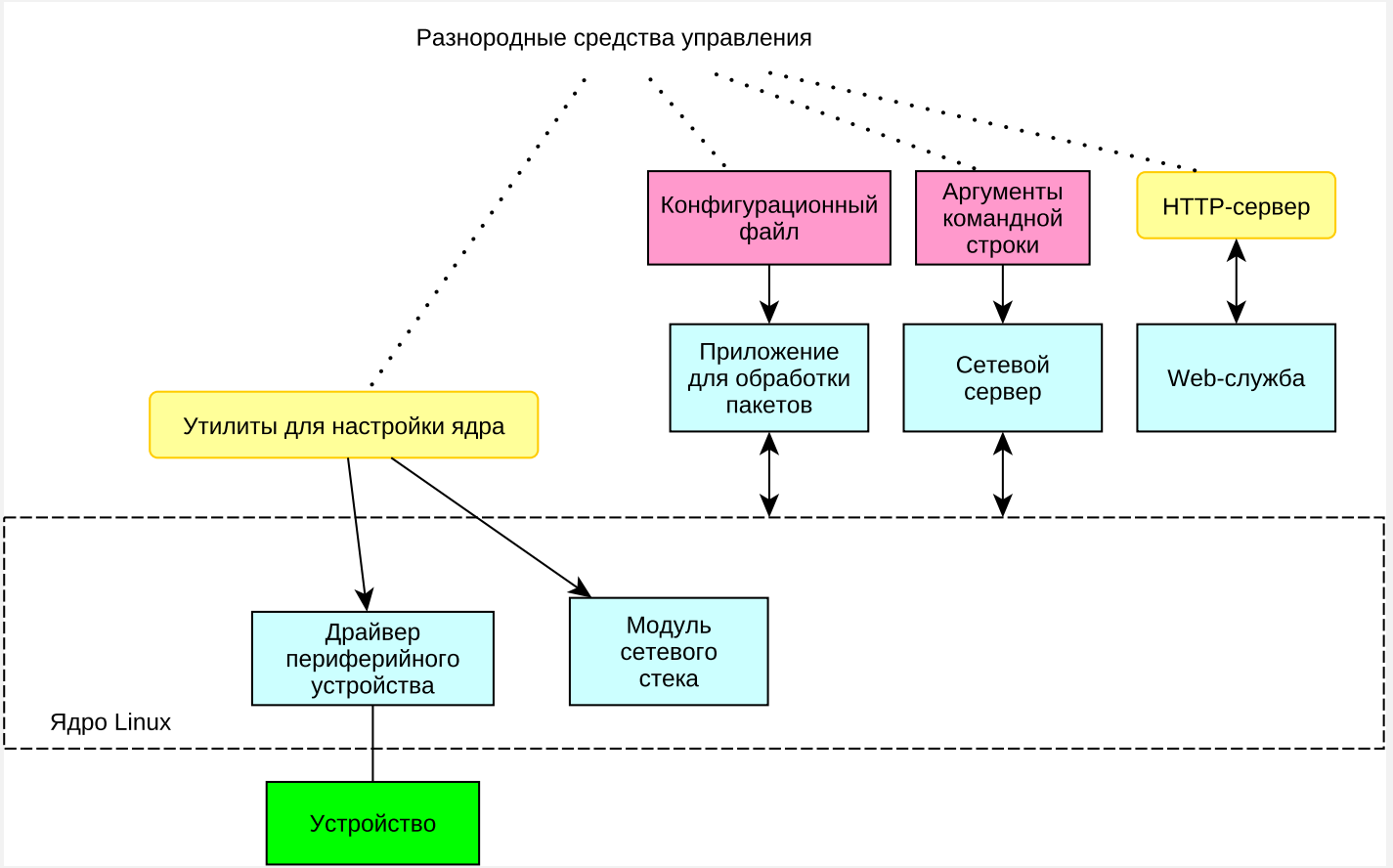
ПО сетевых устройств

Трещановский Павел Александрович, к.т.н.

09.06.20

Управляемые объекты в устройстве



Программно-конфигурируемые сети

Современные сети (в частности, сети IoT) - больше, чем совокупность составляющих их устройств.

Программно-конфигурируемая сеть (SDN - Software-Defined Network) - концепция построения и администрирования сети, основанная на автоматическом централизованном управлении всеми ресурсами сети за счет представления этой сети в виде единого программируемого объекта.

Контроллер сети - объект, отвечающий за управление сетью SDN.

О каком управлении идет речь?

- Конфигурирование - приведение поведения устройства в соответствие с некоторым документом или структурой данных.
- Мониторинг.
  - Чтение статуса - текущее состояние системы (наличие соединений, состояние конечных автоматов и пр.).
  - Чтение статистики - счетчики переданных пакетов, ошибок, отвергнутых соединений и пр.
  - События - автономные сообщения об изменении состояния системы.

Пример

- Конфигурирование - задание IP-адреса:

```
# ip addr add dev eth0 192.168.0.8/24
```

- Чтение статуса - наличие (state UP) или отсутствие соединения.

```
# ip link
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master lxcbr0
state UP mode DEFAULT group default qlen 1000
    link/ether 88:d7:f6:7f:98:84 brd ff:ff:ff:ff:ff:ff
```

- Чтение статистики.

```
ip -s link show dev enp7s0
...
RX: bytes  packets  errors  dropped overrun mcast
60078028   90936    0       0       0       10607
```

Традиционный подход к управлению сетевыми устройствами

- Специальный „Cisco-подобный” CLI.
- Web-интерфейс.
- CLI операционной системы (shell), внутренние конфигурационные файлы.

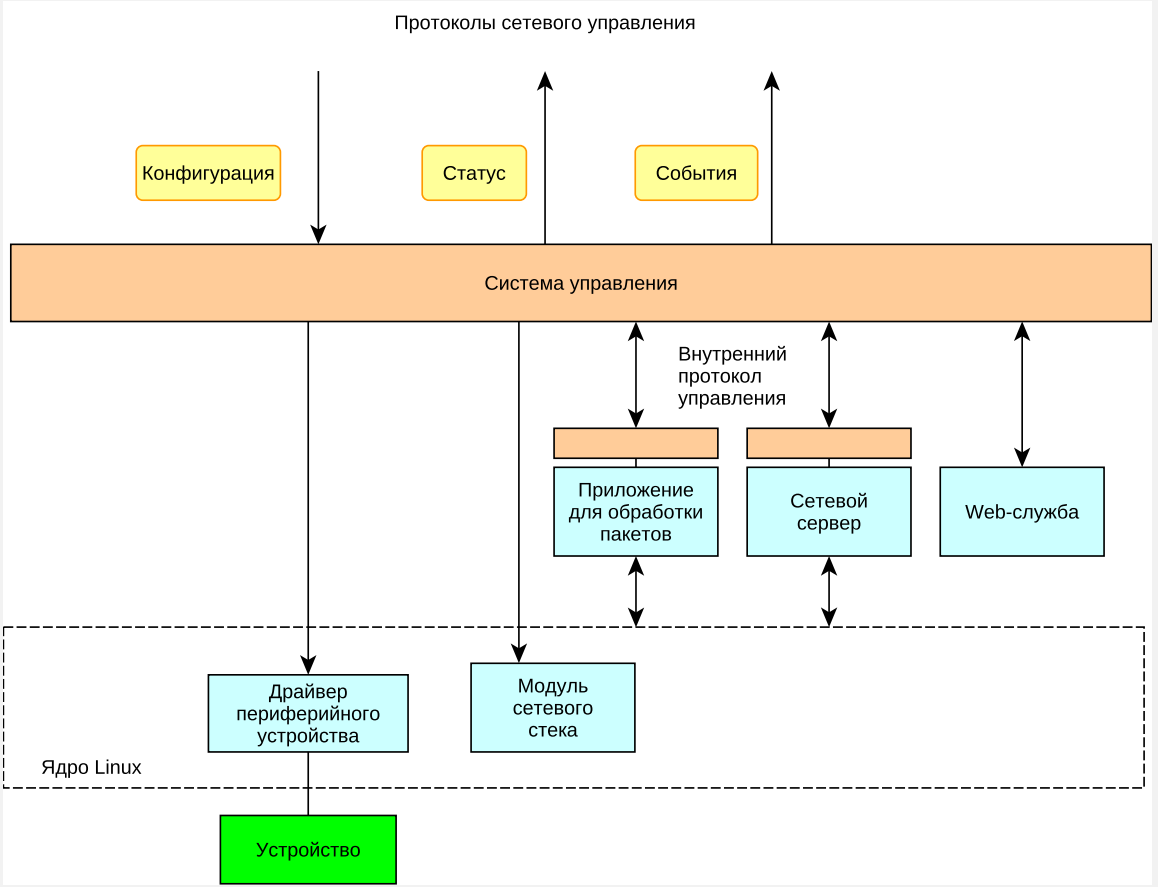
Концепцию SDN в рамках традиционного подхода реализовать тяжело.

- „Хрупкие” API - вывод команд и набор аргументов не стандартизован, отличия между производителями и версиями ПО.
- Не предусмотрена автоматическая обработка исключительных ситуаций. Например, если из 10 команд одна выполнилась неуспешно. В каком состоянии в итоге окажется система?
- Ограниченные возможности - автономная отправка событий не предусмотрена.

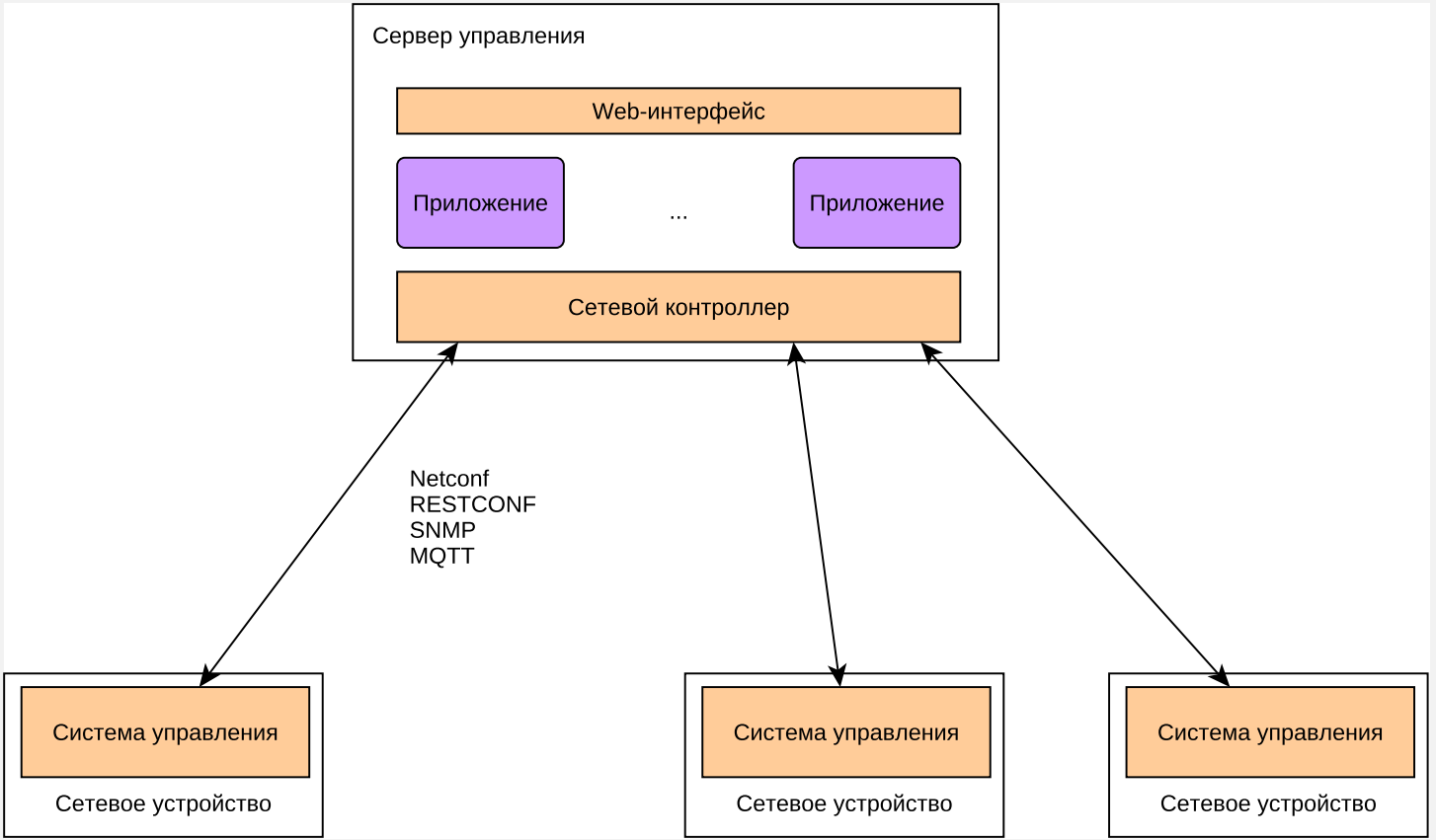
Потенциальные применения концепции SDN

- Масштабная автоматизация телекоммуникационных сетей.
- Динамическое распределение пропускной способности на основе параметров радио-каналов, загруженности сети и др..
- Автоматическое управление сетью ЦОДа при создании или удалении виртуальных машин.
- Внедрение технологий „умного” дома (города, предприятия) предполагает наличие программируемой сети.

Система управления устройством



Контроллер сети



Документ JSON

```
{
	"netowrk" : {
		"interfaces" : {
			"eth0" : {
				"enable" : true,
				"address" : "192.168.0.5",
				"prefix-length" : "28",
				"traffic-types" : ["multicast"]
			},
			"eth1" : {
				"enable" : false,
				"address" : "192.168.1.10",
				"prefix-length" : 22,
				"traffic-types" : ["unicast", "multicast"],
				"mtu" : 1400,
				"state" : {
					"used-by-process" : []
				}
			}
		}
	}
}
```

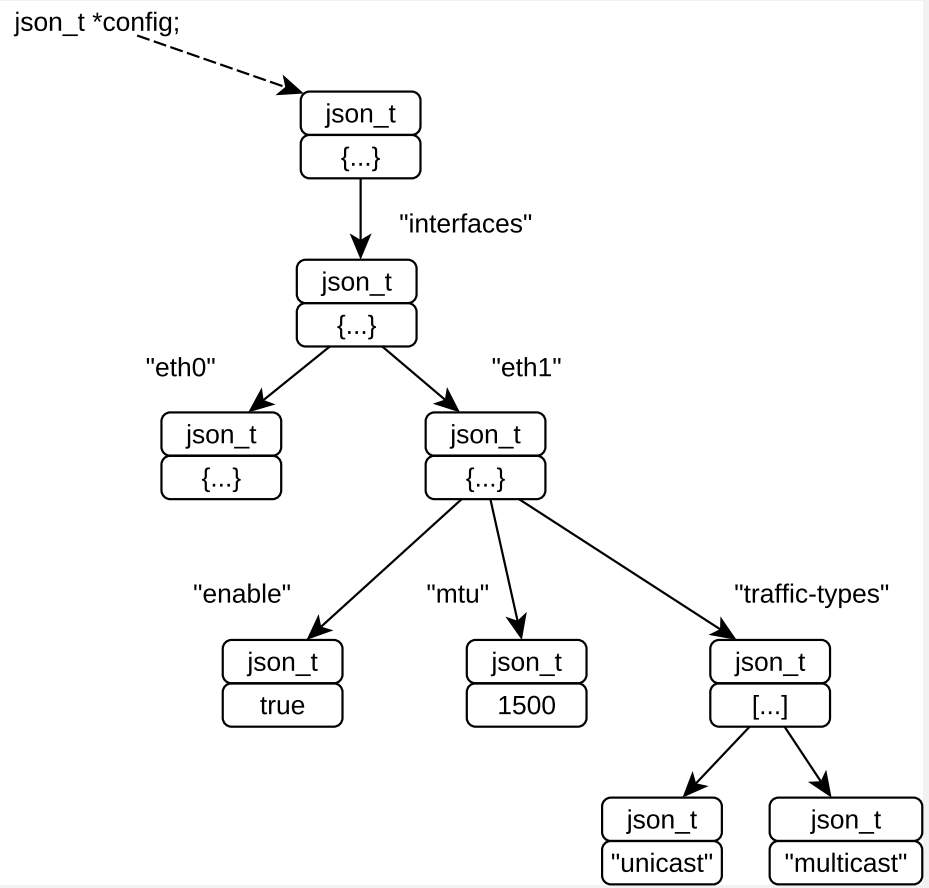
Терминология и типы данных

- Формат представления структурированных данных.
- JSON - JavaScript Object Notation, подмножество языка JavaScript.
- Скалярные значения: целые числа (без кавычек), строки (с кавычками), булевы значения.
- *Объект* - контейнер для произвольных элементов JSON, индексируемый строкой. Примерно соответствует словарю `struct avl_tree`.
- *Массив* - упорядоченная последовательность произвольных элементов JSON. Примерно соответствует списку `struct list_head`.
- Библиотека на C - jansson.

Особенности

- JSON обеспечивает сериализацию и десериализацию структурированных данных в сетевых протоколах, конфигурационных файлах и т.д.
- Самоописывающий формат - информация о структуре документа хранится в самом документе (, [] и пр.). Упрощает создание расширяемых протоколов с возможностью избирательной совместимости в каждой частной реализации.
- Свободная структура документа - определение множества допустимых документов возлагается на приложение. Для сравнения - в SQL это множество определяется *схемой* базы данных.
- Человекочитаемый формат. Можно просматривать и модифицировать в текстовом редакторе.

Представление документа в виде дерева `json_t`



Чтение и вывод документа

```
int main(int argc, char *argv[])
{
	json_t *jconf;
	json_error_t jerr;
	char *str;

	/* json_t *json_load_file(const char *path, size_t flags, json_error_t *err) */
	jconf = json_load_file("configuration.json", 0, &jerr);
	if (!jsonf)
		goto on_error;

	/* char *json_dumps(const json_t *json, size_t flags); */
	/* JSON_ENCODE_ANY - корневой элемент не обязательно является объектом,
	 * JSON_INDENT - форматированный вывод с заданной шириной отступов. */
	str = json_dumps(jconf, JSON_ENCODE_ANY | JSON_INDENT(3));
	if (!str)
		goto on_error;

	printf("Document:\n%s\n", str);
	free(str);
	json_decref(jconf);
}
```

Подробная информация об ошибках

```
typedef struct json_error_t {
       int line; /* Строка, на которой возникла ошибка. */
       int column; /* Столбец, на котором возникла ошибка. */
       char text[JSON_ERROR_TEXT_LENGTH]; /* Описание ошибки. */
       ...
} json_error_t;

int main(int argc, char *argv[])
{
       json_t *jconf;
       json_error_t jerr;

       jconf = json_load_file("configuration.json", 0, &jerr);
       if (!jconf) {
              fprintf(stderr, "Failed to load configuration: "
                            "%s (line %d, column %d)\n",
                            jerr.text, jerr.line, jerr.column);
              goto on_error;
       }
       json_decref(jconf);
}
```

Типы json_t

```
typedef enum {
	JSON_OBJECT, JSON_ARRAY, JSON_STRING, JSON_INTEGER,
	JSON_TRUE, JSON_FALSE, ...
} json_type;

int main(int argc, char *argv[])
{
	json_t *jdata = ...;
	json_type type;
	type = json_typeof(jdata);

	if (json_is_object(jdata))
		/* Использование функций json_object_*. */
	ellse if (json_is_array(jdata))
		/* Использование функций json_array_*. */
	ellse if (json_is_string(jdata))
		/* Использование функций json_string_*. */
	ellse if (json_is_integer(jdata))
		/* Использование функций json_integer_*. */
	ellse if (json_is_boolean(jdata))
		;
}
```

Целые числа и булевы значения

```
int main(int argc, char *argv[])
{
	json_t *jint, json_t *jbool;

	jint = json_integer(6);
	/* json_integer выделяет память, поэтому может завершиться ошибкой. */
	if (!jint)
		goto on_error;
	printf("Integer value: %d\n", json_integer_value(jint));
	json_decref(jint);

	jbool = json_boolean(1);
	/* json_boolean не выделяет память. */

	printf("Boolean value: %d\n", json_boolean_value(jbool));
	/* json_decref ничего не делает для булевых значений. */
	json_decref(jbool);
}
```

Строки

```
int main(int argc, char *argv[])
{
	char origin_str[] = "ABCDEFG";
	const char *new_str;
	json_t *jstr;

	jstr = json_string(origin_str);
	/* json_string копирует переданную строку. */
	if (!jstr)
		goto on_error;

	/* Исходную строку можно произвольно менять. Это не повлияет на jstr. */
	memset(origin_str, 0, sizeof(origin_str));

	new_str = json_string_value(jstr);
	/* json_string_value _не_ копирует строку. json_decref пока делать нельзя. */

	printf("String value: %s\n", new_str);

	/* Теперь можно. */
	json_decref(jstr);
}
```

Объекты

```
int main(int argc, char *argv[])
{
	json_t *jobj, *jchild;
	int ret;

	/* Создание пустого объекта. */
	jobj = json_object();
	if (!jobj)
		goto on_error;

	jchild = json_boolean(0);
	ret = json_object_set(jobj, "some-key", jchild);
	/* Добавление элемента в объект требует выделения памяти. */
	if (ret < 0)
		goto on_error;

	jchild = json_object_get(jobj, "some-key");
	json_object_del(jobj, "some-key");

	json_decref(jstr);
}
```

Замечания

- `json_object_get()` возвращает `NULL`, если указанный ключ отсутствует в объекте.
- `json_object_set()` заменяет существующий дочерний элемент с указанным ключом и освобождает его. Для сравнения - `avl_insert` возвращает -1 при добавлении двух жлементов с одинаковыми ключами.
- `json_object_del()` возвращает -1, если указанный ключ отсутствует в объекте.
- `json_decref` обычно (см. далее более точное правило) освобождает объект со всеми дочерними элементами. В отличие от `struct avl_tree` перебор дочерних желементов не требуется.

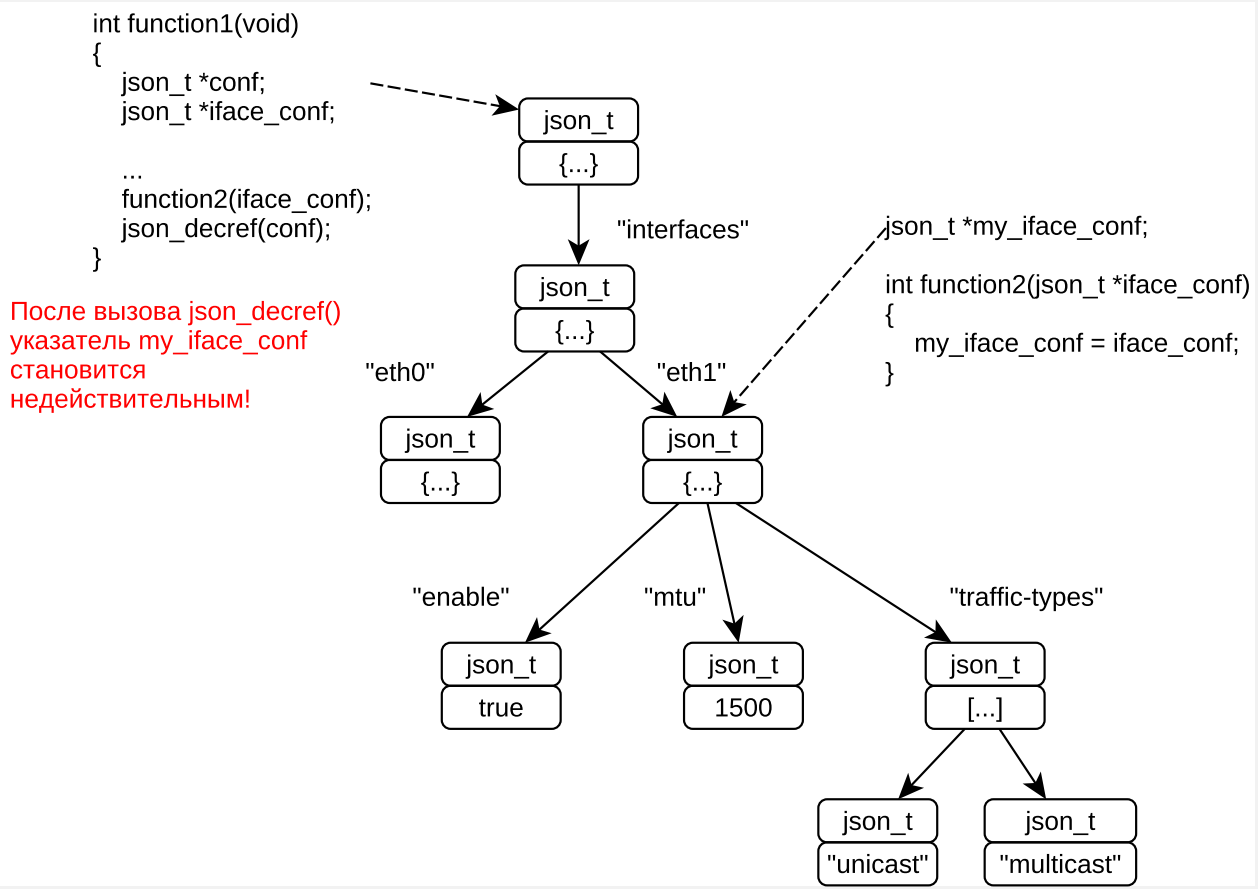
Массивы

```
int main(int argc, char *argv[])
{
	json_t *jarr, *jchild;
	int ret;

	/* Создание пустого массива. */
	jarr = json_array();
	if (!jarr)
		goto on_error;

	jchild = json_boolean(0);
	ret = json_array_append(jarr, jchild);
	/* Добавление элемента в массив требует выделения памяти. */
	if (ret < 0)
		goto on_error;

	/* Извлечение элемента с индексом 0. */
	jchild = json_array_get(jarr, 0);
	/* Удаление элемента с индексом 0. */
	json_array_remove(jarr, 0);
	json_decref(jarr);
}
```

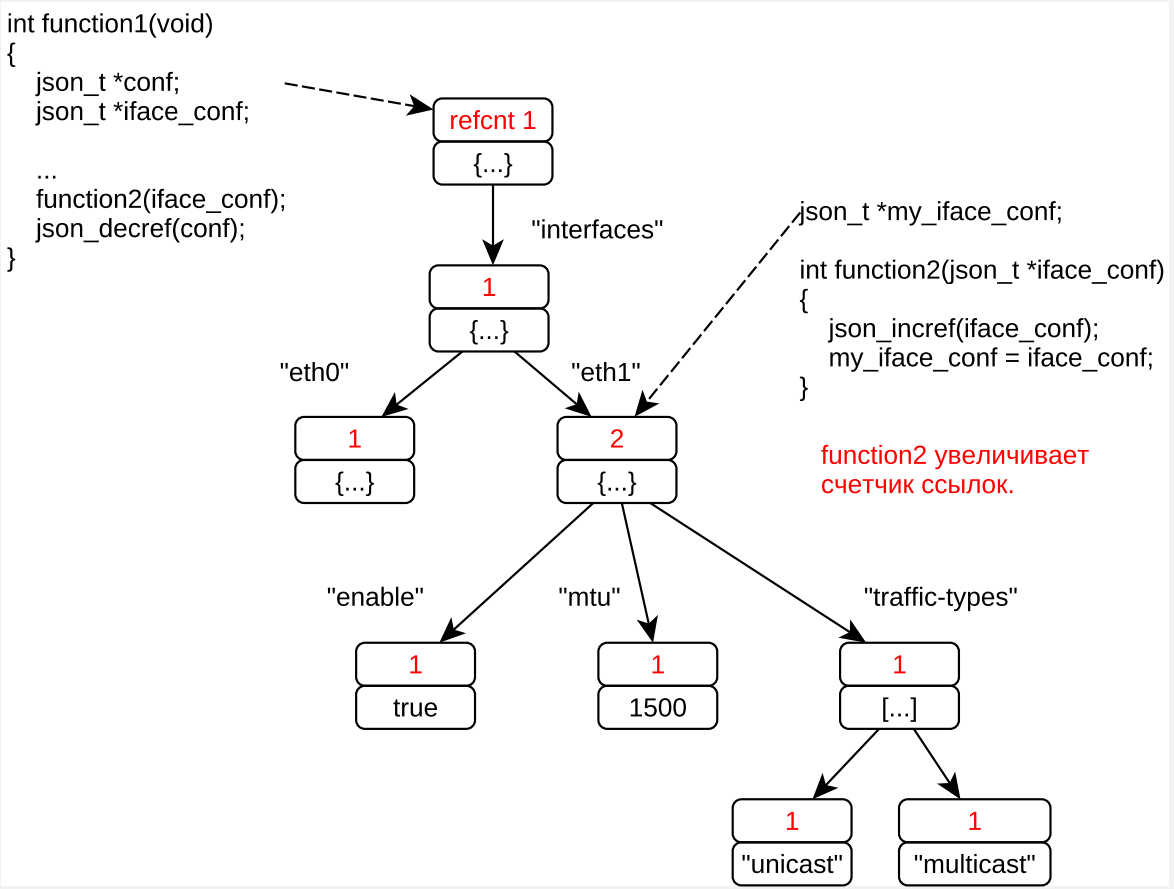
Перебор всех элементов

```
int main(int argc, char *argv[])
{
	json_t *jobj, *jarr, *jchild;
	const char *key;
	int index;

	json_object_foreach(jobj, key, jchild) {
		printf("Object child: key %s, type %d\n", key, json_typef(jchild));

		/* Удалять jchild нельзя!. */
	}

	json_array_foreach(jarr, index, jchild) {
		printf("Array child: index %d, type %d\n", index, json_typef(jchild));

		/* Удалять jchild нельзя!. */
	}
}
```

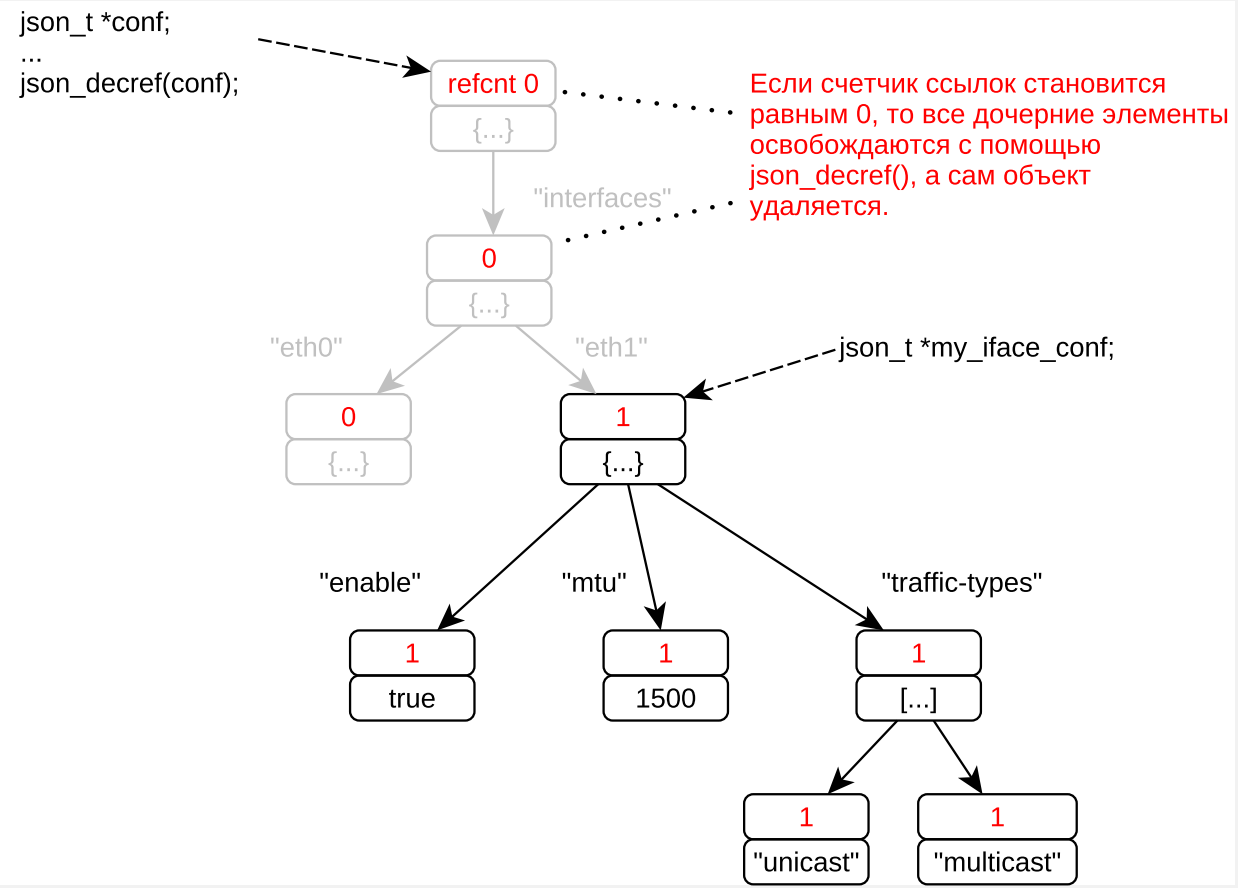
Совместное использование документа



Использование счетчика ссылок



Освобождение части документа



Правила использования счетчика ссылок

- В каждом `json_t` есть счетчик ссылок `refcount`.
- При создании элемента его счетчику присваивается значение 1.
- Вызов `json_decref()` уменьшает счетчик на 1.
- Когда значение счетчика становится равным 0, элемент удаляется.
- Когда значение счетчика объекта или массива становится равным 0, производится уменьшение на 1 счетчиков всех дочерних элементов.
- Явное инкрементирование счетчика производится функцией `json_incref()`.
- Некоторые функции jansson инкрементируют счетчики обрабатываемых элементов.

Добавление без инкрементирования refcount

```
int main(int argc, char *argv[])
{
	json_t *jobj, *jchild;
	int ret;

	jobj = json_object();

	jchild = json_integer(10);

	/* Функция не инкрементирует refcount. jchild после вызова принадлежит jobj. */
	json_object_set_new(jobj, "some-key", jchild);

	/* Удаление jobj вместе с jchild, так как refcount jchild равен 1. */
	json_decref(jobj);

	/* Так делать нельзя! Повторное освобождение приведет к разрушению памяти. */
	json_decref(jchild);
}
```

Добавление с инкрементированием refcount

```
int main(int argc, char *argv[])
{
        json_t *jobj, *jchild;
        int ret;

        jobj = json_object();

        jchild = json_integer(10);

        /* Функция инкрементирует refcount. */
        json_object_set(jobj, "some-key1", jchild);

        /* Можно добавить еще раз с другим ключом. */
        json_object_set(jobj, "some-key2", jchild);

        json_decref(jobj);

        /* У нас по-прежнему осталась наша ссылка.
         * Если не вызвать json_decref, произойдет утечка памяти. */
        json_decref(jchild);
}
```

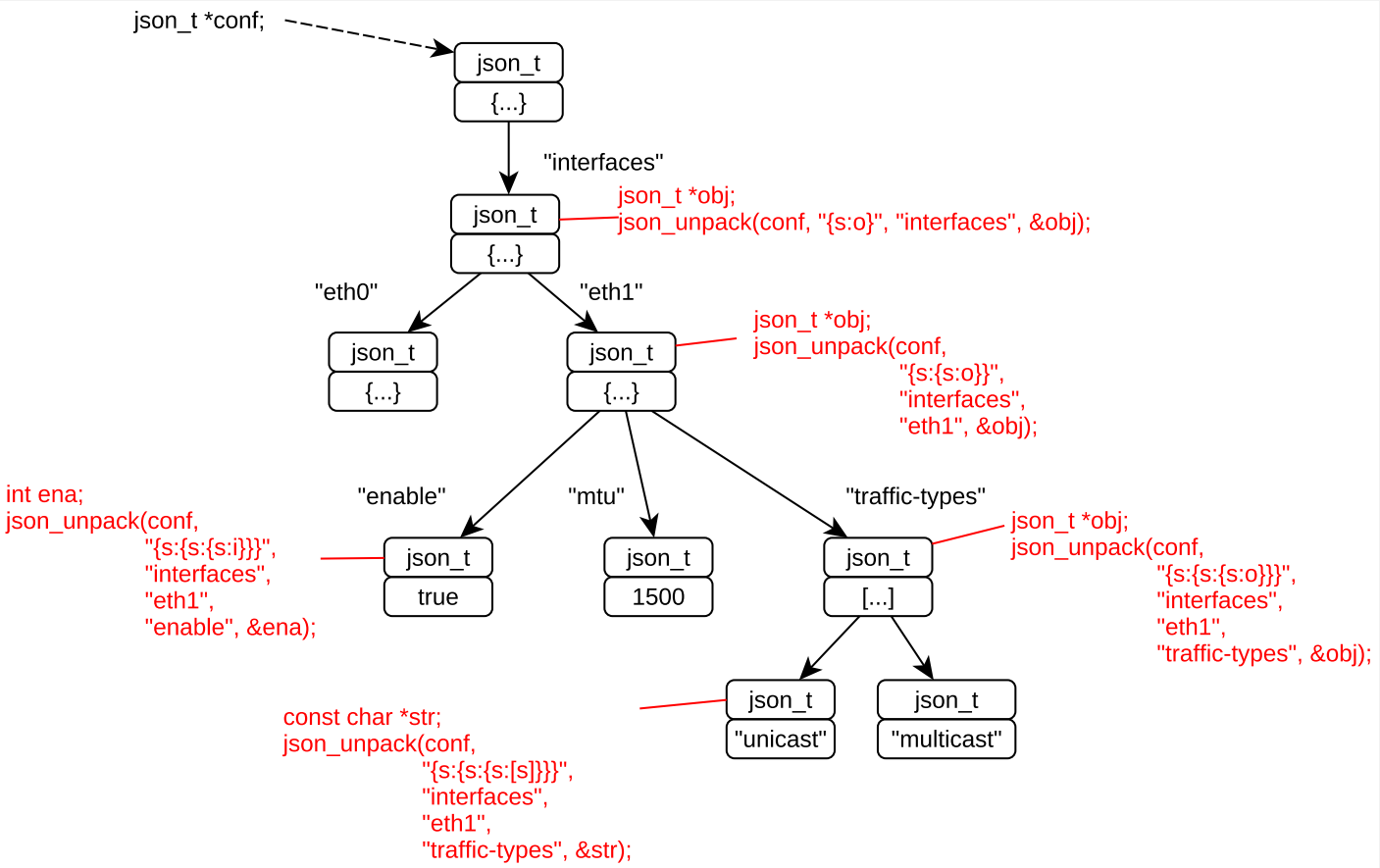
Что такое „ссылка”?

- Ссылка существует лишь неявно. Нет специальной структуры, представляющей ссылку в программе.
- Ссылка - обязательство вызвать `json_decref()` после завершения использования некоторого дерева `json_t`.
- Функция *заимствует* (borrow) ссылку, если она увеличивает `refcount` на 1 и гарантирует вызов `json_decref()` в будущем (возможно, после завершения функции).
- Функция *ворует* (steal) ссылку, если она гарантирует вызов `json_decref()` в будущем, но не увеличивает `refcount`.
- Нежелательно создавать функции, которые в зависимости от условий либо воруют, либо заимствуют ссылку.

Пример

```
int main(int argc, char *argv[])
{
	json_t *jobj, *jchild;
	int ret;

	jobj = json_object();

	jchild = json_integer(10);

	/* Функция ворует ссылку. */
	ret = json_object_set_new(jobj, "some-key", jchild);
	if (ret < 0) {
		/* Мы не смогли добавить jchild в объект. Надо ли его освобождать? Нет.
*/
		/* json_decref(jchild); */
		goto on_error;
	}

	json_decref(jobj);
}
```

Извлечение вложенных элементов



Использование json_unpack_ex()

```
{
	int enable;
	const char *address = NULL; /* Обязательно инициализировать! */

	/* '?' означает, что при отсутствии ключа переданный указатель не меняется. */
	ret = json_unpack_ex(jconf, &jerr, 0,
	                     "{s:{s:b s?s}}",
	                     "interfaces",
	                     "enable", &enable,
	                     "address", &address);
	if (ret < 0) {
		fprintf(stderr, "Invalid interface config: %s\n", jerr.text);
		return -1;
	}
	/* Что делать, если ключ "address" не был найден? */
	if (!address)
		address = "192.168.0.1";
	else
		/* address указывает на строку внутри json_t.
		 * Ее нельзя менять или освобождать. */
}
```

Генерация дерева json_t

```
{
        int enable = 1;
        const char *address = "192.168.0.2";
        json_t *jchild1, *jchild2;

        jchild1 = json_integer(3);
        jchild2 = json_integer(4);

        /* 'o' - ссылка воруется,
         * 'O' - ссылка заимствуется. */
        jconf = json_pack("{s:{s:b s:s s:o s:O}}",
                        "interfaces",
                        "enable", enable, /* Теперь значение, а не ссылка. */
                        "address", address,
                        "key1", jchild1,
                        "key2", jchild2);
        if (!jconf) {
                json_decref(jchild2);
                return -1;
        }
}
```

Документ XML

```
<network>
	<interface>
		<name>eth0</name>
		<enable>true</enable>
		<address>192.168.0.5</address>
		<prefix-length>28</prefix-length>
		<traffic-type>multicast</traffic-type>
	</interface>
	<interface>
		<name>eth1</name>
		<enable>false</enable>
		<address>192.168.1.10</address>
		<prefix-length>22</prefix-length>
		<traffic-type>unicast</traffic-type>
		<traffic-type>multicast</traffic-type>
		<mtu>1400</mtu>
	</interface>
</network>
```