# OPENBSD HARDWARE SENSORS FRAMEWORK

*A unified and ready-to-use system for hassle-free hardware monitoring.*

Constantine A. Murenin and Raouf Boutaba
University of Waterloo

*AsiaBSDCon 2009 — 12–15 March 2009 — Tokyo, Japan*

## Abstract

In this paper, we will discuss the origin, history, design guidelines, API and the device drivers 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 70 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.

## 1. Introduction

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

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 whose main intention is to deliver 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].

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] 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 or a low-frequency radio receiver). The ntpd(8) daemon uses these timedelta sensors to correct the local clock, with the aim to ensure 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 external trusted 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 1. List of sensor types.*

As a summary, it has been found that all of these sensor-like data (see Table 1 for the complete list) can be aggregated in a single and straightforward interface, which we will be describing in the following sections. Needless to say, the monitoring of many of these environmental sensors can predict and diagnose system failure. [Gwynne.Open06] We will try to argue that OpenBSD's interface is much 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.

### 1.1. 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 was judged as not being a positive approach.

To illustrate this example of lacking or incomplete documentation, let us consider how voltage sensors work. When we read a value from 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 2 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 2. 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]

## 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 2009), 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. We will now describe 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", "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 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 the next. 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 other hand, if the driver does know the state of its sensors (for example, as is the case with IPMI), then those states may be one of *OK*, *WARN*, *CRIT* or *UNKNOWN*.

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

For the list of sensor types, please refer to Table 1, or the source code.

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, in the next chapter we will continue to refer to these structures from the userland perspective of *sensor* and *sensordev*, even if the kernel structures are the ones we're talking about.

### 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 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 thing that the driver must do to make its whole tree of sensors available system-wide is call *sensordev_install()*. The driver may abort anytime 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.

## 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* display 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 you 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). For details, 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 moni-

tor and report the changes in sensors states on those sensors that keep their state (for example, as is the case with IPMI, ESM 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/.

## 4. Sensor Drivers

In this section, we will try to 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, rest assured that majority of the drivers are actually enabled in most GENERIC kernels in OpenBSD, thus we deem that such a comparison is still reasonable.

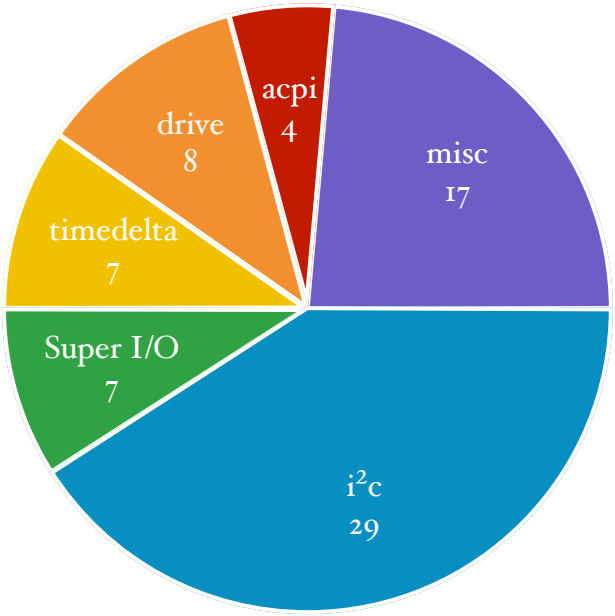


*Chart I. Number of sensor device drivers in OpenBSD 4.5 by primary category.*

In general, the drivers can be divided into the following categories: Super I/O hardware monitors, 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 can be evidenced from Chart I. Note that in this chart we had to put some drivers into the general miscellaneous category for simplicity, which include IPMI and various other embedded sensors.

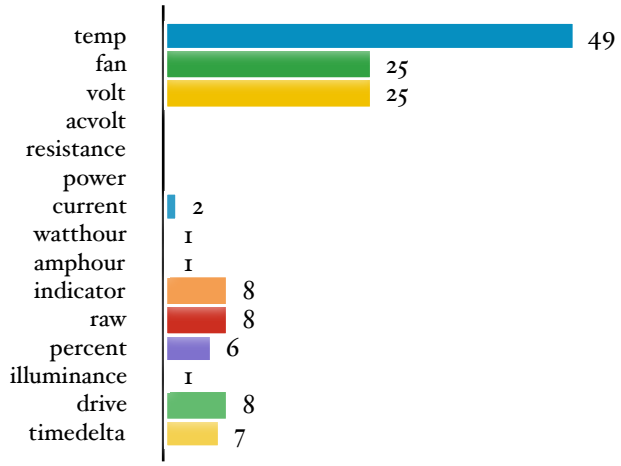


*Chart II. Sensor type popularity in OpenBSD 4.5 based on the number of drivers using each type.*

OpenBSD 4.5 contains 72 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). We note that temperature sensor type is by far the most popular (used by 49 out of the 72 drivers), with the fan and voltage sensors having a draw for the next most popular type (each is used by 25 out of the 72 drivers).

## 5. I²C Sensors and Bus Scan

In this section, we will 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 doesn't have a standard method of detecting what devices appear at which addresses. 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 their 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 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

However, 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 I²C 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 be the chips that didn't fully adhered to the I²C standard; however, it's 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 proof that such care has been very adequate.

In a nutshell, all I²C 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 support it, meaning that in the vast majority of times there is absolutely no need for the end-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 I²C sensor drivers are enabled in the GENERIC kernels on OpenBSD and work out of the box on all supported architectures.

### 5.3. I²C register dumps

As explained earlier, the i2c_scan.c mechanism is run automatically (during the kernel boot time) on all architectures that it supports. 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 the dmesg, the most often occurring register value is reduced from the dump before it 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 hard-

ware that the users posses. 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 and variations of the hardware.

### 5.4. 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 we want to make sure that we don't write anything to its registers unless absolutely necessary. [deRaadt.priv06]) Therefore, if we have a full selection of register values, we can then simulate the I²C bus in the userland and test much of the functionality of the driver without even booting a new kernel or having the required hardware available at our disposal. [Murenin.TOJ07.wbng]

To do such simulation, we have to implement several small functions and copy some other functions from the kernel. First, we have to 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're 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 don't 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 our 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, we have to adjust several of its functions. Apart from removing *splhigh()* and *splx()* calls and some extra *#include*s, we have to reimplement the *sensor_task_register()* routine. All we need it 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 I²C sensor driver that meets the criteria of not doing any unconditional writes to any registers can be taken directly from the kernel to our 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 our example).

Having such a sandbox environment streamlines I²C driver development and initial testing. The wbng(4) and andl(4) represent the two drivers that have been developed in the said sandbox and first tested against several I²C 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 will try to 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 sensors 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 III 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 the upcoming OpenBSD 4.5 (2009) will have 72 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].

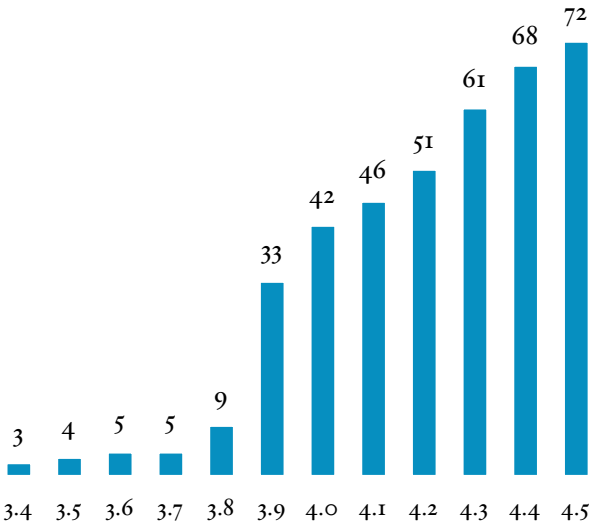


*Chart III. Number of drivers using the sensors framework from OpenBSD 3.4 to 4.5.*

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 invoca-

tion 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

| OpenBSD | Exposing sensors |
|---|---|
| 3.4 | extern struct sensors_head sensors;<br>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 4.5.*

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.

## 7. Related Frameworks

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

As a 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 competing products to this day. [deRaadt.jedec08] [sdtemp.4] [Biancuzzi.44]

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

The total number of drivers in the latest development version of NetBSD as of February 2009, calculated by the number of calls to the *sysmon_envsys_register()* function, is only 32, versus 72 as the respective count in the latest version of OpenBSD. 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).

In 2007, a new version of the envsys 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 didn't support detachable sensors. On OpenBSD, detachable sensors have been supported since January 2006.

Sensor types are roughly the same between NetBSD and OpenBSD. Sensors of *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 February 2009.

### 7.2. 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 application was submitted in the last day that Google was accepting applications for GSoC2007, since the suggestion was found by chance; nonetheless, the proposal [Murenin.GSoC07.prop] has been voted up by the FreeBSD committers to be approved for a funding slot, and was subsequently selected and funded by Google.

### 8.1. Summer of Code 2007

During Google Summer of Code 2007, all relevant parts of the framework that were promised 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 didn't 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]

Google funding also allowed Constantine to fix several not very related bugs in both FreeBSD and OpenBSD, which included a 10-year-old pointer-arithmetic bug in OpenBSD's make/job.c [Murenin.makej07] [Murenin.TOJ07.make] and a 12-year-old permission validation bug in FreeBSD's sys/kern/kern_sysctl.c [Murenin.sysctl07].

Summer of code experience was specifically pleasant thanks to Shteryana Shopova *(syrinx)*, Alexander Leidinger *(netchild)*, Rui Paulo *(rpaulo)*, Robert Watson *(rwatson)*, Sam Leffler *(sam)* and many other users and developers who have provided useful suggestions and feedback, and were a pleasure to work with.

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]; this took Constantine somewhat by surprise, as he was himself contemplating doing the said port.

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 didn't solve the niche in the FreeBSD-way. [phk.gsocgood07] Poul-Henning Kamp emphasised that he doesn't 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]

## 9. Conclusion

In this paper, we have described OpenBSD's sysctl hardware sensors framework and its accompanying feature set. We have 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 have shown 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.

## 10. Future Projects

Several future projects remain possible in regards to the sensors framework. In this section, we will try to identify some of them.

### 10.1. 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.

### 10.2. 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.

### 10.3. 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.

### 10.4. 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.

### 10.5. Fan-speed controlling

Fan-speed controlling was the reason that Constantine A. Murenin got originally interested in the sensors framework back in 2005. Whilst the topic is interesting, many obstacles are present.

In general, 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. In fact, a prototype patch for supporting exactly this functionality has been already developed as a part of a bigger project in 2006. [Murenin.BSc06] [Murenin.IEEE07]

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 controlling 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 simplicity and generality of its feature set.

## 11. 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.

This document is a second public revision of the paper that has been written exclusively for AsiaBSDCon 2009 (March 2009, Tokyo, Japan) and was published in the official proceedings; however, the talk itself has been also presented earlier at EuroBSDCon 2008 in Strasbourg, France in October 2008 [Murenin.Euro08], and at BSDCan 2008 in Ottawa, Ontario, Canada in May 2008 *(Invited Talk)* [Murenin.Can08] (both of these previous presentations lacked a formal paper). The BSDCan 2008 presentation and audience participation have been autosummarised in *;login: The Usenix Magazine* issue from August 2008 [Murenin.Login08], as well as summarised by the KernelTrap editor Jeremy Andrews in the KernelTrap computing news web-site [Andrews.KernelTrap08].

## 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

[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

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

[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.makej07] Constantine A. Murenin. "10-year-old pointer-arithmetic bug in make(1) is now gone, thanks to malloc.conf and some debugging". LiveJournal. 12 June 2007. http://cnst.livejournal.com/24040.html

[Murenin.TOJ07.make] Constantine A. Murenin. "Developer blog: cnst@: fixing make". The OpenBSD Journal. 19 June 2007.
http://undeadly.org/cgi?action=article&sid=20070619104027

[Murenin.sysctl07] Constantine A. Murenin. "12-year-old bug in FreeBSD's kern_sysctl.c". LiveJournal. 03 September 2007. http://cnst.livejournal.com/37740.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. 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.

[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

[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

[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

## Biography

***Constantine A. Murenin*** is an MMath graduate student at the David R. Cheriton School of Computer Science at the University of Waterloo (CA). Prior to his graduate appointment, Constantine attended and subsequently graduated from East Carolina University (US) and De Montfort University (UK), receiving two Bachelor of Science degrees in Computer Science, with Honors and Honours, in 2007 and 2006, respectively. An OpenBSD Committer, FreeBSD Google Summer of Code Student Alumnus and Mozilla Contributor, Constantine's interests range from standards compliance and usability at all levels, to quiet computing and hardware monitoring. You can contact him via email at <*cnst@openbsd.org*>.

***Raouf Boutaba*** received the MSc and PhD degrees in Computer Science from the University of Paris 6, Paris, in 1990 and 1994, respectively. He is currently a Professor of Computer Science at the University of Waterloo. His research interests include network, resource and service management in wired and wireless networks. He is the founder and Editor-in-Chief of the IEEE Transactions on Network and Service Management and on the editorial boards of several other journals. He is currently a distinguished lecturer of the IEEE Communications Society, the chairman of the IEEE Technical Committee on Information Infrastructure and the Director of the ComSoc Related Societies Board. He has received several best paper awards and other recognitions such as the Premier's research excellence award.