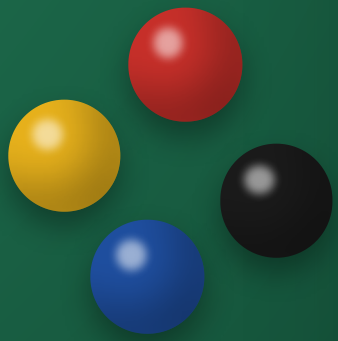
ТЕХНИЧЕСКОЕ ОПИСАНИЕ

Архитектура игры «Бильярд»

Как Flash-игра для ВКонтакте работает на сервере и транслируется в браузер игрока потоком H.264 и Opus по WebSocket



Проект fengine · ноябрь 2026

player (C) · manager (Go) · nginx · клиент HTML5 · сервер игры (Perl, Go)

Содержание

1	Общая картина	задача, компоненты, путь игрока
2	Маршрутизация и запуск экземпляров	nginx, manager, подпись VK, узлы
3	player изнутри	NPAPI-хост, потоки, правило X11
4	Видео, звук и ввод	H.264, Opus, мышь и клавиатура
5	Протокол WebSocket	двоичные кадры и JSON-события
6	Интеграция с ВКонтакте	VK API, LocalConnection, платежи, ссылки
7	Полный экран и буфер обмена	монитор Flash, скроллбар, копирование
8	Изоляция и надёжность	Xvfb, HOME, shm, сеть Flash
9	Сервер игры	Perl сегодня, Go на замену
10	Ключевые параметры	сводная таблица

Коротко. Игра — это SWF-файл, рассчитанный на браузерный Flash Player, которого в современных браузерах нет. Поэтому Flash исполняется на сервере: на каждого игрока запускается свой процесс `player` с настоящим `libflashplayer.so` в виртуальном X-сервере. Картинка кодируется в H.264, звук — в Opus, и всё уходит в браузер по WebSocket. Мышь и клавиатура идут обратно и превращаются в события X11 для Flash. Всё, что игра хочет от ВКонтакте, выполняет браузер игрока через VK Bridge.

1 Общая картина

Система состоит из пяти частей: страницы-клиента в iframe ВКонтакте, nginx, менеджера экземпляров, процессов `player` и сервера самой игры.

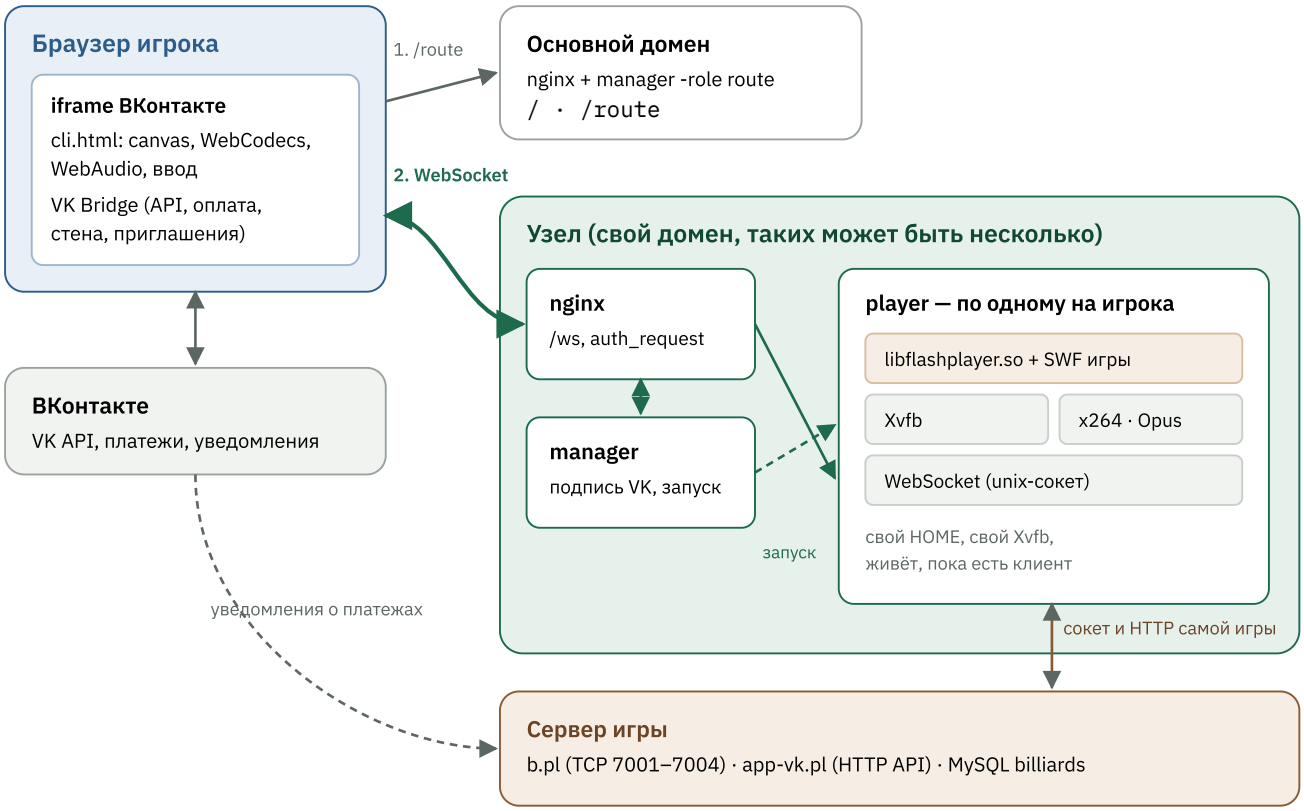


Рис. 1. Компоненты и связи. Сплошные стрелки — запросы при входе игрока, пунктир — фоновые.

Путь игрока от клика до картинки

1. ВКонтакте открывает страницу клиента в iframe и передаёт в адресе параметры запуска: `viewer_id`, `auth_key` или подписанные `vk_*` для мини-приложения.
2. Клиент спрашивает основной домен `GET /route`. Тот проверяет подпись и называет узел: `{"ws":"wss://<узел>/ws"}`. Если серверов один, клиент сразу идёт на свой `/ws`.
3. nginx узла через `auth_request` спрашивает менеджер. Менеджер снова проверяет подпись, при необходимости запускает `player` и возвращает uid в заголовке `X-Player-Uid`.
4. nginx проксирует WebSocket в unix-сокет игрока `/run/player/<uid>.sock`. uid берётся только из ответа менеджера, никогда из запроса.
5. `player` отдаёт ключевой кадр H.264, затем изменения экрана и звук. Клиент шлёт обратно мышь, клавиатуру и ответы VK Bridge.

Состав репозитория

player

`C · h264/player.c`

NPAPI-хост для Flash 32. Грузит SWF, рисует в pixmap Xvfb, кодирует H.264 и Opus, принимает ввод. Корневой `player.c` — вариант с MJPEG, общие части правятся в обоих.

Заглушки звука

`C`

`alsa_shim.c` — виртуальная звуковая карта, отдающая PCM в кодер. `pulse_shim.c` — «PulseAudio недоступен», без неё Flash падает при выходе.

manager

`Go`

Проверка подписи VK, запуск экземпляров, маршрутизация между узлами. Состояния не хранит: экземпляр жив, если его сокет принимает соединения.

nginx

`nginx/*.conf`

TLS, страница клиента, `auth_request` к менеджеру, прокси WebSocket в unix-сокеты. Наружу открыт только он.

Клиент

`HTML · cli.html`

Одна страница: декодер H.264 (WebCodecs или tinyh264 в WASM), Opus, ввод, VK Bridge, полный экран, буфер обмена.

Сервер игры

`Perl · pl/`

`b.pl` — игровой сервер на сокетах, `app-vk.pl` — HTTP API, `hub.pl` — залы, MySQL `billiards`.

Новый сервер

`Go · server/`

Замена `pl/` для HTML5-клиента: серверная физика, правила 8-ки, залы, та же MySQL.

Исходники AS3

`as/`

Ранняя версия игры 2011 года. В сборке не участвуют, но служат эталоном физики и правил и картой обфусцированного SWF.

2 Маршрутизация и запуск экземпляров

Менеджер — небольшой Go-сервис без базы данных. Он отвечает на два вопроса: «кто этот игрок» и «где его экземпляр».

Проверка подписи ВКонтакте

Тип запуска	Подпись	uid
iframe-приложение	`auth_key = md5(api_id _ viewer_id _ secret)`	`viewer_id`
мини-приложение	`sign` = base64url(HMAC-SHA256(secret, отсортированные `vk_*`)), `vk_ts` не старше 24 ч	`vk_user_id`

Игра написана для iframe-приложения. Если её запустили как мини-приложение, менеджер дописывает недостающие `api_url`, `viewer_id`, `auth_key` и другие параметры тем же секретом — их проверяет сервер игры. Секрет хранится только в переменной окружения `VK_APP_SECRET`.

Запуск экземпляра

Если сокет игрока не отвечает, менеджер запускает:

```
player -u /run/player/<uid>.sock -P profiles/<uid> -i <idle> -f <query> <swf>
```

- `-P` — постоянный профиль игрока с блокировкой: настройки и SharedObjects Flash переживают перезапуск.
- `-f` — параметры VK передаются как `flashvars` без перекодирования: от порядка и экранирования зависят подписи. URL SWF при этом постоянный, и кеш SWF общий для всех.
- `-i` — экземпляр сам выходит, если клиентов нет дольше заданного времени.
- Экземпляры запускаются в своей сессии (`setsid`) и переживают перезапуск менеджера.

Несколько серверов

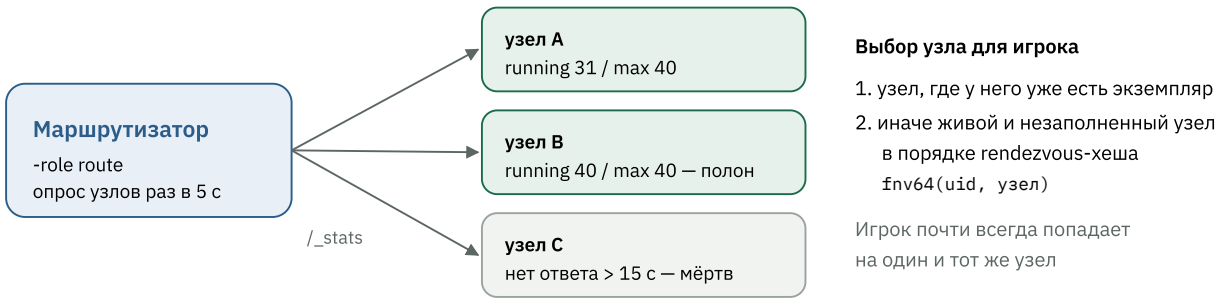


Рис. 2. Маршрутизатор основного домена и узлы. Общего состояния между узлами нет.

Основной домен — адрес приложения в настройках ВКонтакте. Он отдаёт страницу и выбирает узел, а WebSocket клиент открывает прямо к узлу: правило одного источника на WebSocket не распространяется, а VK Bridge остаётся на странице основного домена. Узел принимает подключения только со страницы основного домена (`-origins`) и сам ещё раз проверяет подпись. После трёх неудачных подключений подряд клиент запрашивает маршрут заново.

3 player изнутри

`player` играет роль браузера для Flash: реализует NPAPI, загружает файлы, отдаёт DOM-объекты, принимает отрисовку. Flash работает в windowless-режиме и рисует в pixmap виртуального X-сервера.

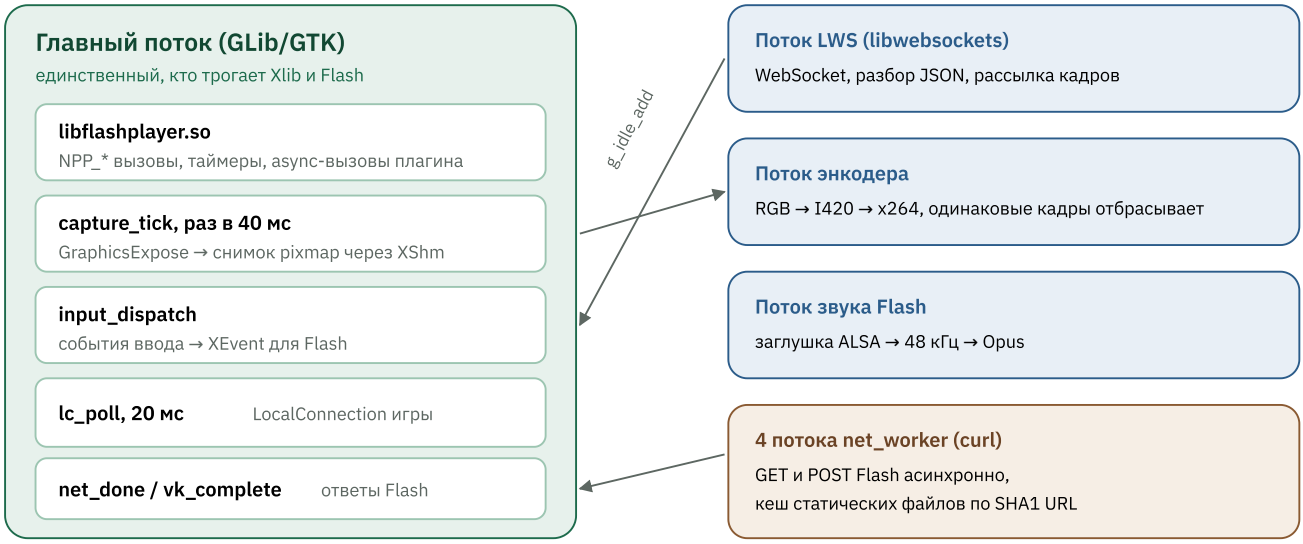


Рис. 3. Потоки процесса player. Xlib не инициализирован для многопоточности, поэтому всё, что касается X11 и Flash, идёт через главный поток.

Правило X11

Поток WebSocket никогда не обращается к X-серверу и к плагину. Каждое событие ввода он кладёт в очередь главного потока через `g_idle_add`, поэтому порядок событий сохраняется. Ответы сети и VK API тоже доставляются Flash только из главного потока: `deliver_stream` + `NPP_URLNotify`.

Сеть Flash

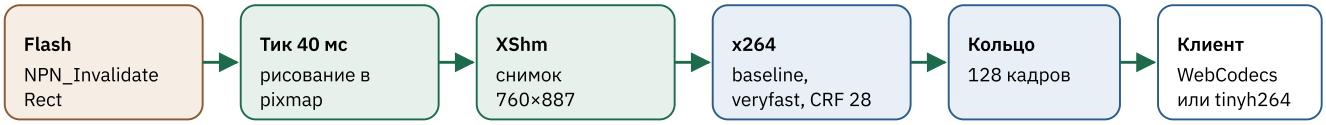
Раньше curl выполнялся в главном потоке, и на время запроса к API игры (0,5–1,5 с) Flash не рисовал и не получал ввод. Теперь запросы идут через очередь `net_queue` и четыре рабочих потока, у каждого своё соединение с сохранением TLS-сессий. Синхронно остаётся только загрузка самого SWF при старте.

Особенности NPAPI, которые легко сломать

- `libflashplayer.so` — отладочная сборка. На необработанное исключение ActionScript она открывает модальный диалог и зависает, поэтому в `~/mm.cfg` экземпляра пишется `SuppressDebuggerExceptionDialogs=1`.
- Scriptable-объект плагина отпускается только после `NPP_Destroy`, как в Firefox.
- `NPN_GetValue` пишет ровно размер переменной: `NPBool` — 1 байт.

4 Видео, звук и ввод

Видеоконвейер



Нет клиентов — тик не рисует и не кодирует, изменения копятся до подключения.

- **Рисование только в тике.** SWF идёт с частотой 40 кадров/с, а в браузер уходит 25. Растеризовать по каждому `InvalidateRect` значило бы тратить CPU на кадры, которых никто не увидит. Частоту самого SWF `player` при загрузке снижает до 25, правя заголовок файла: CPU стола без движения — 46% → 24% ядра.
- **Кадр только при изменении.** Энкодер обязан отдавать кадр сразу (`maximum_delayed_frames == 0`), иначе последнее изменение — закрытое окно — зависало бы до следующего, и клики казались бы недошедшими.
- **Ключевые кадры.** Новый или отставший больше чем на 25 кадров клиент ждёт ключевой кадр, который запрашивается у энкодера сразу. SPS/PPS повторяются перед каждым ключевым кадром.
- **Сжатие.** Пресет `veryfast`, CRF 28, потолок 5000 кбит/с: на записях игры поток на 43% меньше прежнего `superfast`/CRF 23. Профиль только baseline — его понимает программный декодер клиента.

Звук

До загрузки плагина `player` подгружает две заглушки с SONAME `libasound.so.2` и `libpulse.so.0`, и Flash находит их вместо системных библиотек. Заглушка ALSA ведёт темп по монотонным часам и отдаёт PCM прямо в кодер. Flash принимает только частоту 44100 Гц, поэтому до 48 кГц для Opus передискретизирует сам `player`. Клиент держит задержку воспроизведения около 80 мс и пересинхронизируется, если отстал больше чем на 300 мс.

Ввод

- **Мышь.** Flash получает движение не чаще тика и только последнее: клиент шлёт 60+ движений в секунду, а игра на каждое пересчитывает линию прицела. Главный поток при прицеливании: 57% → 33–40%. Перед нажатием отложенное движение отдаётся сразу, так что координаты кликов не меняются.
- **Клавиатура.** `KeyboardEvent.key` → keysym → keycode. Кириллицы в раскладке Xvfb нет, поэтому символ временно назначается на свободный keycode. Shift для печатных символов определяет сам символ, а не Shift игрока.
- **Вставка.** Ctrl+V набирает текст по символу. Сервер пропускает только печатные символы, без `<` и `>` (чат игры передаёт HTML), до 300 символов и не чаще раза в секунду.
- **Залипание.** При подключении новой вкладки и уходе старой все нажатые кнопки и клавиши отпускаются, иначе Flash считал бы их зажатыми.

5 Протокол WebSocket

Одно соединение несёт всё: двоичные кадры от сервера, JSON-сообщения в обе стороны.

Двоичные сообщения сервера

Первый байт	Содержимое
`0x02`	H.264: флаги (бит 0 — ключевой), ширина, высота, метка времени в мс, затем Annex B
`0x01`	Opus: каналы, число сэмплов, метка времени в сэмплах 48 кГц, пакет
`0xFF`	JPEG целиком (только вариант MJPEG)

JSON от сервера, поле `type`

type	Назначение
`vkapi`	Вызов метода VK API от имени игры; клиент отвечает событием `vkapi` с тем же `id`
`vkbridge`	Событие VK Bridge как есть: оплата, стена, приглашение, права
`open`	Открыть ссылку игры (профиль, группа) в новой вкладке
`copy`	Текст, скопированный в игре, — записать в буфер обмена браузера
`fullscreen`	Flash вышел из полного экрана — выйти и браузеру

События клиента, поле `event`

event	Назначение
`move`, `down`, `up`	Мышь; `button` 0/1/2
`scroll`	Колесо: `dir` или `deltaY`
`keydown`, `keyup`	Клавиатура по `KeyboardEvent.key`
`text`	Вставка из буфера обмена
`copy`	Ctrl+C в игре
`audio`	Включить или выключить звук
`keyframe`	Запросить ключевой кадр
`fullscreen`	Вход в полный экран с размером экрана или выход
`vkapi`, `vkbridge`	Ответы на запросы сервера: `id`, `result`
`log`	Отчёт клиента о показанных кадрах для лога сервера

Одна вкладка на игрока. Новое подключение вытесняет прежние: они закрываются кодом 4000 `replaced`, их ввод игнорируется. Клиент на этот код сам не переподключается, иначе две вкладки выбивали бы друг друга по кругу; игра возвращается во вкладку по клику. После пяти неудачных подключений подряд клиент просит обновить страницу: свежие параметры VK даёт только перезагрузка всей страницы.

6 Интеграция с ВКонтакте

Сессия VK выдана на IP браузера игрока, поэтому с сервера запросы к VK не работают. Всё, что игра хочет от ВКонтакте, `player` перехватывает и пересылает браузеру.



Рис. 4. Два пути обращений игры к ВКонтакте и их выполнение в браузере игрока.

VK API

POST игры на хосты VK API разбирается в метод и параметры и уходит клиенту как `{"type":"vkapi","id","method","params"}`. Клиент получает ключ игрока (`VKWebAppGetAuthToken`, scope `friends`) и вызывает метод через VK Bridge. Ответ возвращается Flash асинхронно; если его нет 60 секунд, Flash получает ошибку в формате VK и игра работает без данных VK. Игра обращается к VK редко: свой профиль и друзья при запуске, профили по 20 штук для «ТОП».

LocalConnection — контейнер VK

Старый Flash API ВКонтакте общается с игрой через LocalConnection: команды контейнеру уходят в `_in_<lc_name>`, ответы ожидаются на `_out_<lc_name>`. `player` сам регистрируется слушателем в общем сегменте памяти Flash, разбирает сообщения AMF0/AMF3 и отвечает в формате AS3.

Команда игры	Что делает player
`initConnection`	Отвечает тем же `initConnection` — рукопожатие
`showOrderBox`	`VKWebAppShowOrderBox` → `onOrderSuccess`, `onOrderCancel` или `onOrderFail`
`api wall.post`	`VKWebAppShowWallPostBox` → `apiCallback`
`showInviteBox`	`VKWebAppShowInviteBox`
`showSettingsBox`	Биты маски → scope → `VKWebAppGetAuthToken` → `onSettingsChanged`
`showInstallBox`	Сразу `onApplicationAdded`: запущено из VK — значит, установлено
`navigateToURL`	Ссылка уходит клиенту, тот открывает её в новой вкладке

Отладочный вывод игры (клиент Arthropod) тоже идёт через LocalConnection; `player` слушает и это имя и пишет строки `[GAME] debug` в лог.

Платежи. `player` только сообщает игре исход окна оплаты. Золото зачисляет сервер игры по уведомлению ВКонтакте о платеже, а новый баланс приходит игре сообщением `updateGold` по её собственному сокету.

7 Полный экран и буфер обмена

Полный экран (вариант H.264)

Кнопка клиента переводит браузер в полный экран и включает полноэкранный режим самой игры: стол во всю ширину, свой скроллбар справа. Flash при этом рисует не в pixmap, а в собственное окно на экране Xvfb.

1. Клиент ждёт, пока размер окна не меняется 6 кадров подряд, и шлёт `{"event":"fullscreen","on":true,"w","h"}`.
2. `player` считает «монитор»: ширина всегда 1280 (флаг `-F`), высота — по пропорциям экрана, плюс скрытая полоса около 3,5% справа, куда уходит скроллбар игры.
3. Flash узнаёт размер монитора у GDK — эти функции перехвачены. Затем `player` нажимает значок игры «Во весь экран»: для Flash это действие пользователя.
4. Окно Flash снимается с корневого окна Xvfb, энкодер пересоздаётся под новый размер, ввод идёт через XTest.

> **Почему ширина постоянная.** Игра запоминает размер монитора первого входа и держит по нему скроллбар до перезагрузки. Клиент в iframe VK иногда присылал размер iframe (770×930) или окна на полпути к полному экрану, и скроллбар оставался посередине стола.

Копирование из игры

Буфер обмена Flash — это CLIPBOARD X-сервера `player`, браузер его не видит. Игрок выделяет текст мышью, например `uid = …` в профиле, и нажимает Ctrl+C.

1. Клиент узнаёт сочетание по `e.code` в любой раскладке и шлёт `{"event":"copy"}`.
2. `player` нажимает Ctrl+C во Flash; Flash кладёт текст в буфер через GTK.
3. Сигнал GTK `owner-change` забирает текст (до 2000 символов) и отправляет клиенту `{"type":"copy","text"}`.
4. Клиент пишет его в буфер браузера: `navigator.clipboard`, иначе `execCommand("copy")`, иначе `VKWebAppCopyText`.

В полном экране копирование не работает: с клавиатуры Flash там принимает только Esc.

8 Изоляция и надёжность

На одном сервере работают десятки экземпляров Flash. Каждый должен видеть только своё: экран, файлы, общую память, сеть.

Свой Xvfb

Номер дисплея через `-displayfd`, свой MIT-cookie, `PR_SET_PDEATHSIG` и отдельная группа процессов: X-сервер умирает вместе с `player`.

Свой HOME

Flash хранит SharedObjects и настройки в `~/.macromedia`. Каталог экземпляра `/tmp/player-<pid>-*`, каталоги мёртвых PID удаляет следующий запуск.

Своя LocalConnection

Flash использует фиксированный ключ shm `0xcbc384f8`. `shmget` / `semget` перехвачены, и ключ превращается в ключ экземпляра; объекты мёртвых PID удаляются.

Свой сокет

Unix-сокет с правами 0660 для группы nginx. Если сокет уже принимает соединения, второй экземпляр не стартует.

Кеш файлов

В общий кеш попадают только статические файлы, ключ — SHA1 полного URL. SWF игры кешируется один раз для всех.

Самозавершение

Без клиентов дольше `-i` секунд экземпляр выходит сам. Менеджеру не нужно ничего отслеживать.

Снаружи

- Все процессы слушают только `127.0.0.1` или unix-сокеты; наружу открыт nginx.
- Ошибки менеджера, кроме 401 и 403, nginx превращает в 500 — клиент не видит внутренних подробностей.
- Токен статистики узлов — в `MANAGER_STATS_TOKEN`, секрет VK — в `VK_APP_SECRET`, оба только в окружении.

Диагностика

Лог экземпляра показывает, что ушло клиенту и с какой задержкой:

- `[ws] 5s: кадров N (ключевых K), M КБ, отставаний L`
- `[input] down x,y: в очереди N мс` — задержка нажатия до Flash
- `[client] …` — отчёт браузера: показано кадров, ждут в декодере, сбросы
- `[net]`, `[LC]`, `[GAME] debug`, `[open]`, `[copy]`, `[paste]`

9 Сервер игры

Flash-клиент сам держит TCP-соединение с сервером игры. `player` в этот обмен не вмешивается: для сервера это обычный игрок.

Текущий сервер на Perl (`pl/`)

- **`b.pl`** — игровой сервер на сокетах, порты 7001–7004. Протокол: 2 байта длины + JSON, XOR по кругу с ключом; после `connect` ключ — `sig` из ответа сервера.
- **`app-vk.pl`** — HTTP API (FastCGI), в том числе уведомления ВКонтакте о платежах.
- **`hub.pl`** — залы и подбор соперника; `chat-vk.pl`, `tour8.pl`, `top.pl` — чат, турниры, рейтинг.
- База — MySQL `billiards`. Сервер не пускает один аккаунт дважды: «Приложение уже запущено на данном компьютере».

Новый сервер на Go (`server/`)

Пишется на замену `pl/` для будущего HTML5-клиента, которому Flash не нужен. Главное отличие — исход удара решает сервер.

physics

Построчный перенос физики из `as/`: событийная аналитическая модель, удар считается до остановки сразу, с сохранением округлений AS3.

table · rules

Стол 8-ки и правила с фолами, заказом лузы, установкой битка и проверкой пределов удара.

game

Залы и столы из `hub.pl`, ставки, время хода 60 с, начисление опыта и золота. Стол — горутина с очередью команд.

srv · store

WebSocket-сервер с проверкой подписи VK, протокол в `PROTOCOL.md`. Хранилище — MySQL или SQLite.

Эталон — `as/`, а не текущий SWF. Текущий модуль `vk8sfx5` во многом тот же код, но обфусцирован и местами переделан, поэтому выводы из исходников 2011 года сверяются с самим SWF.

10 Ключевые параметры

760×887
размер сцены игры

25 к/с
частота кадров в браузер

CRF 28
x264 veryfast, baseline

48 кГц
Opus, задержка ~80 мс

Параметр	Значение	Где задаётся
Тик захвата	40 мс	`capture_tick`
Частота SWF	40 → 25 к/с при загрузке	`SWF_MAX_FPS`, флаг `-R`
x264	veryfast, CRF 28, VBV 5000 кбит/с, буфер 10000	`VIDEO_*`
Ключевой кадр	не реже раза в 4 с при анимации	`VIDEO_KEYINT`
Кольцо кадров	128, отставание > 25 — перескок	`VIDEO_RING`, `VIDEO_MAX_LAG`
Уровень H.264	3.1 до 1280×720, иначе 4.0	`x264_open`
Полный экран	ширина 1280 (до 1920), высота 300–1080	флаг `-F`
Ответ VK API	60 с; окно оплаты — 10 мин	`VK_API_TIMEOUT`, `VK_ORDER_TIMEOUT`
Вставка	до 300 символов, раз в секунду	`PASTE_MAX`
Копирование	до 2000 символов	`COPY_MAX`
Подпись мини-приложения	не старше 24 ч	`-sign-max-age`
Опрос узлов	раз в 5 с, мёртв через 15 с	`manager/route.go`
Сетевые потоки Flash	4	`net_worker`

Сборка и запуск

`h264/build.sh` собирает заглушки звука и `player`; зависимости — GTK2, Xlib с расширениями, libwebsockets, json-c, libcurl, libopus, libx264, QuickJS, Xvfb. Менеджер: `cd manager && go build`. `player` запускается из каталога, где лежат `libflashplayer.so` и заглушки. Автотесты есть у менеджера (подпись VK) и у нового сервера; `player` проверяется запуском под Xvfb с WebSocket-клиентом и разбором кадров через ffmpeg.