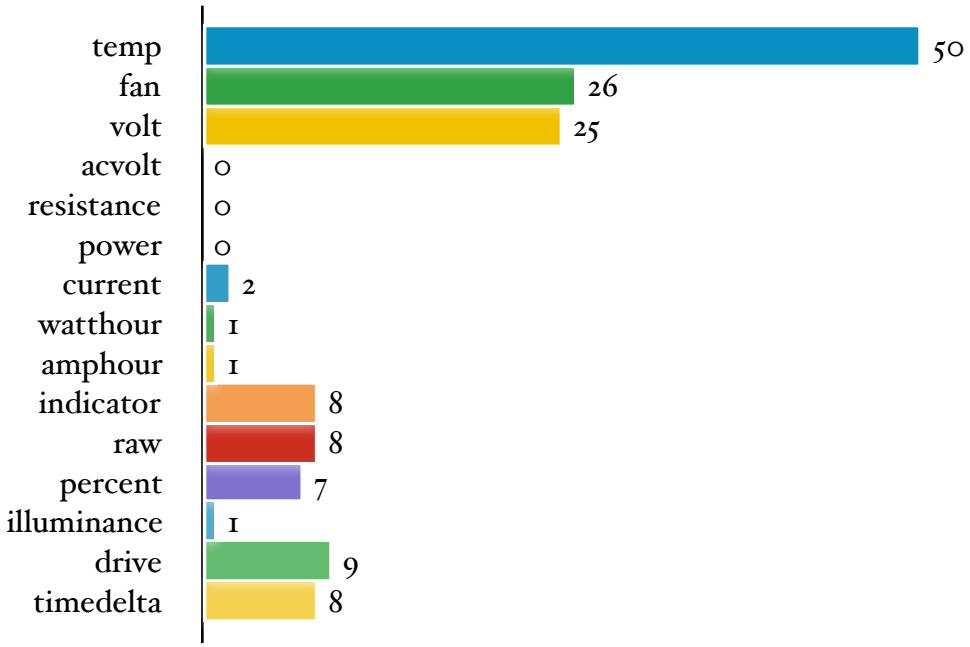

Chart II. Sensor type popularity in OpenBSD 4.6 based on the number of drivers using each type.

OpenBSD 4.6 (July / October 2009) contains 75 drivers that expose sensors using the sensors framework. Chart II represents sensor type popularity based on the number of drivers that are using each type (nonexclusively). Note that the temperature sensor type is by far the most popular (used by 50 out of the 75 drivers), with the fan and

voltage sensors having nearly a draw for the next most popular type (used by 26 and 25 out of the 75 drivers, respectively).

Since OpenBSD 3.8 (2005), the sensors framework has had a notion of sensor status, defined by *enum sensor_status*, which can be one of *UNSPEC*, *OK*, *WARN*, *CRIT* or *UNKNOWN*. It is optional for the drivers to specify the status of each sensor, and specification is intended and recommended only in cases where the drivers are specifically aware of such status (please refer to the section describing the framework API for more details). If the driver does not specifically identify the status of the sensor (which is intended to be the case in the drivers of simple sensor devices that have no information regarding the bounds or appropriate limits of their specific sensing application), then the driver should leave the *status* field of the *sensor* datastructure alone (of course, that is because the datastructure should be zeroed before first use, which is done automatically by the autoconf(9) or malloc(9) with the *M_ZERO* flag, leaving the *status* field at the *UNSPEC* value of the enumeration). Chart III represents the status popularity with the drivers in OpenBSD 4.6, where the implicit *UNSPEC* popularity is computed by subtracting the total number of unique drivers that specify some sensor state — 24, from the total number of sensor drivers in OpenBSD 4.6 — 75.

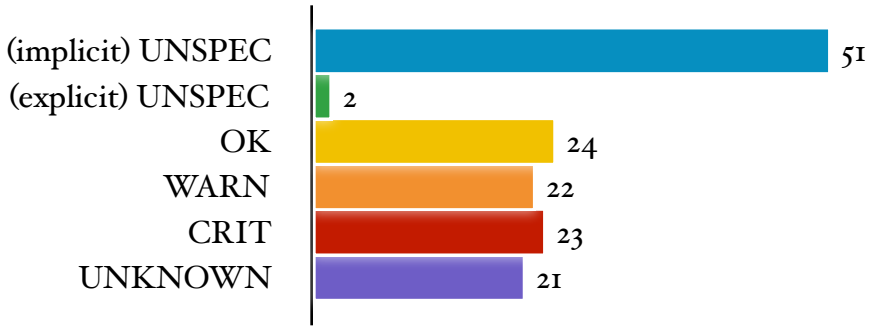

Chart III. Sensor status popularity in OpenBSD 4.6 based on the number of drivers specifying each status.

We are satisfied to note that the drivers appear to adhere to the design principles of the framework, since the popularity between different states is roughly the same (apart from the anticipated variance of the implicit and explicit *UNSPEC*, of course). From these numbers we can also see that only 1/3rd of the drivers utilise sensor status, with 2/3rds having no use of it, which for visual purposes is summarised in the following pie chart — Chart IV.

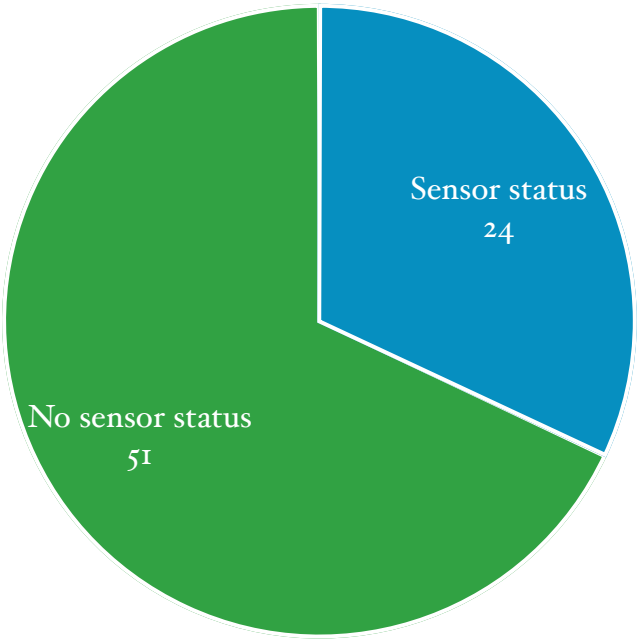

Chart IV. Number of sensor device drivers in OpenBSD 4.6 providing and not providing the sensor status.

It is noteworthy to mention that there are currently no I^2C drivers that utilise sensor status. Chart V provides information on sensor type popularity between the drivers that have a notion of sensor status, and Chart VI includes a comparison of type popularity based on the presence or absence of support for sensor status (essentially summarising Charts II and V). (For consistency and verification purposes, we

have populated the 'no status' counts of type popularity based on the source-code analysis, even though it could have been determined from the other two fields alone.)

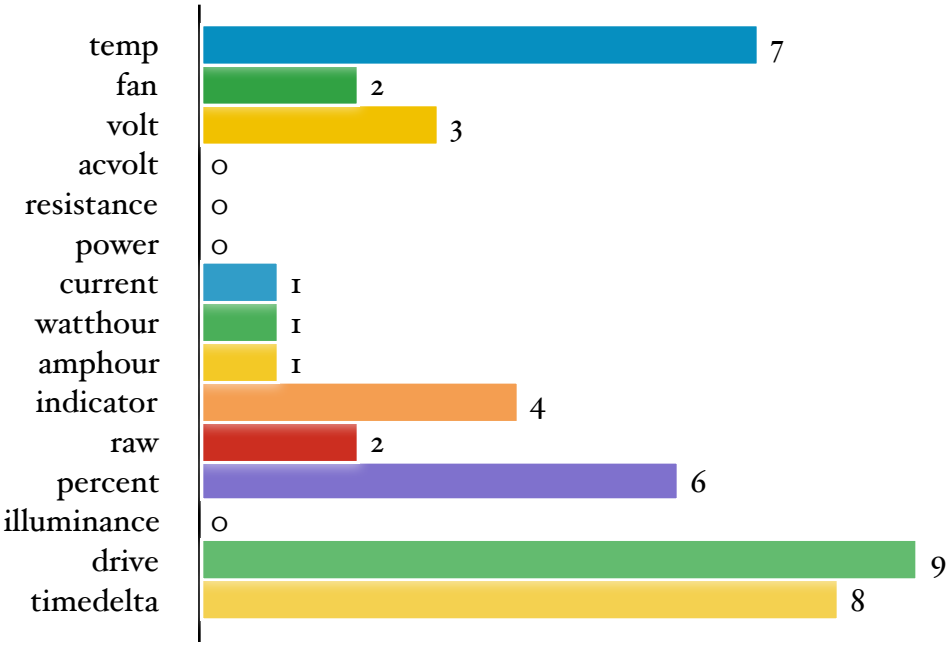

Chart V. Sensor type popularity in OpenBSD 4.6 based on the number of ***drivers with status support*** *using each type.*

It is apparent from Chart VI that the temperature, fan and voltage sensors, which are the most popular sensor types amongst the set of all drivers, do not usually carry any information regarding the status of their sensors. It is also clear that the design of certain sensors, like *drive* and *timedelta*, always presumes the inclusion of the status information.

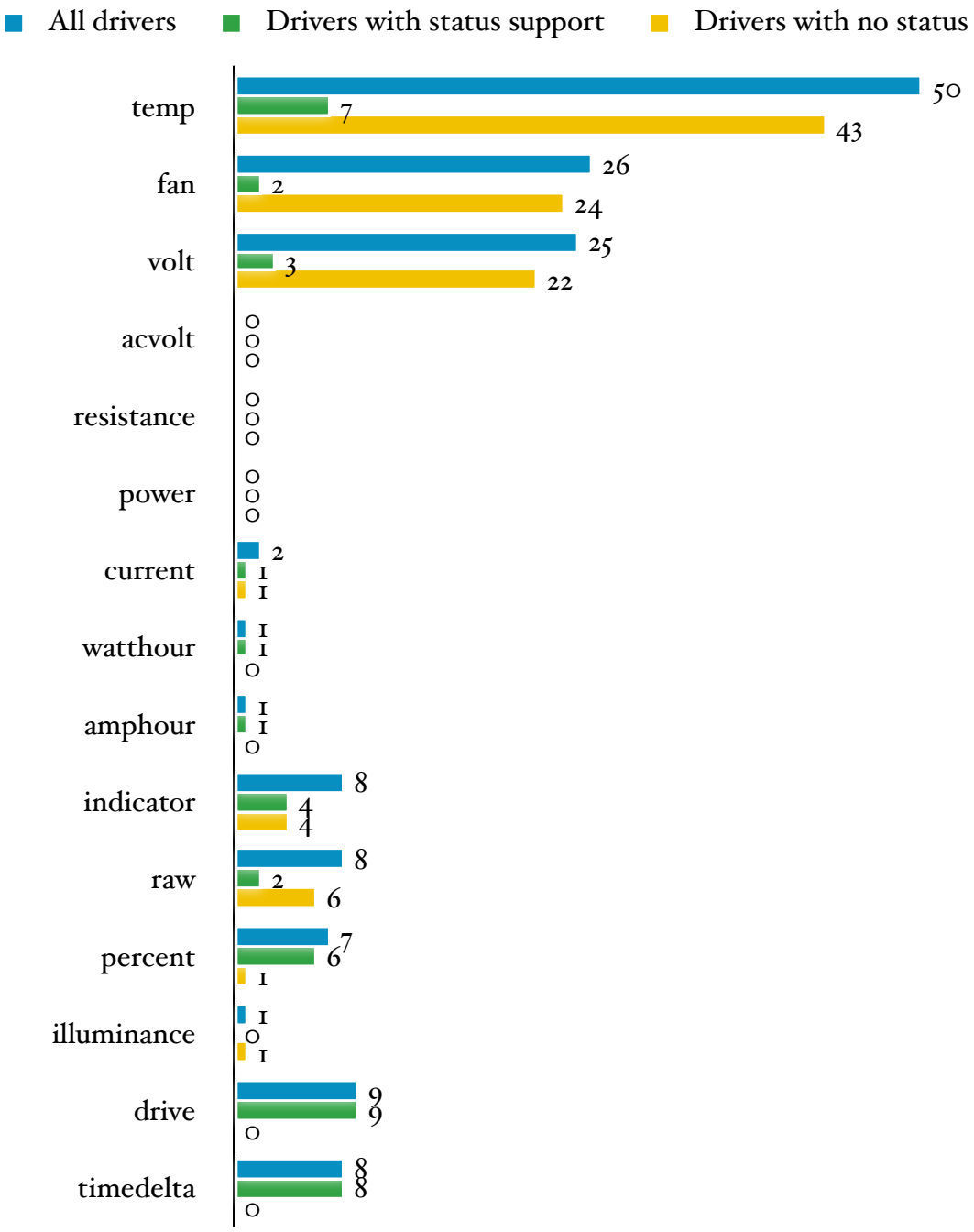

Chart VI. Sensor type popularity in OpenBSD 4.6 based on the number of drivers using each type, itemised by status support.

5. I²C Sensors and Bus Scan

In this section, we describe how OpenBSD goes about detecting various sensors on the I²C bus.

For a general-purpose operating system, the I²C bus poses a significant problem as it does not have a standard method of detecting what devices appear at which addresses. (Suffice it to say that some automatic enumeration has been introduced in SMBus 2.0 specification [Intel.SMBus20], named SMBus Address Resolution Protocol (ARP); however, hardware/silicon support for SMBus 2.0 and ARP is still quite rare in this day (ten years after the publication), thereby, software support is not deemed warranted yet.) Most devices on the I²C bus have at most 256 registers from which information can be read, or to which some data can be written, and various manufacturers use different registers to place the identification information regarding their chips. Moreover, in many cases this identification information could very easily be rather limited and not terribly unique, making conflicts of all kinds possible. (In some cases, there may not be any identification information at all, making it impossible to have automatic support of certain chips on generic hardware.) In addition, the bus is rather slow, and accessing the same registers multiple times may take a significant amount of time if all the drivers would individually probe the chips at all possible addresses and would be enabled at the same time.

5.1. Open Firmware and I²C

The problem with a lacking discovery mechanism is alleviated on the Open Firmware architectures — macppc and sparc64 in OpenBSD — where the operating system can query Open Firmware properties such that it then knows exactly which chips are to be found at which I²C addresses on which I²C bus. In turn, the *match()* procedure of each individual sensor driver then does no probing other than a simple comparison of ASCII strings — the string with the name of the chip as supplied by

the bus to the strings that the driver supports. For example, on macppc such a string could be "adt7467" [adt.4] or "ds1775" [lmtemp.4].

5.2. I²C bus scan through i2c_scan.c

Those architectures that do not have Open Firmware, but still support I²C (i386, amd64, alpha, armish, socppc), have a scanning mechanism that is implemented in /sys/dev/i2c/i2c_scan.c. The idea is to be able to enable as many I²C sensor drivers as possible without any adverse effects on the stability and reliability of the boot process. This is accomplished in several ways.

The scanning algorithms run through all I²C addresses that are known to contain certain interesting sensors, and a different scanning function is used for those addresses containing EEPROM chips (like those implementing Serial Presence Detect (SPD) functionality that provides information about the memory modules [spdmem.4]). During much of the scanning procedure, the value from each register is ever read from each I²C address at most once (being cached for subsequent reads during the scanning procedure). Certain registers at certain addresses, however, are banned from ever being read from the hardware during the scanning procedure, if it is known that accessing such registers could cause unintended results. For example, the logic never tries reading the 0xfc register from chips that may resemble Maxim 1617 in some way, as reading such register may cause some problematic behaviour on some hardware.

The result of a successful i2c_scan.c iteration over each individual I²C address is a string describing the chip, similarly to the one provided by the Open Firmware on the Open Firmware architectures. An example of such a string could be "w83793g" [wbng.4]. Since the I²C slave interface between the Open Firmware-based I²C discovery and i2c_scan.c-based discovery is the same, the same sensor drivers can be used across all architectures without any losses of the more trustable information regarding identification of the chips that the Open Firmware architectures provide.

Misidentification or even improper probing of the chips can be fatal — it is well known that some versions of the lm_sensors package from the Linux land have "bricked" many ThinkPads due to improper probing of the I^2C bus, where the contents of some EEPROM chip would be wiped out during the lm_sensors probing procedure. [lm_sensors.ThinkPad] It is noteworthy to mention that the real cause with such "bricking" is believed to have been the chips that do not fully adhere to the I^2C standard; however, it is hardly a good excuse if one's laptop is dead after running some part of the lm_sensors package on GNU/Linux. Therefore, on OpenBSD a great care has been taken to avoid any such incidents at its root; and with the probing procedure being enabled by default in all GENERIC kernels on all architectures that require it, there is a sufficient empirical proof that such care has been entirely adequate.

5.3. I^2C drivers

In a nutshell, all I^2C sensor drivers in OpenBSD match exclusively based on strings provided by either Open Firmware or i2c_scan.c, and both of the scanning mechanisms are enabled by default on those architectures that they support, meaning that in the vast majority of times there is absolutely no need for the user to do any kind of juggling to find about which chips are located at which addresses and are supported by which drivers (an unfortunate approach that is taken by NetBSD, for example). To rephrase, all supported I^2C sensor drivers are enabled in the GENERIC kernels on OpenBSD and automatically work out of the box on all supported architectures.

5.4. I^2C register dumps

As explained earlier, the i2c_scan.c probing is run automatically (during the kernel boot time) on all systems that have no other I^2C discovery mechanism. When it encounters a chip with an unknown signature, or with a known signature, but that is still unclaimed by any driver, then it dumps the whole register set of the chip into

the “dmesg”, the system message buffer [dmesg.8]. (In order to avoid redundant data in dmesg, the most often occurring register value is reduced from the dump before the dump is printed into the dmesg.)

It is a standard and longtime practice in OpenBSD to ask users to voluntarily send in their dmesgs to dmesg@openbsd.org archive, which is a private archive accessible only by OpenBSD developers. [deRaadt.misc98] This practice ensures that OpenBSD developers will always have confirmations that OpenBSD continues running on various hardware that the users possess. Because all necessary information regarding unsupported I²C sensor chips is already conveniently located in the dmesg by default, it makes it very easy for the user to cooperate and provide such information to the developers by simply sending the dmesg (and, preferably, the output of `sysctl hw.sensors`, too) to dmesg@openbsd.org. This allows OpenBSD developers to ensure that both old and new hardware is always properly supported, and perform quality assurance regarding the stability of such support, as well as account for the variations of the hardware.

5.5. I²C sandboxing for driver development

It is relatively easy to implement a sandbox environment in which new I²C drivers could be tested against the I²C register dumps from dmesgs. The reason for this is that many hardware sensor device drivers only do *reads* from these registers — in other words, they usually do not do any *writes*. (Lack of unconditional writes is usually done on purpose, since on general purpose hardware there is no definite certainty that the driver is actually talking to a sensor chip, as opposed to some EEPROM device, so nothing should be written to the registers of the chip unless such writes are absolutely necessary. [deRaadt.priv06]) Therefore, a full selection of register values can be used to simulate the I²C bus read operations in the userland and to test much of the functionality of the driver without even booting a new kernel or having the required hardware available at one’s disposal. [Murenin.TOJ07.wbng]

To do such simulation, we implement several small functions and copy some other functions from the kernel. First, we parse the register values from the dump and fill in a 256-array of *uint8_t* type. Then we allocate a *softc* of the size as defined by the *struct cfattach* of the driver we are trying to sandbox (e.g. *wbng_ca.ca_devsize*). Next, we define a local *struct i2c_attach_args* datastructure, allocate its *ia_tag* field of size *sizeof(struct i2c_controller)*, and set *ia_tag*'s *ic_acquire_bus* and *ic_release_bus* fields to some dummy functions that do not do anything other than returning a 0 and nothing respectively. Implementation of the *iic_exec()* emulation is equally straightforward, where the read operation is based on the contents of the pre-populated *unit8_t* array of 256 elements, whereas the write operation returns a permanent error.

The next step is ensuring that the kern_sensors.c is adapted to the userland. For this, several of its functions must be adjusted. Apart from removing *splhigh()* and *splx()* calls and some extra *#include*s, the *sensor_task_register()* routine has to be reimplemented. All the *sensor_task_register()* routine needs to do is call the refresh function of the driver once, and do nothing more. We would also like to be able to see the sensor readings as updated by the sandboxed driver, and for this, a *print_sensor()* function can be copied from the userland sysctl(8) utility, changing the type of its only parameter from *struct sensor* to *struct ksensor*.

After all of this preparatory work is done, all we have to do is call the *ca_attach* function of the *struct cfattach* datastructure with the preallocated *softc* and *struct i2c_attach_args* (e.g. do a *wbng_ca.ca_attach(NULL, softc, &ia)* call), and go through the linked list of all sensors on the relevant sensor device to print the readings with *print_sensor()*. All files must be compiled with "-D_KERNEL -I/usr/src/sys/".

In the described procedure, the file of any I^2C sensor driver that meets the criteria of not doing any unconditional writes to any registers can be taken directly from the kernel to the sandbox environment without requiring any modifications of the

driver's code. All that needs to be modified is the reference to the appropriate *struct cfattach* variable in our sandbox (*wbng_ca* in the example).

Having such a sandbox environment streamlines I^2C driver development and initial testing. The wbng(4) and andl(4) represent the two drivers that have been developed in the said sandbox and that were originally tested against several I^2C dumps from the dmesg@openbsd.org archive. [Murenin.TOJ07.wbng]

6. Evolution of the Framework

The first revision of OpenBSD's sysctl-based hardware sensors framework has originally been brought to OpenBSD in 2003 by Alexander Yurchenko *(grange)* to accommodate some hardware monitoring drivers that he was porting from NetBSD. [grange.priv05] In this section, we describe how the framework has evolved and what were the major milestones in the development.

6.1. Framework timeline

During 1999/2000, envsys(4) and sysmon(4) interfaces have been introduced in NetBSD, along with the lm(4) and viaenv(4) hardware monitoring sensor drivers. A utility called envstat(8) is used to query /dev/sysmon for various sensor readings. From 2000 until 2007, the documentation of NetBSD's envsys(4) interface has been suggesting that the entire API should be replaced by a sysctl(8) interface, should one be developed. (This comment has since been removed with the introduction of the envsys 2 API on 2007-07-01.)

On 2003-04-25, the lm(4) and viaenv(4) drivers have been ported from NetBSD and committed to OpenBSD by Alexander Yurchenko *(grange)*; however, instead of porting the envsys(4) API from NetBSD, a much simpler and more straightforward API has been devised and developed based on the sysctl interfacing. The sysctl addressing has been made very simple — any sensor from any sensor device would have a global ordinal number, and would be accessible by `sysctl hw.sensors.N`, where N would be such global ordinal number.

During some periods of 2004 and 2005, various general and shared parts of the framework have been improved in several ways by many people, mostly Alexander Yurchenko *(grange)*, David Gwynne *(dlg)*, Mark Kettenis *(kettenis)* and Theo de Raadt *(deraadt)*. For example, David Gwynne has introduced optional sensor states before OpenBSD 3.8, and then the *sensor_task_register()* routine before OpenBSD 3.9. [Gwynne.Open06] Theo de Raadt has implemented a large part of the i2c_scan.c

logic and a big deal of related I^2C drivers during December 2005 and January 2006 before OpenBSD 3.9. [deRaadt.zdnet06] Various other individuals have made great contributions on the driver front; for a full list of their names, please see the related OpenBSD sourcecode and CVS revision history.

On 2006-12-23, Theo de Raadt has committed the patches provided by Constantine A. Murenin that converted the 44 device drivers (i.e. all the drivers that were using the sensors framework at that time) and multiple userland applications from the simplistic one-level "hw.sensors.<N>" style of addressing to the more evolved and flexible two-level "hw.sensors.<xname>.<type><numt>" style of addressing (e.g. *hw.sensors.11* became *hw.sensors.lmo.temp2* after the change). The new style of addressing brought up several benefits, from unbloating the kernel by removing certain redundant information from the drivers (like the "Temp1" strings in sensor description which used to be required for some identification purposes) to making it easier to use the sensors API in both shell scripts and C/C++ programmes (since the addressing became more stable and predictable across heterogeneous machines). [Murenin.IEEE07] [Murenin.TOJ06] The userland API of the framework has been stable since this patch and OpenBSD 4.1.

In 2007, two final changes have been made to the ABI of the framework and the kernel *sensor_task_register(9)* API. The first change by Theo de Raadt separated the datastructures used by the kernel and the userland, so that certain internal information used for bookkeeping (i.e. the linked lists) would not be released into the userland. The second change was made by David Gwynne regarding the *sensor_task* API.

Outside of the OpenBSD realm, a project for porting the framework to FreeBSD has been suggested, proposed, approved and funded for Google Summer of Code 2007. The project has been successfully completed and the final patch has been released on 2007-09-13. The results of the work have been committed into both DragonFly BSD

and FreeBSD shortly thereafter. [Murenin.GSoC07.sum] For more details, please refer to a separate section in this paper.

6.2. Evolution of drivers

The sensors framework has originally been introduced with OpenBSD 3.4 (2003) and only had 3 drivers that were using the functionality provided by the framework — lm(4), it(4) and viaenv(4). Chart VII represents how the number of drivers that expose sensors to the framework has been growing over the years since the original introduction of the framework, where OpenBSD 4.6 (July / October 2009) has 75 drivers using the framework. One can notice a significant spike in the number of drivers around OpenBSD 3.9, where the i2c_scan.c functionality was developed and 19,5 new I^2C sensor device drivers have been introduced (the 0,5 refers to the lm(4) attachment at iic(4)) [deRaadt.zdnet06].

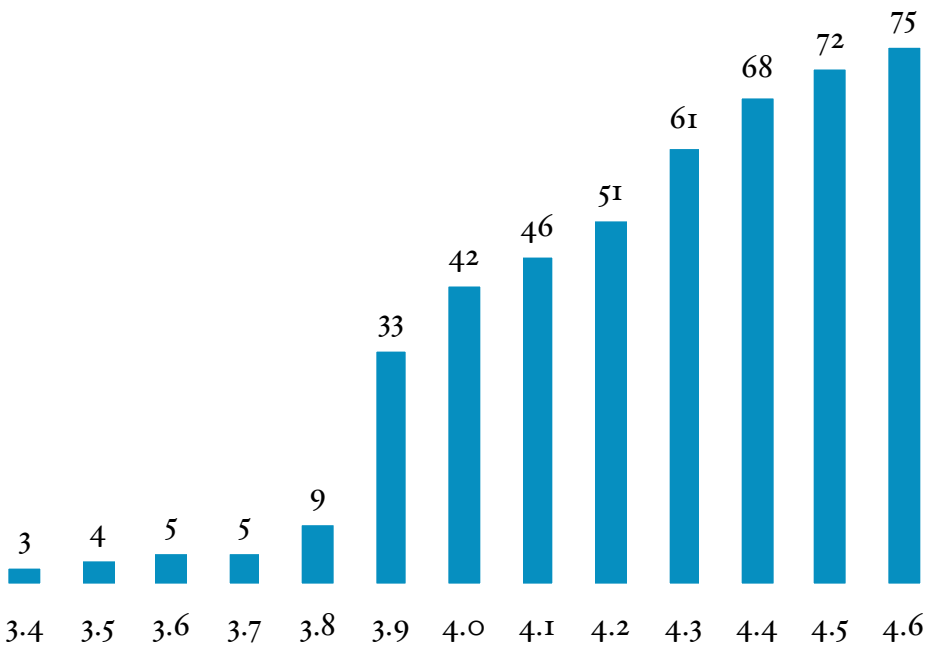

Chart VII. Number of drivers using the sensors framework from OpenBSD 3.4 to 4.6.

The counts regarding the number of drivers using the framework have been generated by counting the number of files that do a *sensordev_install()* call, a call that links the individual sensor tree of each invocation of the driver with the global sysctl tree. For OpenBSD versions prior to 4.1 (i.e. OpenBSD 3.9 and 4.0), the number of files with the *sensor_add()* call was counted; for OpenBSD versions prior to 3.9 (i.e. OpenBSD 3.5 to 3.8), the number of files with the *SENSOR_ADD()* macro invocation was counted; for OpenBSD versions prior to 3.5 (i.e. OpenBSD 3.4), the number of driver files with the reference to the *sensors_head* variable was counted. These changes in the API are summarised in Table III.

OpenBSD	Exposing sensors
3.4	extern struct sensors_head sensors; SLIST_INSERT_HEAD(&sensors,);
3.5 to 3.8	SENSOR_ADD();
3.9 and 4.0	sensor_add();
4.1 and up	sensordev_install();

Table III. Evolution of the sensor API from OpenBSD 3.4 to 4.6.

7. Related Frameworks

OpenBSD's hardware sensors framework compares favourably with the competition in the areas of simplicity, hardware support and ease of use right out of the box.

As a relatively recent example of unique hardware support, OpenBSD 4.4 (November 2008) was the first release of any operating system to support the integrated temperature sensors in AMD Family 10h processors (right out of the box, of course). [km.4] Another noteworthy leadership of OpenBSD 4.4 is in the support of the JEDEC JC-42.4 SO-DIMM temperature sensors, which are still unsupported by many competing products to this day. [deRaadt.jedec08] [sdtemp.4] [Biancuzzi.44]

In this section, we discuss alternative solutions to the OpenBSD's sysctl hardware sensors framework. For practical comparison and external reference, we have also developed a driver, aibs(4), from scratch, supporting ATK0110 ACPI hardware monitor that is available in most recent desktop motherboards from ASUSTeK, and we have ported this driver to all the four open-source BSD systems (OpenBSD, DragonFly BSD, NetBSD and FreeBSD), such as to provide a hands-on example of the best characteristics of each system from the point of view of hardware monitoring. As of this writing, the aibs(4) driver has already been fully integrated and readily available in OpenBSD, DragonFly BSD and NetBSD.

7.1. NetBSD envsys / sysmon

In general, many NetBSD and OpenBSD drivers have been cross-ported as far as the sensors framework is concerned. The NetBSD API is more complicated than the one in OpenBSD, which was specifically true before some simplification was brought with NetBSD's envsys 2 on 2007-07-01.

On NetBSD, the majority of sensor drivers are disabled by default. There is also no automated I^2C scanning procedure (the user is expected to know exactly which sen-

sor devices they have at which addresses before the drivers can be enabled). Many I^2C drivers that are present in OpenBSD are still missing from NetBSD.

The total number of drivers in the last release of NetBSD as of December 2009, NetBSD 5.0.1, calculated by the number of calls to the *sysmon_envsys_register()* function from unique drivers (ug(4) is counted only once), is only 31, versus 75 as the respective count in the latest version of OpenBSD — OpenBSD 4.6. Much of the difference in the number of drivers is due to the stagnated I^2C scene on NetBSD, which only has 5,5 I^2C sensor drivers, whereas OpenBSD has 29,5 I^2C sensor drivers (the 0,5 refers to the lm(4) driver, which can be attached on both isa(4) and iic(4) busses). (Attentive readers may find this comparison to be somewhat superficial, since some drivers may support more sensors than others; however, the general idea that NetBSD supports considerably fewer I^2C sensor chips still stands even after a closer examination of the drivers.)

In July 2007, a new proplib(3)-based version of the envsys(4) framework was introduced, called envsys 2, which has later been adjusted the same year. NetBSD's *sysmon_envsys_sensor_attach()* API introduced in 2007-11 appears to be paying a tribute to the OpenBSD's *sensor_attach()* API that has been available in OpenBSD for about a year prior. Prior to envsys 2 introduction in July 2007, NetBSD's API did not support detachable sensors. On OpenBSD, detachable sensors have been supported since January 2006.

In June 2009, extended support for sensor limits was introduced in the development version of NetBSD — NetBSD 5.99.13, whereby the drivers could optionally export limits (critmax, critmin, warnmax, warnmin) from each individual sensor into the sysmon_envsys(9) framework, such that the kernel framework could then automatically monitor the limits and generate events when the limits are crossed. The said introduction also featured the possibility for the user to modify the limits used by the underlying hardware (whereby the driver would write the user-specified limits

directly into the hardware), such that the hardware could then automatically alert the driver when the limits are triggered, so that the driver could in turn change the status field in the kernel datastructure containing the sensor without anything in the kernel actually doing the limit comparisons (thereby intendedly reducing the load on the system). In December 2009, as a part of our aibs(4) porting from OpenBSD / DragonFly BSD to NetBSD, it was, however, discovered that the relevant src/sys/dev/sysmon/sysmon_envsys_events.c code from June 2009 was not tested adequately, and the advertised functionalities did not work as intended. [Murenin.techkern09]

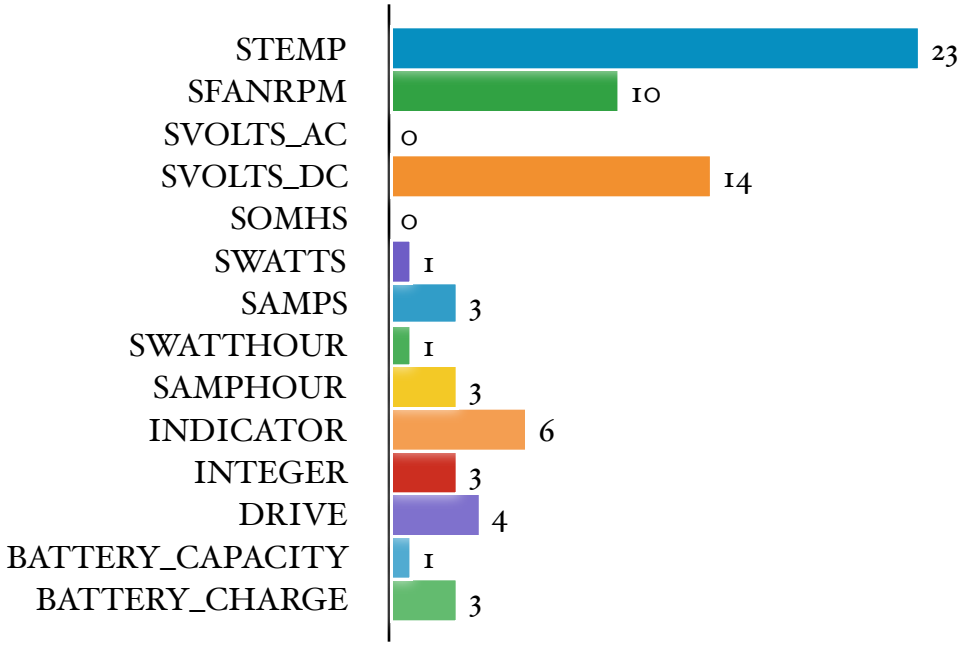

Chart VIII. Sensor unit popularity in NetBSD 5.0.1 based on the number of drivers using each envsys unit.

Sensor types are roughly the same between NetBSD and OpenBSD. Sensors of the *drive* type from OpenBSD's bio(4)-based device drivers [Gwynne.Open06] were committed to NetBSD on 2007-05-01. No *timedelta* sensors have been ported to NetBSD as of December 2009. Chart VIII provides a graphical summary between

the popularity of sensor types in NetBSD 5.0.1 out of a total of 31 drivers; a trend similar to the one in OpenBSD is apparent — temperature sensors are by far the most popular, with voltage and fan sensors being in the second most popular category, much ahead of all the sensors of all other types.

It is evidenced from Chart IX that the sensor states are used differently in NetBSD than they are in OpenBSD: essentially, a state of OK is missing in NetBSD, since every driver is required to initialise the state of any valid sensor to SVALID (and all drivers promptly do), but the SVALID state makes no distinction between a sensor that has defined limits and is within those limits and a sensor that does not have any limit monitoring whatsoever. (On OpenBSD, sensor invalidation is accomplished with a sensor flag SENSOR_FINVALID, and the status field is used exclusively for the limit-based status monitoring alone.)

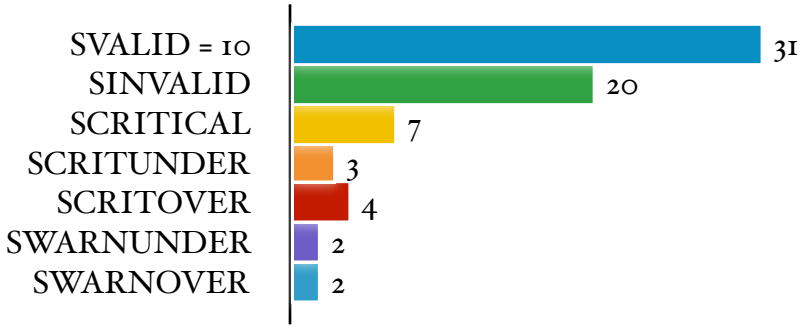

Chart IX. Sensor state popularity in NetBSD 5.0.1 based on the number of drivers specifying each state.

In theory, NetBSD's sysmon_envsys(9) kernel framework has considerably more features than OpenBSD's sensors.h, including things such as kernel events and driver-initiated and user-modifiable limits that are supported on the kernel level. OpenBSD does support the limits on the driver level, but there is no way for the user to see what the driver limits are (only the resulting state is exported by the drivers); however, although the user cannot amend the limits on the kernel level in OpenBSD, user limits can always be specified in addition to the (automatic) driver

limits in sensorsd.conf, and then both the driver and the user limits could be monitored in the userland by means of sensorsd(8). In practice, through the porting of the aibs(4) driver from the OpenBSD hardware sensors framework to NetBSD's sysmon_envsys(9), we have discovered firsthand that many features in NetBSD's sysmon_envsys(9) are quite ambiguous, unnecessarily complicated and even questionable, where the meaning and the reasoning behind the naming, introduction or staying of, for example, certain flags or incomplete functionality, is unclear. It is generally agreed within the relevant NetBSD community that the NetBSD's sysmon_envsys(9) framework indeed needs another redesign or an overhaul.

7.2. FreeBSD general sysctl tree

The last release of FreeBSD as of February 2010 — FreeBSD 8.0–RELEASE — does not have a sensors framework per se, however, several drivers exist that nonetheless export sensor-like data into the general sysctl tree. These drivers include, but are not limited to, acpi_thermal(4), acpi_ibm(4), acpi_aiboost(4), ad7418(4), coretemp(4), amdtemp(4) and possibly others.

Most of these FreeBSD drivers do not adhere to any specific namespace by which they make the sensors available, the only exception being the temperature sensors and only in some select few drivers, where the temperatures could specifically be marked with an "IK" (Integer Kelvin) *oid_fmt* modifier and are encoded through the ACPI Kelvin notation (where 0 °C is represented by the integer value of 2732, and the readings are automatically converted back to °C by sysctl(8)). Even then it would appear that not even all the drivers use the "IK" notation for their temperature sensors, where several drivers instead provide regular typeless integers that are read directly off of hardware. These integers sometimes may not even correspond to any existing convention regarding applicable units — for example, acpi_aiboost(4) essentially provides the unitless integers that represent temperatures on the deci-Centigrade scale (e.g. a unitless integer of 430 would mean 43.0 °C). In other words,

non-temperature sensors are always encoded as simple plain integers without any specific units associated with them, and temperature sensors are not necessarily unit-bound either.

For additional details regarding the sensor development on FreeBSD, please refer to a separate section in this paper describing the port of the OpenBSD hardware sensors framework to FreeBSD.

7.3. lm_sensors

The lm_sensors package in GNU/Linux requires significant amount of configuration by the end-user, and is much more difficult to get right compared to the OpenBSD's sysctl hardware sensors framework, which works right out of the box. [deRaadt.zdnet06]

However, lm_sensors does provide additional functionality that is still missing from OpenBSD, namely, the ability to do extensive configuration and customisation of certain chips as well as the monitoring environment. Interfacing with some fan-controlling functionality is provided in some drivers, as well as the ability to modify fan divisor bits [Murenin.IEEE07]. That is, if the user has the patience and time to figure it all out.

8. Port to FreeBSD / DragonFly BSD

Outside of the OpenBSD realm, a project for porting the framework to FreeBSD has been suggested on the FreeBSD's mailing lists [Theile.arch07] and then added to the official "ideas" page [FreeBSD.ideas] in early 2007. Based on the suggestion, an application was submitted by Constantine A. Murenin for Google Summer of Code 2007 funding to port over the framework to FreeBSD. The proposal [Murenin.GSoC07.prop] has been voted up by the FreeBSD committers to be approved for a funding slot, and was subsequently funded by Google.

8.1. Summer of Code 2007

During Google Summer of Code 2007, all relevant parts of the framework that were expected to be ported from OpenBSD to FreeBSD have been ported successfully. This included the sensors API and all relevant documentation, and appropriate parts of the userland applications sysctl(8) and systat(1). (The sensorsd(8) sensor monitoring daemon did not require any modifications to its C code for it to be ported, since the userland API was made compatible between OpenBSD and the FreeBSD port; however, some glue integration code was, of course, developed and submitted in the final patch.) In addition to the base components of the sensors framework itself, two sensor drivers have been ported that support the hardware monitoring modules of the most popular Super I/O solutions: lm(4), supporting many Winbond chips, and it(4), supporting many ITE Tech chips. Moreover, FreeBSD's coretemp(4) driver has been converted to use the new framework, too. [Murenin.FQSR07]

On 2007-09-13, a complete final patch that combined all the little parts of the framework has been publicly announced and released, together with a bullet list of all the items that were included in the said patch. [Murenin.GSoC07.fin] However, the FreeBSD HEAD tree was still frozen during that time due to the upcoming RELENG_7 branching.

8.2. Sensors framework in DragonFly

On 2007-09-25, Hasso Tepper posted a message to DragonFly's *submit@* mailing list [Tepper.submit07], and contacted Constantine with a thank-you note regarding the port, mentioning that, with small adaptations, the work will be soon committed into DragonFly BSD [Tepper.priv07].

On 2007-10-02, the framework and the three ported drivers have been committed into DragonFly BSD 1.11 [Murenin.GSoC07.sum], and so far have been part of multiple DragonFly BSD releases.

8.3. Sensors framework in FreeBSD CVS

Shortly after the DragonFly BSD commit, the patchset with the framework was approved by re@FreeBSD.org (FreeBSD's Release Engineering team) to be committed into CVS HEAD once the RELENG_7 branching is done and the freeze is over.

On 2007-10-14 (the same week when the branching was done and the freeze was lifted), the framework has been committed into FreeBSD 8.0-CURRENT by Alexander Leidinger. The commit has generated a lot of attention in the FreeBSD community, as some people were very happy to finally be able to use the framework right out of the tree, yet others were unhappy with certain architectural decisions that were much more appropriate to the OpenBSD architecture and philosophy than to the one of FreeBSD.

On the same day as the commit was made, Poul-Henning Kamp voiced his objections to the architecture of the framework, for the framework having too much OpenBSD feel into it. A very heated discussion arose, where many people tried voicing their opinion about whether the framework should or should not stay in FreeBSD (see FreeBSD archives of the *cvs-src@* and *arch@* mailing lists around the time for complete discussion threads). Poul-Henning has requested for the framework to be backed out; it was then backed out a day later.

Technically, a separate sensors framework is less needed in FreeBSD as opposed to OpenBSD, since FreeBSD has "sysctl internal magic" since 1995 that dynamically manages every node in the sysctl tree. In OpenBSD, on the other hand, the majority of the nodes in the sysctl tree are still statically defined at compile time, using preprocessor defines for MIB integers and arrays of strings for textual representation of such MIB elements. In NetBSD, the sysctl auto-discovery and dynamic registration of nodes were introduced only in December 2003, whereas the envsys framework has been available for several years prior. In general, however, the sensors framework provides more restricted namespace for devices to export sensor-like data, whereas nodes in the sysctl tree are often rather arbitrary. [phk.arch07] This is precisely the reason why a separate sensor framework is valuable nonetheless, since it allows one to have many sensor-like values from different components under a single and predictable tree.

It is important to note, however, that the summer of code project was in fact done to PHK's satisfaction; he was unsatisfied merely with the fact that the framework did not solve the niche in the FreeBSD-way. [phk.gsocgood07] Poul-Henning Kamp emphasised that he does not want the framework to be available in FreeBSD such that the space is left clear, and someone might design a framework more suitable for FreeBSD in the long term. However, since the framework in question was based on a framework that has been available in NetBSD since as early as 1999, and FreeBSD is still missing any such framework, it remains unclear if such a framework will ever be developed for FreeBSD. [Murenin.Login08]

We trust that this manuscript provides enough interesting information regarding the overall picture of the sensor drivers on various BSD systems that these issues could be revisited at a later time, and perhaps a new design of the framework could be envisioned specifically for FreeBSD, ensuring that the pitfalls taken by OpenBSD and NetBSD are not repeated, and FreeBSD architectural decisions and FreeBSD's Newbus device framework are given a higher consideration from the ground up.

9. Availability

All described OpenBSD source code, apart from the userland I^2C sandboxing environment, is publicly available in the OpenBSD CVS repository and in the official releases. The final patch for FreeBSD is available in FreeBSD's perforce repository. The history of the FreeBSD commit is available in the FreeBSD CVS and SVN repositories, as is the complete patch of the framework itself. DragonFly BSD code is available in the DragonFly CVS and GIT repositories and is part of the official releases.

10. Introduction and Motivation for Fan Control

Power consumption and heat dissipation are gaining widespread attention in many sectors where personal computers are deployed. The process of transferring the heat away from the system is usually accomplished with the help of some combination of fans. However, fans themselves are known to significantly contribute to the total power consumption of the system, and also pose an additional problem of emitting a persistent background noise, which, in turn, is believed to increase the stress levels of those who are exposed to the noise for prolonged periods of time and decrease the motivation of the workforce [Evans.stress].

Brand-name system integrators have often been solving the problem of balancing the noise and thermal characteristics by carefully choosing the fans that are used to cool the system down, together with employing some proprietary fan-controlling logic that automatically adjusts the speed of the fans as needed, in order to achieve the lowest possible noise levels without compromising the thermal zone requirements of the system.

However, smaller integrators, whether individual users or companies assembling personal computers from off-the-shelf components, have to play it safe, and oftentimes have to install excessive fans, even if such fans are not strictly necessary for maintaining reliable operation of the computer, due to uncertainties regarding the power consumption of individual components that are used to assemble the system. In essence, this results in unnecessary noise, whereas the opposite approach of putting fewer than necessary fans may result in overheating problems when the computational capabilities of the system are fully utilised for an extended time period. Fortunately, not everything is lost, and many off-the-shelf motherboards feature an integrated hardware monitoring silicon chip that could also allow the user to control the voltages that are supplied to the fans, thus allowing the user to have more control over the environmental characteristics inside of their PC.

Opponents of fan control may cite various reasons against decreasing fan-speed and increasing the operating temperature of the system. A commonly cited negative factor from such a group is the decrease in the product life span of certain components, especially and most importantly Hard Disc Drives (HDDs). An observation that was perhaps relevant in the past is now often disputed — a recent study published by Google Inc. suggests that temperature and activity levels are much less correlated with drive failures than previously reported. [Google.FAST07] As for other components of the system, most of them are rarely considered as critical and irreplaceable as the HDDs, and most often they are specifically designed and advertised to operate in rather extreme temperature conditions anyway. Proponents and practitioners of quiet computing can also easily provide the empirical proof that their systems, if configured for technically-reasonable upper-level temperatures, can run stable and reliable for a number of years to come.

In this part of the thesis, we outline some features and problems with these hardware monitoring chips and with their use in the commonly available hardware as it relates to fan control, and we discuss the interfacing options that allow the user to conveniently communicate and enforce their thermal policies.

11. Related Work on Fan Control

In this section, we provide an overview of some related works that allow the user to specify thermal characteristics of their system.

11.1. Interfacing from the BIOS

Although popular off-the-shelf motherboards have been physically supporting at least some fan-controlling features for a considerably long time now, it has not been until recently that these boards have been bundled with BIOSes that allow any kind of interfacing with the fan-controlling characteristics of the hardware monitoring chips.

Although most modern BIOSes that are included with the new motherboards do allow the user to monitor the temperature, fan and voltage characteristics of the board through the hardware monitoring chip, it is still not universally common to find boards that feature adequate level of fan-controlling interfacing from within the BIOS menus: some boards do not have any configurable options at all, whereas others simply have an On / Off switch regarding some brand-name 'Quiet Fan' feature from the manufacturer of the motherboard.

As a specific example, consider a relatively popular Mini-ITX board that is sold under the Intel brand: Intel D201GLY2. [Intel.D201GLY2] Our sample board has been purchased from newegg.com in December 2007 and originally came with the 0122 BIOS revision dated 2007-08-22. As far as the hardware monitoring and fan controlling goes, the original BIOS only had the 'Hardware Monitoring' menu selection in its Advanced System Setup tab, with no options for fan control. However, from the release notes that accompany later BIOSes, we could see that 'System Fan Control' option has been introduced in a new 'Advanced' → 'Fan Control Configuration' menu of the 0129 BIOS revision from 2007-10-08. In order to test the new option, we have updated the BIOS of this board to the very latest revision numbered 0149 and dated 2008-12-16. After updating the BIOS and going into the 'Advanced'

→ 'Fan Control Configuration' menu, we have been rather disappointed to find out that the 'System Fan Control' is the only option that has now been implemented, and the only parameters it can have is 'Enabled' or 'Disabled'. For illustration purposes, please refer to Figure I. In further sections, we show that the hardware monitoring chip itself has many more fan-controlling options that may be of specific and reasonable interest to the user.

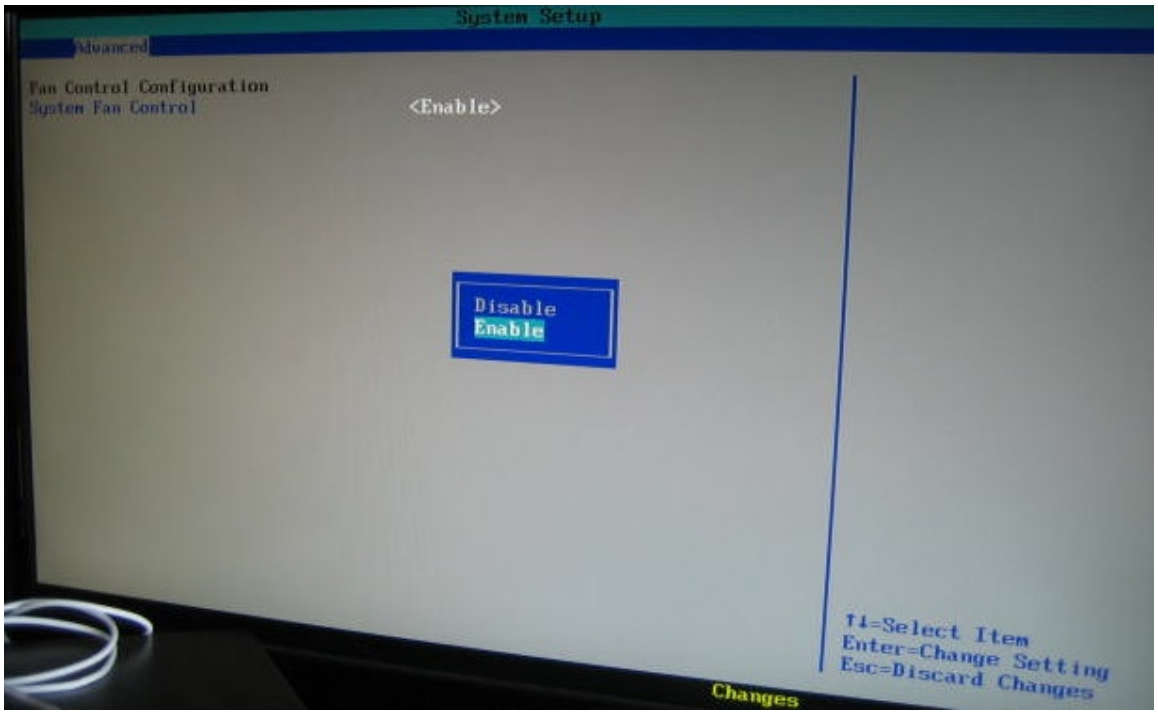

Figure I. A sample screenshot of the options that have been provided in the BIOS revision 0149 dated 2008-12-16 of the Intel D201GLY2 board, which was one of the boards we have used in testing our prototype.

11.2. ACPI

ACPI was introduced with the intention of providing a unified interface for various hardware discovery and power management functions. [Watanabe.ACPI] [Intel.ACPI] However, the reality of modern implementations shows that fan con-

trolling is not one of the functions that is universally provided by ACPI, at least not on the common desktop and server off-the-shelf motherboards.

Laptops, on the other hand, may sometimes feature more useful details regarding environmental characteristics in their ACPI Differentiated System Description Tables. This may include the ACPI Thermal Zones (the concept of which is not specifically unique to laptops, but in practice is much less likely to be present in the desktop boards). Thermal zones may optionally have a number of Active Cooling objects, which define temperature thresholds at which Fan Devices are engaged. Each Fan Device, in this sense, may be a separate physical device, or may represent a logical setting of a varying speed on a single fan (or a set of fans). In practice, however, most of this functionality is still not implemented in the ACPI tables of available hardware; moreover, it is not even clear if such functionality may be found useful for general-purpose off-the-shelf motherboards, where the creator of the ACPI DSDT (in other words, the motherboard manufacturer) may not possibly be aware of the thermal characteristics and active cooling requirements of a custom-build box. For this reason, we would leave further discussion regarding ACPI Thermal Management for a future discourse.

It should be noted, however, that some brand-name laptops and motherboards do have interesting information in their ACPI DSDT that may relate to the topic of fan control. Notable examples of such systems include IBM/Lenovo ThinkPads, ASUSTeK motherboards with the AI Booster / ATK0110 feature and ABIT motherboards with the ABIT uGuru feature. Let us briefly describe the better known features that are provided through these DSDTs.

Out of our interest to environmental monitoring and fan control, ThinkPads provide multiple temperature sensors (as many as 16 in total), one fan RPM sensor and one fan-speed control setting. The speed control setting could be set to 'automatic', 'disengaged' (meaning, no control is done and fan is run at 100%) and the 'manual'

mode. In the manual mode, the setting can be varied between 0 and 7, where values above zero seem to guarantee that the fan actually runs [thinkwiki], meaning that ThinkPads are, essentially, fool-proof, unlike the manual duty cycle mode in chips of custom-build systems from the off-the-shelf components, which are unlikely to make the fans run under less than 40% of the duty cycle (i.e. under 5 V).

ASUSTeK's ACPI ASOC ATK0110 virtual device provides the fan-controlling settings that are similar or identical to those outlined in the BIOS. Currently, there is no open-source implementation that supports the fan-controlling features of the device, so the fan-control interfacing has to be done through the BIOS. In general, although the settings on some of these boards may usually be adequate for many or even most users, they are still rather limited compared to the settings that individual hardware monitoring silicon chips can provide.

However, the task of exploring in great detail the fan controlling interfaces related specifically to ACPI we leave for future work, noting that our current work should provide a solid ground for any further fan controlling enhancements even in drivers other than those that we specifically mention. As we discussed above, currently ACPI does not provide enough fan-controlling capabilities for it to be interesting in our study anyways.

11.3. SpeedFan on Windows

SpeedFan is a closed-source utility for the Microsoft Windows family of operating systems that allows interested users to monitor several environmental characteristics of their personal computers. The utility provides a GUI, supports many families of hardware monitoring chipsets, and has an interface for controlling the duty cycle of the fans.

For our purposes, however, the utility has a fundamental flaw, in that it provides no interfacing for the automatic in-chip fan controlling modes. This means that what-

ever policy the user specifies in the utility, can be preserved only whilst the utility is still running, and if something happens either to the utility or the operating system, then the thermal characteristics of the system can no longer be predicted or maintained.

11.4. lm_sensors on Linux

The lm_sensors package is the most popular hardware monitoring package for the Linux kernel, supporting a variety of different hardware monitoring chips. However, the package is known to be difficult to configure even for otherwise experienced system administrators, and to our knowledge is not available on any BSD platform.

12. Hardware Monitoring Chips

In this section, we overview some of the basics regarding the hardware part of the fan control, as well as provide an outline of some interesting functionalities that popular hardware monitoring chips implement.

First, consider the fans themselves. Fans in the desktop computers are usually rated for 12 volts, although most would still run at 7 volts, where few would run when voltage is lower than 5 volts. Often, fans require higher voltage in order to start, as opposed to the voltage that would ensure that they continue running. It has also been observed that fans require higher voltage when their temperature is low, compared to the same fan when it is warm. If not accounted properly, this could have negative effects on system stability when, for example, the system is cold-started with a physically limiting fan-controlling solution, such as Zalman Fan Mate 2, set to the lowest voltage which, although sufficient for continued operation, may not be sufficient for a cold start, resulting in the fan making the clicking noises without actually ever spinning, causing a violation of the thermal characteristics of the box.

Our prototype implementation is concentrated on the Winbond Super I/O Hardware Monitors, which account for the bulk majority of the sensor chips that are available in popular motherboards; we thus describe some functionality that is available in the said chips. Our initial patch implements support for the following three families of Winbond Super I/O chips: W83627HF, W83627THF / 37HF and W83627EHF / DHG.

Many of the hardware monitoring chips feature not only the manual mode, where the duty cycle of the fans could be changed directly, but also multiple types of automatic cruise modes. In the case of Winbond, the automatic modes may include Thermal Cruise and Fan Speed Cruise, where the chip, once programmed with the set target temperature or target fan speed, internally determines what duty cycle the fans should be running at in order to satisfy the set cruising requirements.

For simplicity reasons, we have only implemented the manual mode and the thermal cruise mode in our initial prototype implementation. This is also partly due to the fact that the usefulness of the Fan Speed Cruise mode is somewhat questionable, as fans vary between each other, but within each fan it is not very likely that the speed will significantly alternate when the same voltage is given.

12.1. Shortcomings with general-purpose fan-control software

There is one problem that remains unavoidable with any general-purpose fan-controlling software: although many motherboards are in fact wired to do fan control, even if such features are not specifically advertised in motherboards' documentation or are available in the BIOS menus, some cheaper boards that do feature chips that support fan controlling simply do not have the chips wired appropriately with the fan connector headers, such that any attempts to control the speed of the fans from within the hardware monitoring chip may not be successful.

To save costs yet allow the user to still perform some fan control, some boards often feature such wiring that all the fan headers are wired and controlled through a single pin and setting of the fan-controlling chip, even if the chip itself does offer individual pins and controlling settings for more than a single fan.

Unfortunately, there is no known general approach that can reliably detect these situations in due course and with reasonably simple and straightforward logic, so the task of determining the exact peculiarities in supported fan-control functionalities of a mainboard in question are left for the end-user to establish on their own.

13. OpenBSD sysctl hw.sensors Fan-Control

In this section, we briefly describe the general notions of the OpenBSD's sysctl hardware sensors framework, and then provide some suggestions on how it can be altered such as to provide interfacing to the fan-controlling functionalities of the hardware monitoring chips.

The underlying mechanism that is used to transport the datastructures of the hardware sensors framework in OpenBSD is the sysctl interface. Currently, the framework is implemented in such a way that it allows only the drivers to export the data into the sysctl tree, but not get any feedback back from the user. However, the changes that the framework requires in order to support the functionality of passing modified sensor values back from the userland to the kernel are rather minimal, as we explore in the following paragraphs.

One of the ways to accomplish the required functionality is to allow the userland to simply pass the modified sensor data for those sensors which the drivers specifically identify as modifiable. To avoid overengineering, we can also make an assumption that only an integer value should have the possibility of being modified and passed back to the driver, as opposed to the whole sensor structure. With these assumptions, the required modifications for the framework are very straightforward and minimal, as one could see from the patch that we have released for OpenBSD and DragonFly BSD. [Murenin.tech09] [Murenin.fanctl10] The changes in the framework were made to */sys/sys/sensors.h*, */sys/kern/kern_sysctl.c* (only for OpenBSD), */sys/kern/kern_sensors.c* (only for DragonFly BSD) and *sbin/sysctl/sysctl.c*, and we briefly describe the changes below.

13.1. New *upvalue* field and new flags

The */sys/sys/sensors.h*, which is the header file with the definitions of the structures required for the framework, hereby sees the introduction of the *upvalue* field inside

of the *struct ksensor* structure, as well as two new flags, *SENSOR_FCONTROLLABLE* and *SENSOR_FNEWVALUE*.

Note that we have introduced the new *upvalue* field only into the kernel version of the sensor structure — it was deemed unnecessary to introduce the field for the userland version of the structure, since *upvalue* is only intended as the input for the drivers, and then after it is consumed by the driver, the value that the user has set would not be something that the user should be interested in monitoring. The primary reason for allowing such a discrepancy between the kernel's *struct ksensor* and userland's *struct sensor* is that, unless omitted from the userland, the introduction of a new field will cause *sizeof(struct sensor)* to grow, and would thus break the ABI, where the existing C/C++ *sysctl(3)* applications that are trying to get the *struct sensor* structure would not allocate large enough buffers for the *sysctl(3)* to copy out the structure from the kernel to the userland, returning an [ENOMEM] error message instead.

The *controllable* flag, when set by the driver, signifies that the sensor is a read/write sensor. After a new value is provided by the user, it is stored in the *upvalue* field, and the *newvalue* flag is set, which then remains set until the driver's periodic refresh procedure, which loops through all the sensors making any necessary updates, consumes the sensor's *upvalue* and clears the flag.

It deserves mentioning that the way we have designed our fan controlling prototype is such that the driver never modifies any fan-controlling settings inside the chips unless the user explicitly requests any such changes. This has several benefits, one of which is that users should not feel intimidated that the mere fact of applying the patch and rebooting the system is going to do any damage to their system. In fact, there may be situations where the user might simply want to check on the policies the motherboard manufacturer has preloaded the chips with, and as our patch not only allows one to modify the existing fan-controlling behaviour, but also to monitor

the currently applicable settings, or, at the very least, the duty cycle settings that the fans are experiencing, the user does indeed has such an option.

13.2. Sensor types

The next design decision that we discuss is that of *sensor types*. Now that the drivers could declare that certain sensors could have an *upvalue* field that could be modified and passed back into the driver, the question regarding sensor types comes to mind. On the one hand, any new sensor type would break the ABI and, possibly, API of existing utilities, whereas if the existing sensor types are reused, the interfacing may seem to look a bit too generic and somewhat less user-friendly.

For example, if we reuse the *temp* type to specify target temperatures, then those target temperature setting sensors would have to be numbered in the same namespace as those sensors that report actual temperature readings, e.g. temp0 may be the actual temperature sensor, whereas temp3 would be the (corresponding or not) read/write target temperature setting sensor.

One problem that we have found with the approach of reusing the existing sensor types is that not all types appear to be represented in the current version of *sensors.h*. For example, one of the settings that we might want the user to be able to modify is the stop, step-down and step-up time, expressed in seconds, and although there is a sensor type *timedelta*, expressed as a time fraction, it appears that the current use exclusively suggests that the value of such *timedelta* sensors should show the difference between the local wall clock and the wall clock of some external and more accurate timesource. [Balmer.Asia07] [Balmer.Euro07] Therefore, one must be careful in reuse of such sensor types, as it may inadvertently confuse tools like `ntpd`, creating a situation where it could be using such a sensor to adjust the drift of the local clock for very unintended results.

Although introducing new sensor types is very straightforward (a matter of defining each type in two places inside the */sys/sys/sensors.h*, supporting the printout in sysctl, sensorsd, systat etc, and changing the respective sensors in the drivers to the new type), the approach that we have taken thus far in our prototype implementation is to delay any such introduction, allowing us to make an interesting observation that we have managed to implement the fan-controlling interfacing via sysctl hw.sensors tree without breaking neither the existing API nor even the ABI, with the new functionality introduced exclusively on top, but not in place of, any parts of the existing framework.

13.3. Dynamic sensor descriptions

Settings for certain writable sensors may sometimes be rather complicated; for example, the duty cycle of fans may be controlled through several ways, including one manual and several automatic modes. In order to show these settings, we have conveniently used the description field of relevant sensors, where, depending on the data in certain registers, we would update the string describing an individual sensor with the information regarding some complex settings of the said sensor.

13.4. The lm(4) driver

In our prototype, we have implemented fan-controlling support for several chips that are otherwise supported by OpenBSD's lm(4) driver. A brief description of our implementation is outlined below.

First, as already mentioned, the driver does not modify the fan-controlling behaviour unless the user specifically requests such modifications via the sysctl interfacing.

Then, we tried to make the interfacing as intuitive as possible. For example, when the user modifies the duty cycle of the fans directly through the *percent* type sensors, the respective fan control settings automatically switch into the manual mode; same

happens when the user tries to change the target temperature of a given fan-controlling pin — the fan goes into the thermal cruise mode.

Table IV provides an example summary of what sensors were newly added to *lm(4)*, although the exact sensors differ with each supported family. In the example below, it is seen that the chips themselves (in this particular family) can independently control 4 fans (which is not to say that the motherboard manufacturer has necessarily wired everything to make such independent control possible). Note that all of these new sensors are both readable and writable.

Sensor	Usage
percent{0,1,2,3}	summary and duty cycle
temp{3,4,5,6}	target temperature
temp{7,8,9,10}	temperature tolerance
percent{4,5,6,7}	start-up duty cycle
percent{8,9,10,11}	stop duty cycle
indicator{0,1,2,3}	PWM/DC switch

Table IV. Newly added sensors for the W83627EHF / DHG family.

14. Demonstration

We hereby demonstrate some of the functionalities of our prototype, together with the relevant commentary.

Below is the output from a W83627DHG chip on an Intel D201GLY2 box with one small system fan. Values that are not applicable to the current operational mode are automatically marked as 'unknown' in sysctl.

```
% dmesg | fgrep W83627DHG
wbsio0 at isa0 port 0x4e/2: W83627DHG rev 0x25
lm1 at wbsio0 port 0x290/8: W83627DHG

% sysctl hw.sensors
hw.sensors.cpu0.temp0=58.00 degC
hw.sensors.lm1.temp0=45.00 degC (Sys)
hw.sensors.lm1.temp1=51.00 degC (CPU)
hw.sensors.lm1.temp2=14.50 degC (Aux)
hw.sensors.lm1.temp3=38.00 degC (Sys Target)
hw.sensors.lm1.temp4=unknown (CPU Target)
hw.sensors.lm1.temp5=unknown (Aux Target)
hw.sensors.lm1.temp6=unknown (CPU Target)
hw.sensors.lm1.temp7=2.00 degC (Sys Tolerance)
hw.sensors.lm1.temp8=unknown (CPU Tolerance)
hw.sensors.lm1.temp9=unknown (Aux Tolerance)
hw.sensors.lm1.temp10=unknown (CPU Tolerance)
hw.sensors.lm1.fan0=1854 RPM (Sys)
hw.sensors.lm1.volt0=1.34 VDC (VCore)
hw.sensors.lm1.volt1=12.20 VDC (+12V)
hw.sensors.lm1.volt2=3.33 VDC (+3.3V)
hw.sensors.lm1.volt3=3.33 VDC (+3.3V)
hw.sensors.lm1.volt4=-3.95 VDC (-12V)
hw.sensors.lm1.volt5=0.11 VDC
hw.sensors.lm1.volt6=1.62 VDC
hw.sensors.lm1.volt7=3.28 VDC (3.3VSB)
hw.sensors.lm1.volt8=0.03 VDC (VBAT)
hw.sensors.lm1.indicator0=Off (Sys Fan PWM/DC: PWM)
hw.sensors.lm1.indicator1=Off (CPU Fan PWM/DC: PWM)
hw.sensors.lm1.indicator2=Off (Aux Fan PWM/DC: PWM)
hw.sensors.lm1.indicator3=On (CPU Fan PWM/DC: DC)
hw.sensors.lm1.percent0=100.00% (Sys Fan PWM Thermal), OK
hw.sensors.lm1.percent1=100.00% (CPU Fan PWM Manual), OK
hw.sensors.lm1.percent2=100.00% (Aux Fan PWM Manual), OK
hw.sensors.lm1.percent3=100.00% (CPU Fan DC SmartIII), OK
hw.sensors.lm1.percent4=0.39% (Sys Fan Start-up Value), CRITICAL
hw.sensors.lm1.percent5=unknown (CPU Fan Start-up Value)
hw.sensors.lm1.percent6=unknown (Aux Fan Start-up Value)
```

```
hw.sensors.lm1.percent7=unknown (CPU Fan Start-up Value)
hw.sensors.lm1.percent8=29.41% (Sys Fan Stop Value), CRITICAL
hw.sensors.lm1.percent9=unknown (CPU Fan Stop Value)
hw.sensors.lm1.percent10=unknown (Aux Fan Stop Value)
hw.sensors.lm1.percent11=unknown (CPU Fan Stop Value)
```

We alter the target temperature value of the Thermal Cruise mode, and note that the percento value is going down.

```
% sudo sysctl hw.sensors.lm1.temp3=50
hw.sensors.lm1.temp3=38.00 degC {updating} (Sys Target)

% sysctl hw.sensors | fgrep Sys
hw.sensors.lm1.temp0=45.00 degC (Sys)
hw.sensors.lm1.temp3=50.00 degC (Sys Target)
hw.sensors.lm1.temp7=2.00 degC (Sys Tolerance)
hw.sensors.lm1.fan0=1739 RPM (Sys)
hw.sensors.lm1.indicator0=Off (Sys Fan PWM/DC: PWM)
hw.sensors.lm1.percent0=29.41% (Sys Fan PWM Thermal), CRITICAL
hw.sensors.lm1.percent4=0.39% (Sys Fan Start-up Value), CRITICAL
hw.sensors.lm1.percent8=29.41% (Sys Fan Stop Value), CRITICAL
```

Our D201GLY2 board is deemed abnormal, because the fan does not stop much until the duty cycle is almost zero. (Or, perhaps, the issue lies with the fan of the enclosure where the board resides.) So the result is likely to be entirely different on a different board; the status field indicates the likelihood that the fan is not going to run on a given duty cycle.

```
% sudo sysctl hw.sensors.lm1.percent8=10
hw.sensors.lm1.percent8=29.41% {updating} (Sys Fan Stop Value), CRITICAL

% sysctl hw.sensors | fgrep Sys
hw.sensors.lm1.temp0=45.00 degC (Sys)
hw.sensors.lm1.temp3=50.00 degC (Sys Target)
hw.sensors.lm1.temp7=2.00 degC (Sys Tolerance)
hw.sensors.lm1.fan0=1240 RPM (Sys)
hw.sensors.lm1.indicator0=Off (Sys Fan PWM/DC: PWM)
hw.sensors.lm1.percent0=9.80% (Sys Fan PWM Thermal), CRITICAL
hw.sensors.lm1.percent4=0.39% (Sys Fan Start-up Value), CRITICAL
hw.sensors.lm1.percent8=9.80% (Sys Fan Stop Value), CRITICAL
```

Now we go into the Manual mode. Note that the description of the percent0 sensor changes to indicate that the Manual mode becomes active, and that the value goes gradually towards the desired value over some period of time.

```
% sudo sysctl hw.sensors.lm1.percent0=6
hw.sensors.lm1.percent0=9.80% {updating} (Sys Fan PWM Thermal), CRITICAL

% sysctl hw.sensors | fgrep Sys
hw.sensors.lm1.temp0=45.00 degC (Sys)
hw.sensors.lm1.temp3=unknown (Sys Target)
hw.sensors.lm1.temp7=unknown (Sys Tolerance)
hw.sensors.lm1.fan0=1240 RPM (Sys)
hw.sensors.lm1.indicator0=Off (Sys Fan PWM/DC: PWM)
hw.sensors.lm1.percent0=9.80% (Sys Fan PWM Manual), CRITICAL
hw.sensors.lm1.percent4=unknown (Sys Fan Start-up Value)
hw.sensors.lm1.percent8=unknown (Sys Fan Stop Value)

% sysctl hw.sensors | fgrep Sys
hw.sensors.lm1.temp0=45.00 degC (Sys)
hw.sensors.lm1.temp3=unknown (Sys Target)
hw.sensors.lm1.temp7=unknown (Sys Tolerance)
hw.sensors.lm1.fan0=781 RPM (Sys)
hw.sensors.lm1.indicator0=Off (Sys Fan PWM/DC: PWM)
hw.sensors.lm1.percent0=5.88% (Sys Fan PWM Manual), CRITICAL
hw.sensors.lm1.percent4=unknown (Sys Fan Start-up Value)
hw.sensors.lm1.percent8=unknown (Sys Fan Stop Value)
```

We would like to emphasise that the driver only implements reading and writing to the registers of the chip, e.g. the Thermal Cruise mode is still performed by the chip itself. Fan Cruise mode and the Smart Fan III modes are not supported, although one can still monitor their effects via the percent{0,1,2,3} sensors.

15. Conclusion

In this thesis, we described OpenBSD's sysctl hardware sensors framework and its accompanying feature set. We surveyed the origin of the framework and the history of its development and evolution, and provided an overview of the drivers that are utilising the API.

We showed that the framework is very relevant and pervasive in OpenBSD, has been ported and committed into FreeBSD and DragonFly BSD, and remains popular and in high demand.

Certain driver code of the framework is cross-shared between NetBSD, OpenBSD, DragonFly BSD and FreeBSD. The userland interface of the framework is compatible between OpenBSD, DragonFly BSD and patched/backdated FreeBSD.

We also described some hardware monitoring and fan-controlling functionalities of modern chips and provided a prototype for OpenBSD and DragonFly BSD that allows users to conveniently interface with the fan-controlling functionalities of the commonly available hardware.

16. Future Projects

Several future projects remain possible in regards to the sensors framework and fan control. In this section, we identify some of them. Some of these identified projects may be of immediate interest to the actual users of the described systems, whereas others may be of more interest to the researchers of the subject.

16.1. Hardware support

The most obvious project, as a whole, is improving hardware support and writing more device drivers for unsupported sensor chips. Although OpenBSD has many more sensor drivers than does NetBSD, some NetBSD drivers for less popular hardware do not yet have OpenBSD equivalences. Volunteers are needed to port and test any drivers that are missing from OpenBSD, but are available in NetBSD, or which are missing from both systems.

Many sensor drivers could also be ported from OpenBSD to DragonFly BSD.

16.2. Port sensor-detect.pl from lm_sensors

The GNU/Linux lm_sensors package has a script called *sensor-detect.pl*, which scans relevant busses and tries finding the sensors that are hiding on any such busses. It might be a worthwhile project to provide some wrapper utilities for the script, such that the script could be used on OpenBSD or other BSD platforms to identify which (previously unknown) sensor devices are available in the hardware, such that any missing drivers could be written or cross-ported.

16.3. Port i2c_scan.c to other BSDs

Another possible project includes the porting of the i2c_scan.c functionality to other BSD systems, most interestingly the FreeBSD / DragonFly BSD APIs. This would allow a huge number of I^2C drivers to be cross-ported (as well as for all unsupported

I^2C devices to be promptly identified in the future), once the i2c_scan.c porting itself is accomplished.

16.4. Further improve sensorsd

The sensorsd(8) sensor monitoring daemon has been greatly improved since its introduction, but it is still not as flexible as some comparable utilities are, as far as extended functionality and the configuration language are concerned. It would be an interesting project to design a configuration language for sensorsd similar to the one used in OpenBSD's Packet Filter firewall rulesets. Additional monitoring features may also be introduced to sensorsd, such that it would be possible to detect more anomalies on those sensors whose drivers are not keeping up their state, or where such state might still require additional attention from the user.

16.5. Interfacing for fan-speed controlling

Fan-speed controlling was discussed as a part of this thesis, and a prototype has been provided, however, further research is possible in several distinct directions.

In general, as we have shown, OpenBSD's sensors framework requires very little amount of modification to provide an interface for the ability to conveniently pass values from sysctl(8) back into the driver, such that the driver, in turn, could pass such values down to the chip, for the chip to modify the voltage of some fan headers in a certain predetermined fashion.

However, different generations of chips have different logic regarding fan control; many chips of recent generations have multiple temperature levels at which different fan speeds could be sought; certain temperature sensors could be specified to affect decisions regarding the speed of certain fans etc. Concerns for simplicity extinction are amplified by the fact that the majority of motherboards are miswired as far as hardware monitoring datasheets are concerned, since many modern hardware monitoring chips oftentimes provide way much more functionality in regards to fan con-

trolling than the motherboard manufacturer is usually interested in supporting and advertising in its products for its endusers. Therefore, a complete, flexible and round patch for supporting fan controlling functionality might be a long way from OpenBSD's philosophy of being a system where a great deal of effort is paid towards the simplicity and generality of its feature set.

Due to these reasons, it is unclear if any general-purpose fan-controlling prototype will ever be integrated into the main release of the OpenBSD system, so some further research is warranted.

16.6. Possible race conditions between user software and the BIOS

It has also been speculated that user-initiated intervention with the fan-controlling functionality of the chip may cause undesirable consequences to the stability of the system as a result of certain conflicts with the system management code of the BIOS. Although the concern has some grounds, in our experience no undesirable interactions were found as of yet in regards to the matter. Future work may examine assembly code and, perhaps, specifications of various hardware components to determine if the concerns have some more valid grounds.

Motherboard manufacturers may wish to provide fan-controlling specifications through custom ACPI devices, whereby the specifications would be more definitive and less likely to harm the stability and design of the system, at the same time ensuring that each BIOS contains the relevant information of which pins of the chip are actually utilised in the design of the motherboard.

16.7. Fan-controlling through sensorsd

Future work may also be done in regards to a simplified language for specifying various relationships for fan control, and the language may feature fallbacks for the

sensorsd hardware monitoring daemon for those chips that cannot do the monitoring loop by themselves.

References

[Andrews.KernelTrap08] Jeremy Andrews. “BSDCan 2008: Hardware Sensors Framework”. KernelTrap. 7 June 2008.
http://kerneltrap.org/OpenBSD/BSDCan_2008_Hardware_Sensors_Framework

[Balmer.Asia07] Marc Balmer. “Support for Radio Clocks in OpenBSD”. In: AsiaBSDCon 2007 Proceedings. 8–11 March 2007, Tokyo, Japan.
http://www.openbsd.org/papers/radio-clocks-asiabsdcon07.pdf

[Balmer.Euro07] Marc Balmer. “Supporting Radio Clocks in OpenBSD”. On: EuroBSDCon 2007. 12–15 September 2007, Copenhagen, Denmark. Slides:
http://www.openbsd.org/papers/eurobsdcon07/mbalmer-radio_clocks.pdf

[Biancuzzi.42] Federico Biancuzzi. “Puffy’s Marathon: What’s New in OpenBSD 4.2”. O’Reilly ONLamp. 01 November 2007. http://onlamp.com/lpt/a/7155

[Biancuzzi.44] Federico Biancuzzi. “Source Wars – Return of the Puffy: What's New in OpenBSD 4.4”. O’Reilly Community. 3 November 2008.
http://broadcast.oreilly.com/2008/11/source-wars---return-of-the-pu.html

[Chou01] Andy Chou et al. “An empirical study of operating systems errors”. In: Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles. 21–24 October 2001, Banff, Alberta, Canada. ACM SOSP 2001, pp. 73–88. doi:10.1145/502034.502042

[deRaadt.misc98] Theo de Raadt. “See: dmesglog works”. misc@openbsd.org mailing list. 12 November 1998. http://marc.info/?l=openbsd-misc&m=91090366422103&w=2

[deRaadt.priv06] Theo de Raadt. Private emails. 2006.

[deRaadt.zdnet06] Ingrid Marson. “OpenBSD 3.9 adds sensor framework”. ZDNet UK. 24 March 2006, London, UK. http://news.zdnet.co.uk/software/,,39259254,.htm

[deRaadt.jedec08] Theo de Raadt. “New sensor driver, sdtemp(4)”. misc@openbsd.org mailing list. 12 April 2008. http://marc.info/?l=openbsd-misc&m=120804067607451&w=2

[Evans.stress] Gary W. Evans and Dana Johnson. “Stress and Open-Office Noise”. Journal of Applied Psychology, vol. 85, no. 5, pp. 779–783. October 2000. doi:10.1037/0021-9010.85.5.779

[FreeBSD.ideas] —. “The FreeBSD list of projects and ideas for volunteers”. FreeBSD.
http://www.freebsd.org/projects/ideas/

[Google.FAST07] Eduardo Pinheiro, Wolf-Dietrich Weber and Luiz Andre Barroso. “Failure Trends in a Large Disk Drive Population”. Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST’07). February 2007, San Jose, CA, USA.
http://labs.google.com/papers/disk_failures.pdf

[Halderman.Lest] J. Alex Halderman et al. “Lest We Remember: Cold Boot Attacks on Encryption Keys”. 17th USENIX Security Symposium (USENIX Security '08). July 2008, San Jose, CA, USA. http://citp.princeton.edu/memory/

[Intel.ACPI] —. “Advanced Configuration and Power Interface”. Hewlett-Packard, Intel, Microsoft, Phoenix and Toshiba. http://www.acpi.info/

[Intel.acpica] —. “ACPI Component Architecture”. Intel. http://www.acpica.org/

[Intel.D201GLY2] —. “Intel® Desktop Board D201GLY2 / D201GLY2A”. Intel. http://www.intel.com/products/motherboard/D201GLY2/configs.htm

[Intel.SMBus20] —. “System Management Bus (SMBus) Specification, Version 2.0”. Intel et al. 3 August 2000. http://smbus.org/specs/smbus20.pdf

[grange.priv05] Alexander Yurchenko. Private emails. June 2005.

[Gwynne.Open06] David Gwynne and Marco Peereboom. “Bio and Sensors in OpenBSD”. On: OpenCon 2006 – The OpenBSD Conference. 2–3 December 2006, Venice, Italy. Slides: http://www.openbsd.org/papers/opencon06-bio.pdf

[lm_sensors.ThinkPad] —. “README.thinkpad”. lm_sensors. 2001/2004. http://www.lm-sensors.org/browser/lm-sensors/trunk/README.thinkpad?rev=5132

[Murenin.UKUUG06] Constantine A. Murenin. “Hardware temperature monitoring device drivers for OpenBSD”. In: UKUUG Spring Conference and Tutorials: Conference Proceedings. 21–23 March 2006, Durham, UK.

[Murenin.BSc06] Constantine A. Murenin, B. Sc. (Hons) Final Year Project Main Report: “Microprocessor system hardware monitors. Interfacing on OpenBSD with sysctl(8).” Faculty of Computing Sciences and Engineering, De Montfort University, Leicester, UK, May 2006.

[Murenin.TOJ06] Constantine A. Murenin. “New two-level sensor API”. The OpenBSD Journal. 30 December 2006. http://undeadly.org/cgi?action=article&sid=20061230235005

[Murenin.IEEE07] Constantine A. Murenin. “Generalised Interfacing with Microprocessor System Hardware Monitors”. In: Proceedings of 2007 IEEE International Conference on Networking, Sensing and Control. 15–17 April 2007, London, United Kingdom. IEEE ICNSC 2007, pp. 901—906. doi:10.1109/ICNSC.2007.372901

[Murenin.GSoC07.prop] Constantine A. Murenin. “Unified Hardware Monitoring Interface for FreeBSD. (Port OpenBSD’s sysctl Hardware Sensors Framework)”. 6 April 2007. http://mojo.ru/us/GSoC2007.FreeBSD.cnst-sensors.proposal.html

[Murenin.GSoC07.fin] Constantine A. Murenin. “GSoC2007: cnst-sensors.2007-09-13.patch”. freebsd-hackers@freebsd.org mailing list. 13 September 2007. http://lists.freebsd.org/pipermail/freebsd-hackers/2007-September/021722.html

[Murenin.FQSR07] Constantine A. Murenin, Shteryana Shopova. “Porting OpenBSD’s sysctl Hardware Sensors Framework to FreeBSD”. FreeBSD Quarterly Status Report, July to October 2007. http://www.freebsd.org/news/status/report-2007-07-2007-10.html

[Murenin.GSoC07.sum] Constantine A. Murenin. “GSoC2007/cnst-sensors”. FreeBSD. 14 October 2007. http://wiki.freebsd.org/GSoC2007/cnst-sensors

[Murenin.TOJ07.wbng] Constantine A. Murenin. “Developer blog: cnst@: wbng(4) and how it was written”. The OpenBSD Journal. 29 October 2007. http://undeadly.org/cgi?action=article&sid=20071029080000

[Murenin.Can08] Constantine A. Murenin. “OpenBSD Hardware Sensors Framework”. On: BSDCan 2008 – The BSD Conference, Invited Talks track. 14–17 May 2008, Ottawa, Ontario, Canada. Abstract: http://www.bsdcan.org/2008/schedule/events/63.en.html Slides: http://www.openbsd.org/papers/bsdcan08-sensors.pdf

[Murenin.Login08] Constantine A. Murenin. “OpenBSD Hardware Sensors Framework”, “X.Org”, “BSD licensed C++ compiler”. In: Conference Reports, BSDCan: The BSD Conference. USENIX ;login:, August 2008, Volume 33, Number 4, pp. 113–114. http://www.usenix.org/publications/login/2008-08/index.html

[Murenin.Euro08] Constantine A. Murenin. “OpenBSD Hardware Sensors Framework”. On: EuroBSDCon 2008 – The 7th European BSD Conference. 16–19 October 2008, Strasbourg, France. Slides: http://www.openbsd.org/papers/eurobsdcon2008-sensors.pdf

[Murenin.Asia09] Constantine A. Murenin and Raouf Boutaba. “OpenBSD Hardware Sensors Framework”. In: AsiaBSDCon 2009 Proceedings. 12–15 March 2009, Tokyo University of Science, Tokyo, Japan.
Paper: http://www.openbsd.org/papers/asiabsdcon2009-sensors-paper.pdf
Video: http://www.youtube.com/watch?v=TwcWU626TzI

[Murenin.tech09] Constantine A. Murenin. “sysctl hw.sensors lm(4) fan-controlling prototype/hack”. tech@openbsd.org mailing list. 8 May 2009. http://sensors.cnst.su/fanctl/tech@openbsd.org.2009-05-08.fanctl.patch.eml

[Murenin.Can09] Constantine A. Murenin. “Quiet Computing with BSD”. On: BSDCan 2009 – The BSD Conference, Hacking Talks track. 6–9 May 2009, Ottawa, Ontario, Canada.
Abstract: http://www.bsdcan.org/2009/schedule/events/119.en.html
Slides: http://sensors.cnst.su/fanctl/BSDCan2009.cnst-fanctl.slides.pdf
Prototype: http://sensors.cnst.su/fanctl/tech@openbsd.org.2009-05-08.fanctl.patch.eml

[Murenin.techkern09] Constantine A. Murenin. “Re: aibs(4): ASUSTeK AI Booster (ACPI ATK0110) hardware monitor with limit support”. tech-kern@netbsd.org mailing list. 31 December 2009. http://mail-index.netbsd.org/tech-kern/2009/12/31/msg006778.html

[Murenin.fanctl10] Constantine A. Murenin. “[PATCH] fanctl: AsiaBSDCon 2010 DragonFly sysctl hw.sensors lm(4) fan control similar to BSDCan 2009 OpenBSD”. 12 March 2010. http://sensors.cnst.su/fanctl/AsiaBSDCon2010.cnst-fanctl.dfly.patch

[Murenin.Asia10] Constantine A. Murenin and Raouf Boutaba. “Quiet Computing with BSD”. In: AsiaBSDCon 2010 Proceedings. 11–14 March 2010, Tokyo University of Science, Tokyo, Japan.
Paper: http://sensors.cnst.su/fanctl/AsiaBSDCon2010.cnst-fanctl.paper.pdf
Website: http://sensors.cnst.su/fanctl/

[phk.arch07] Poul-Henning Kamp. “Please think architecture...”. freebsd-arch@freebsd.org mailing list. 29 August 2007.
http://lists.freebsd.org/pipermail/freebsd-arch/2007-August/006763.html

[phk.gsocgood07] Poul-Henning Kamp. “Re: cvs commit: src/etc Makefile sensorsd.conf ...”. cvs-src@freebsd.org mailing list. 14 October 2007.
http://lists.freebsd.org/pipermail/cvs-src/2007-October/082407.html

[Tanenbaum06] Andrew S. Tanenbaum, Jorrit N. Herder and Herbert Bos. “Can We Make Operating Systems Reliable and Secure?”. IEEE Computer, vol. 39, no. 5, pp. 44–51. May 2006. doi:10.1109/MC.2006.156

[Tepper.submit07] Hasso Tepper. “Hardware sensors framework and some drivers using it”. submit@dragonflybsd.org mailing list. 25 September 2007.
http://leaf.dragonflybsd.org/mailarchive/submit/2007-09/msg00020.html

[Tepper.priv07] Hasso Tepper. Private email. 25 September 2007.

[Theile.arch07] Volker Theile, Alexander Leidinger, LI Xin, Bruno Ducrot. “Any plans to implement OpenBSD sensor framework into FreeBSD?”. freebsd-arch@freebsd.org mailing list. January 2007. http://lists.freebsd.org/pipermail/freebsd-arch/2007-January/006048.html

[thinkwiki] —. “How to control fan speed”. ThinkWiki.
http://www.thinkwiki.org/wiki/How_to_control_fan_speed

[Watanabe.ACPI] Takanori Watanabe. “ACPI implementation on FreeBSD”. 2002 USENIX Annual Technical Conference, FREENIX Track. 10–15 June 2002, Monterey, CA, USA.