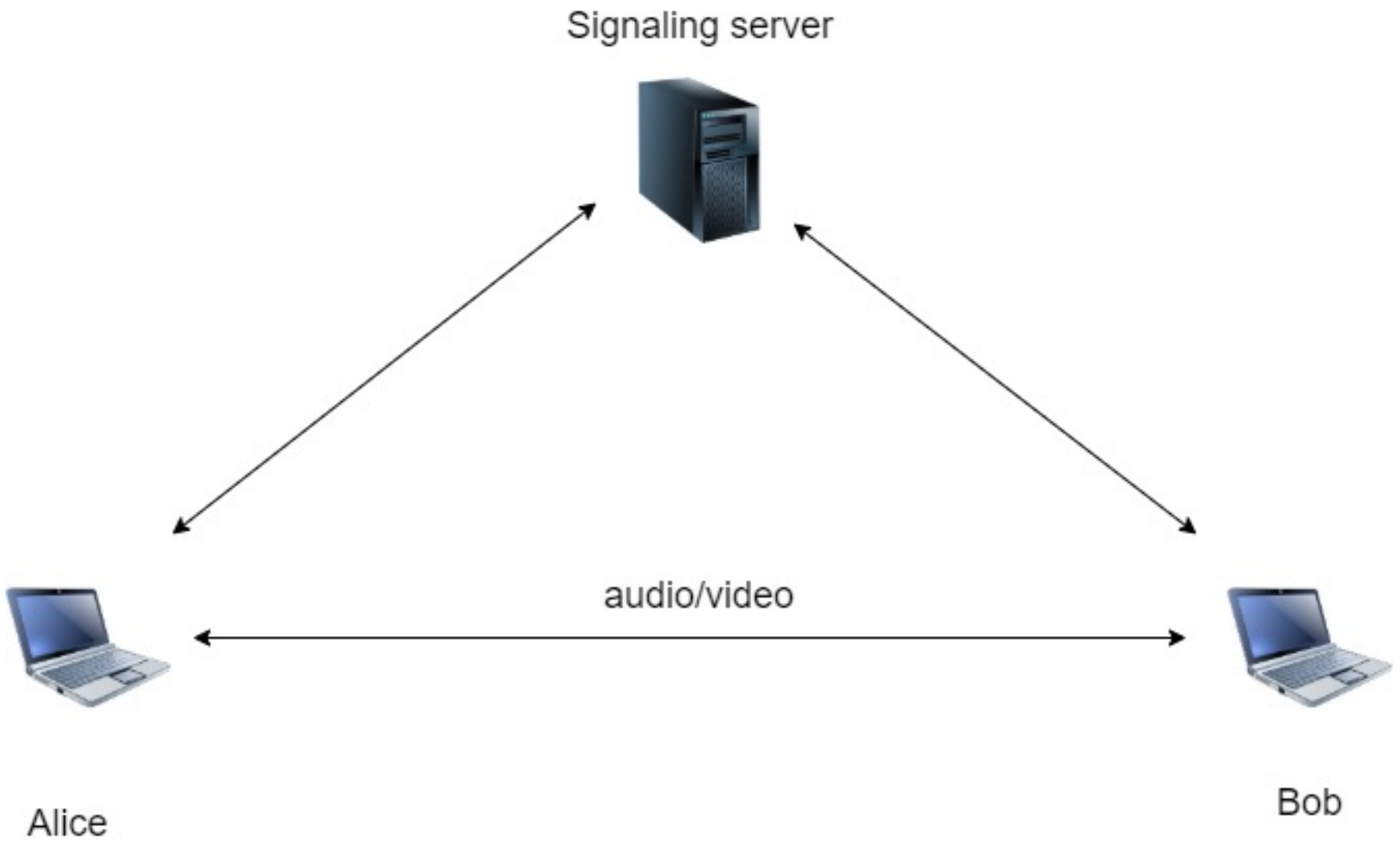
Делаем SFU сервер из WebRTC своими руками

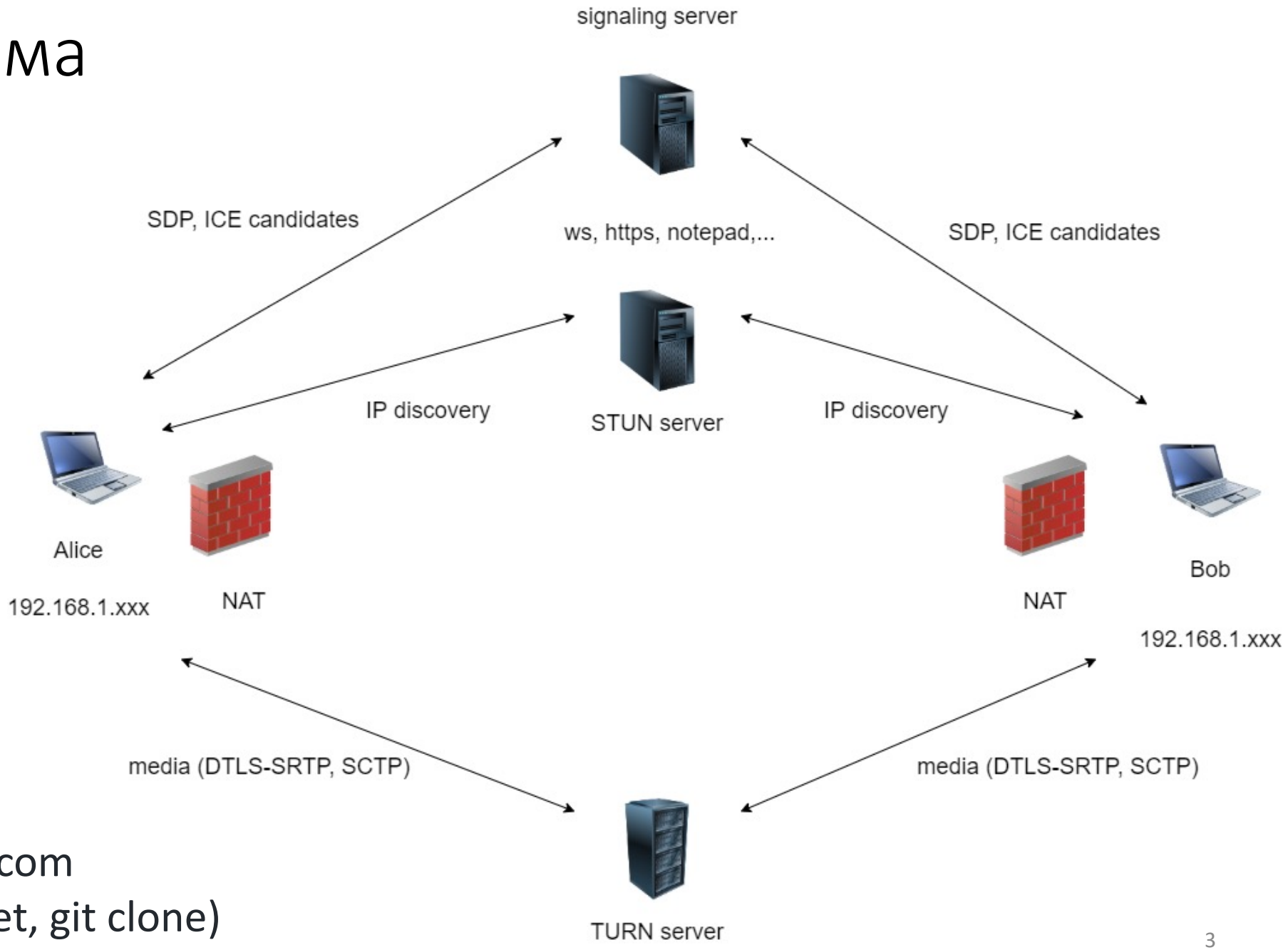
Дмитрий Байдаев
(IVA Technologies)

VideoTech 2021

WebRTC – «это чтобы звонить»



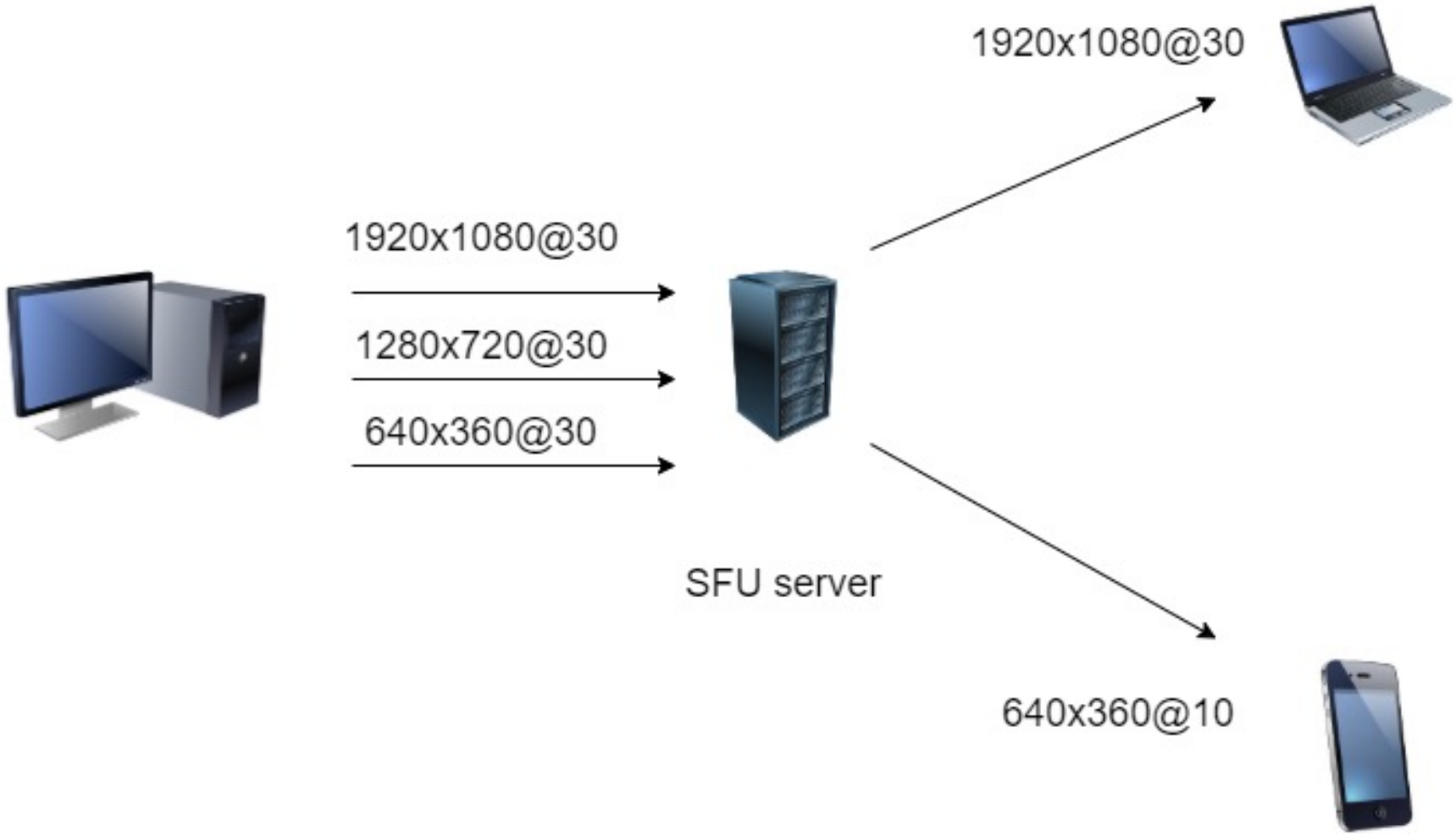
Базовая схема



STUN -> stun.l.google.com

TURN -> coturn (apt-get, git clone)

Simulcast SFU



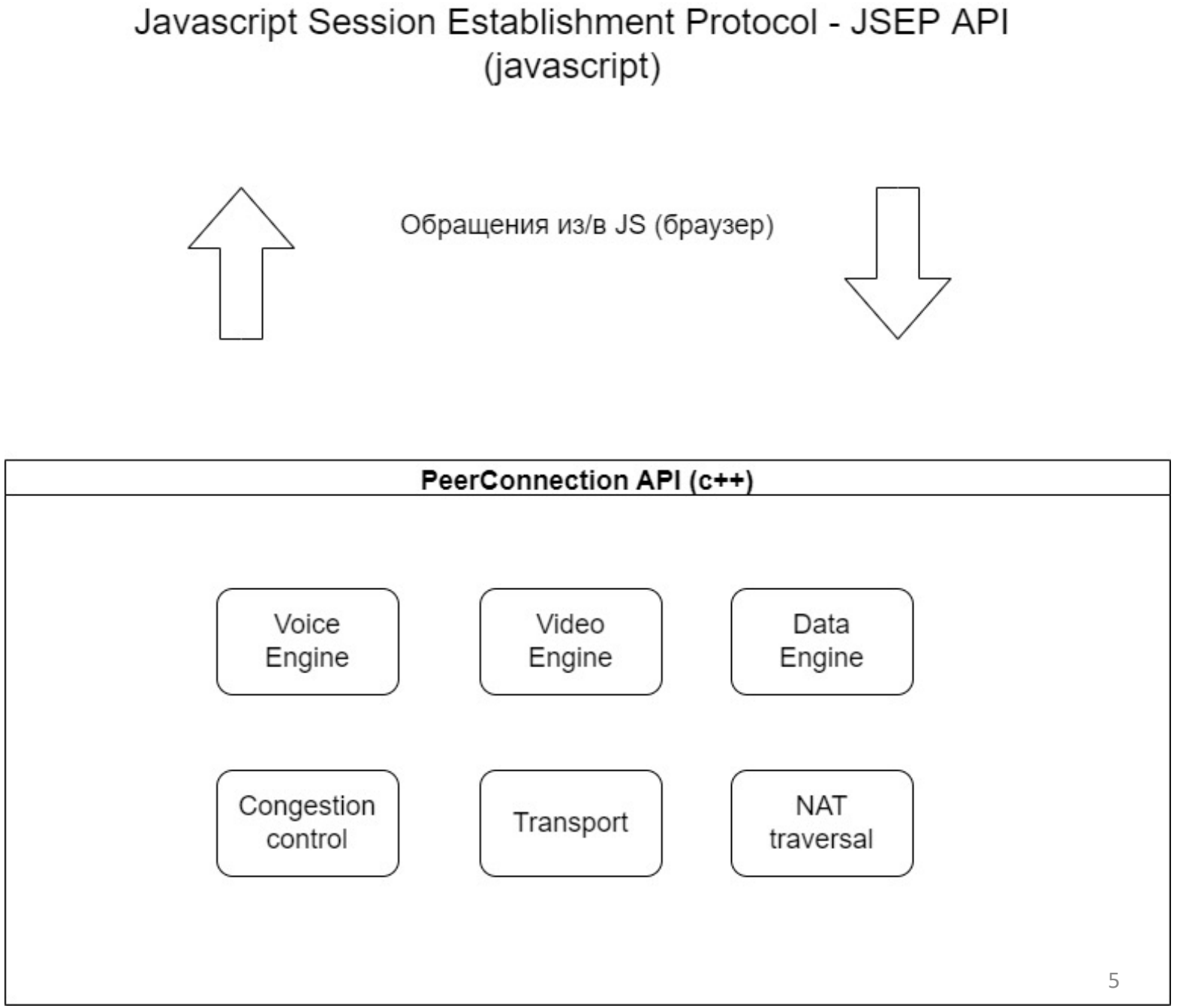
За:

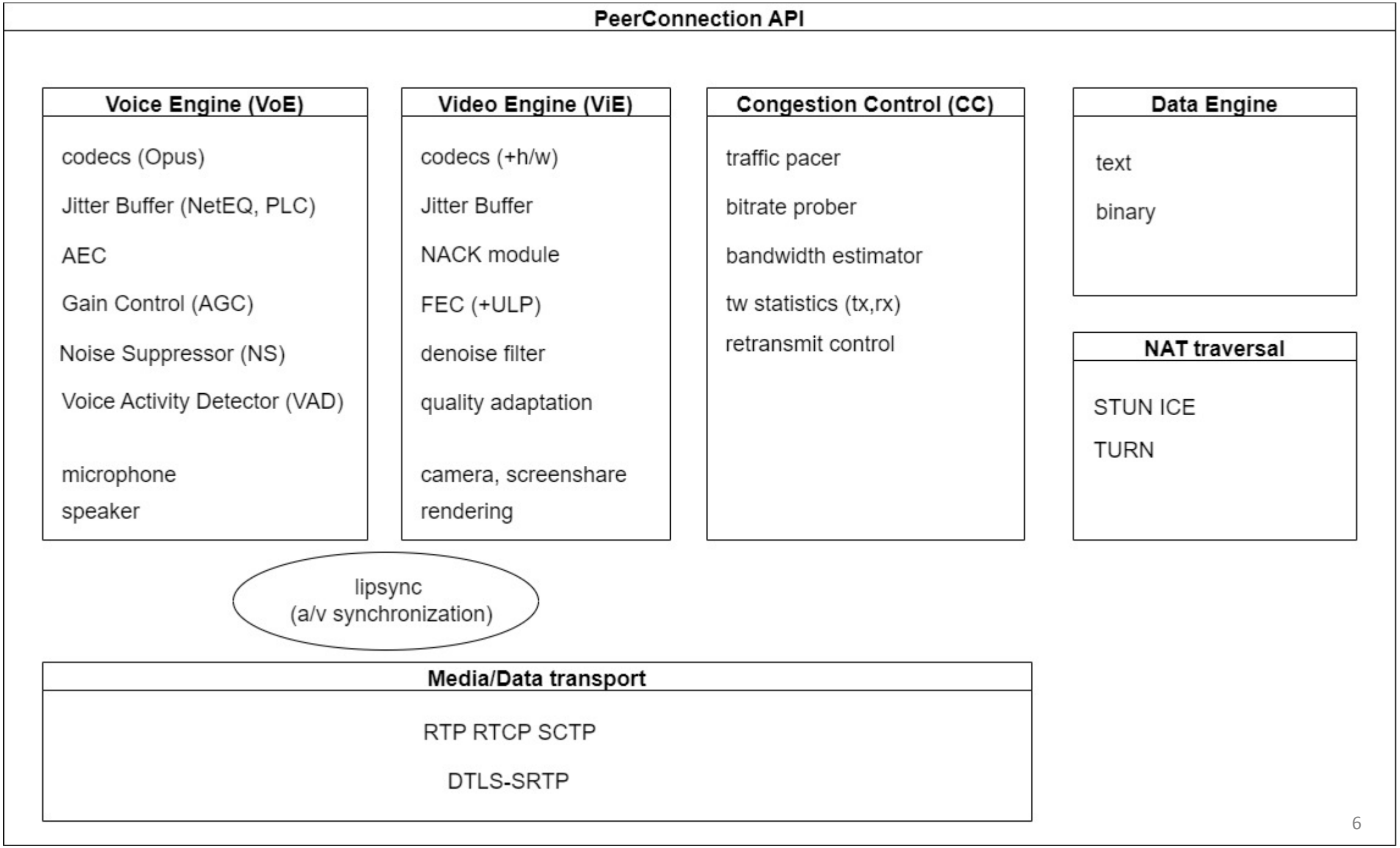
- **Экономия канала** на приемнике
- Снижение **ресурсопотребления** сервером
- Чем больше участников, тем больше экономия

Против:

- **Нагрузка на отправителя** – 3 энкодера
- Высокий **битрейт**

Взгляд сверху




PeerConnection API
Voice Engine (VoE)
codecs (Opus)
Jitter Buffer (NetEQ, PLC)
AEC
Gain Control (AGC)
Noise Suppressor (NS)
Voice Activity Detector (VAD)
microphone
speaker
Video Engine (ViE)
codecs (+h/w)
Jitter Buffer
NACK module
FEC (+ULP)
denoise filter
quality adaptation
camera, screenshare
rendering
Congestion Control (CC)
traffic pacer
bitrate prober
bandwidth estimator
tw statistics (tx,rx)
retransmit control
Data Engine
text
binary
NAT traversal
STUN ICE
TURN
lipsync
(a/v synchronization)
Media/Data transport
RTP RTCP SCTP
DTLS-SRTP

Voice Engine

- Codecs (opus)
- Jitter buffer (NetEQ, PLC)
- AEC
- Gain control (AGC)
- Noise suppressor
- Voice Activity Detector (VAD)

- Microphone
- Speaker

Video Engine

- Codecs (+h/w)
- Jitter buffer
- NACK module
- FEC (+ULP)
- Denoise filter
- Quality adaptation

- Camera, Screensharing
- Rendering (vulkan, metal, GL)

Congestion Control

- Outgoing traffic pacer
- Bandwidth estimator
- Bitrate prober
- Transport Statistics
  (transport wide CC stats)
- Retransmit control

Data engine (SCTP)

- Text
- Binary

NAT Traversal

videotech online

Антон Квятковский

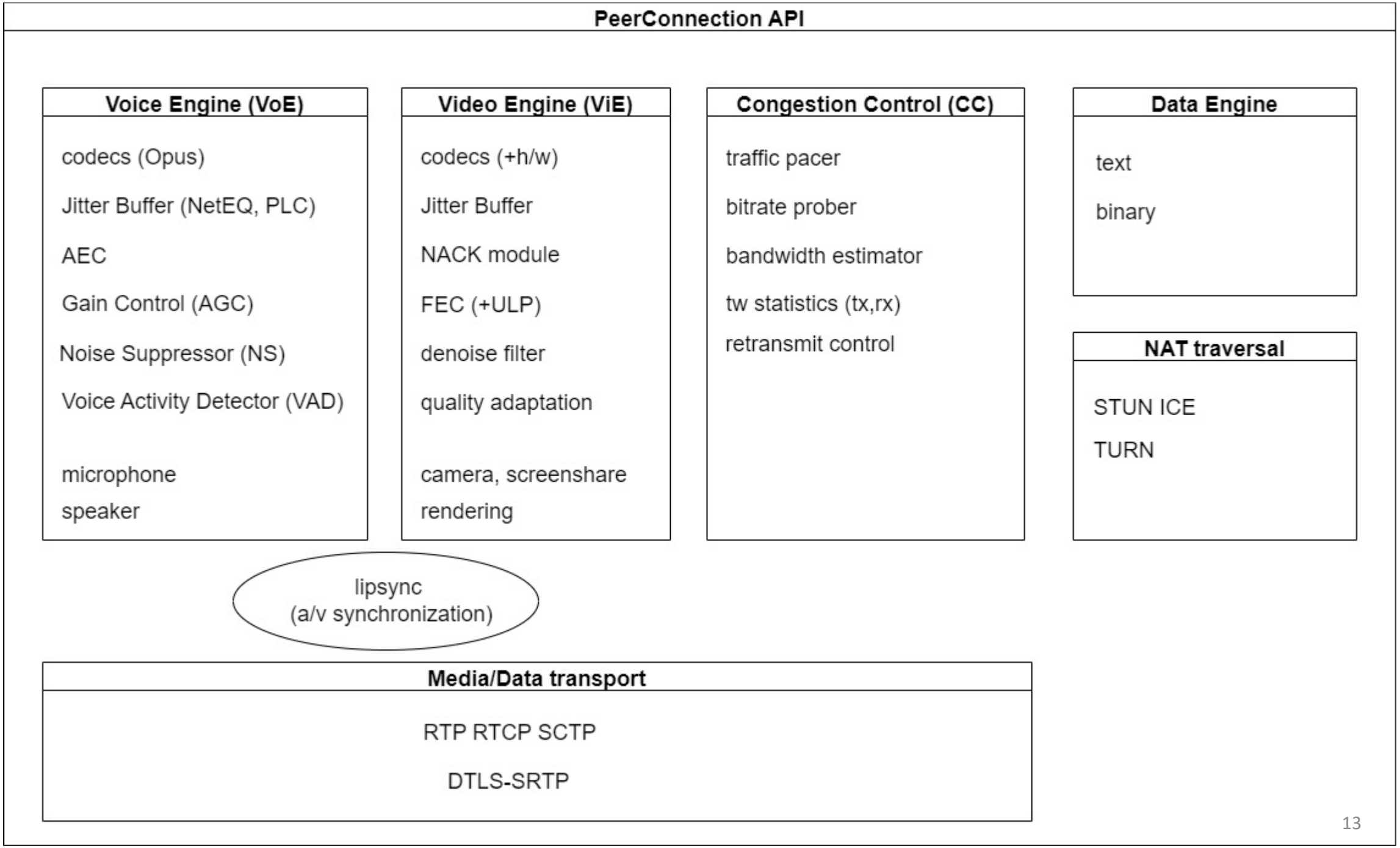
DINS

ICE-cold connection

пт, 3 декабря, 16:00

Media/Data transport

- RTP/RTCP
- SCTP
- DTLS
- AES (deprecated)
- SRTP


PeerConnection API
Voice Engine (VoE)
codecs (Opus)
Jitter Buffer (NetEQ, PLC)
AEC
Gain Control (AGC)
Noise Suppressor (NS)
Voice Activity Detector (VAD)
microphone
speaker
Video Engine (ViE)
codecs (+h/w)
Jitter Buffer
NACK module
FEC (+ULP)
denoise filter
quality adaptation
camera, screenshare
rendering
Congestion Control (CC)
traffic pacer
bitrate prober
bandwidth estimator
tw statistics (tx,rx)
retransmit control
Data Engine
text
binary
NAT traversal
STUN ICE
TURN
lipsync
(a/v synchronization)
Media/Data transport
RTP RTCP SCTP
DTLS-SRTP

Voice Engine

- Codecs (opus)
- Jitter buffer (NetEQ, PLC)
- AEC
- Gain control (AGC)
- Noise suppressor
- Voice Activity Detector (VAD)

- Microphone
- Speaker

Video Engine

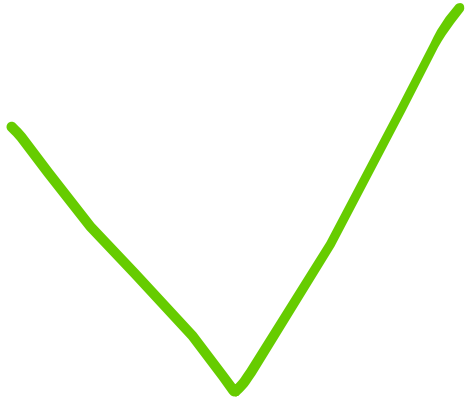
- Codecs (+h/w)
- Jitter buffer
- NACK module
- FEC (+ULP)
- Denoise filter
- Quality adaptation

- Camera, Screensharing
- Rendering (vulkan, metal, GL)

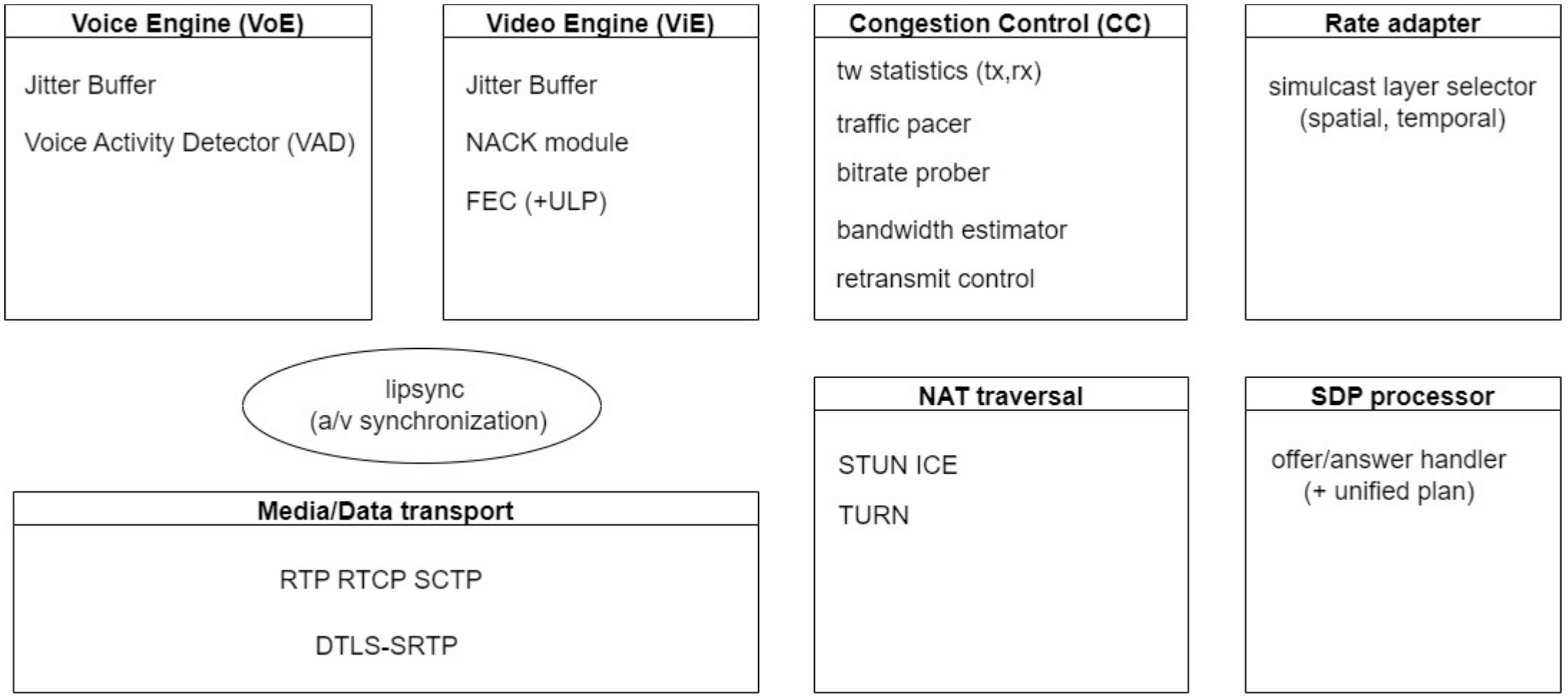
Data engine (SCTP)

- Text
- Binary

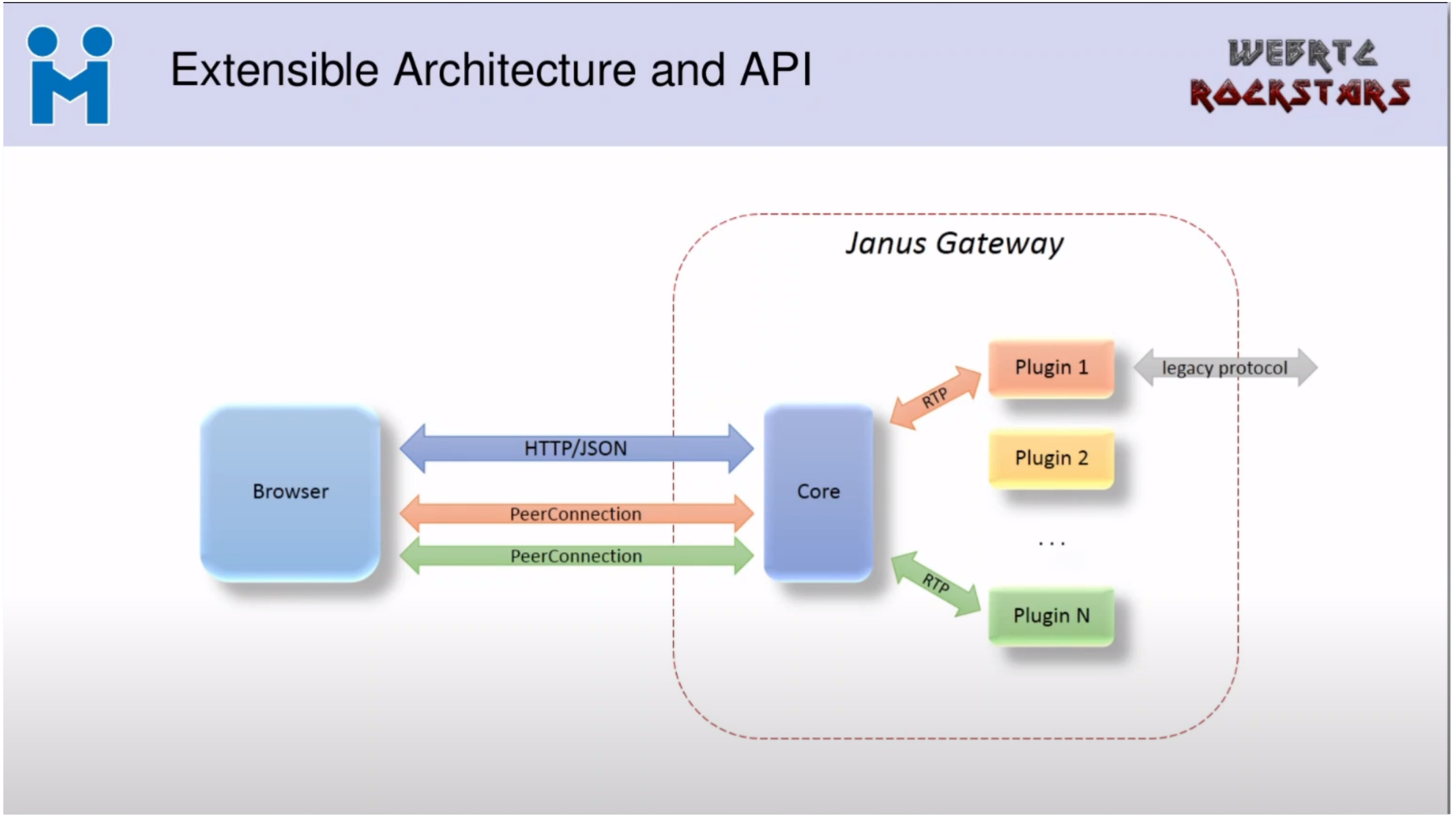
Congestion Control, NAT traversal, Transport



Строительные блоки для Simulcast SFU



А тем временем…Janus



Janus, как референс, в плане производительности

Почему:

- **Минимальный функционал** – только прием/пересылка трафика
- Поддержка **transport wide statistics**

Ограничения (by design):

- Нет **сглаживающей очереди**
- Нет **синхронизации** аудио/видео
- Нет **адаптации**

Первый старт. Сравнение с Janus.

Chrome + virtual I/O:
--use-fake-device-for-media-stream
--use-file-for-fake-audio-capture="1.wav" --use-file-for-fake-video-capture="Johnny_1280x720_60.y4m"

Анализ. От чего отталкиваемся

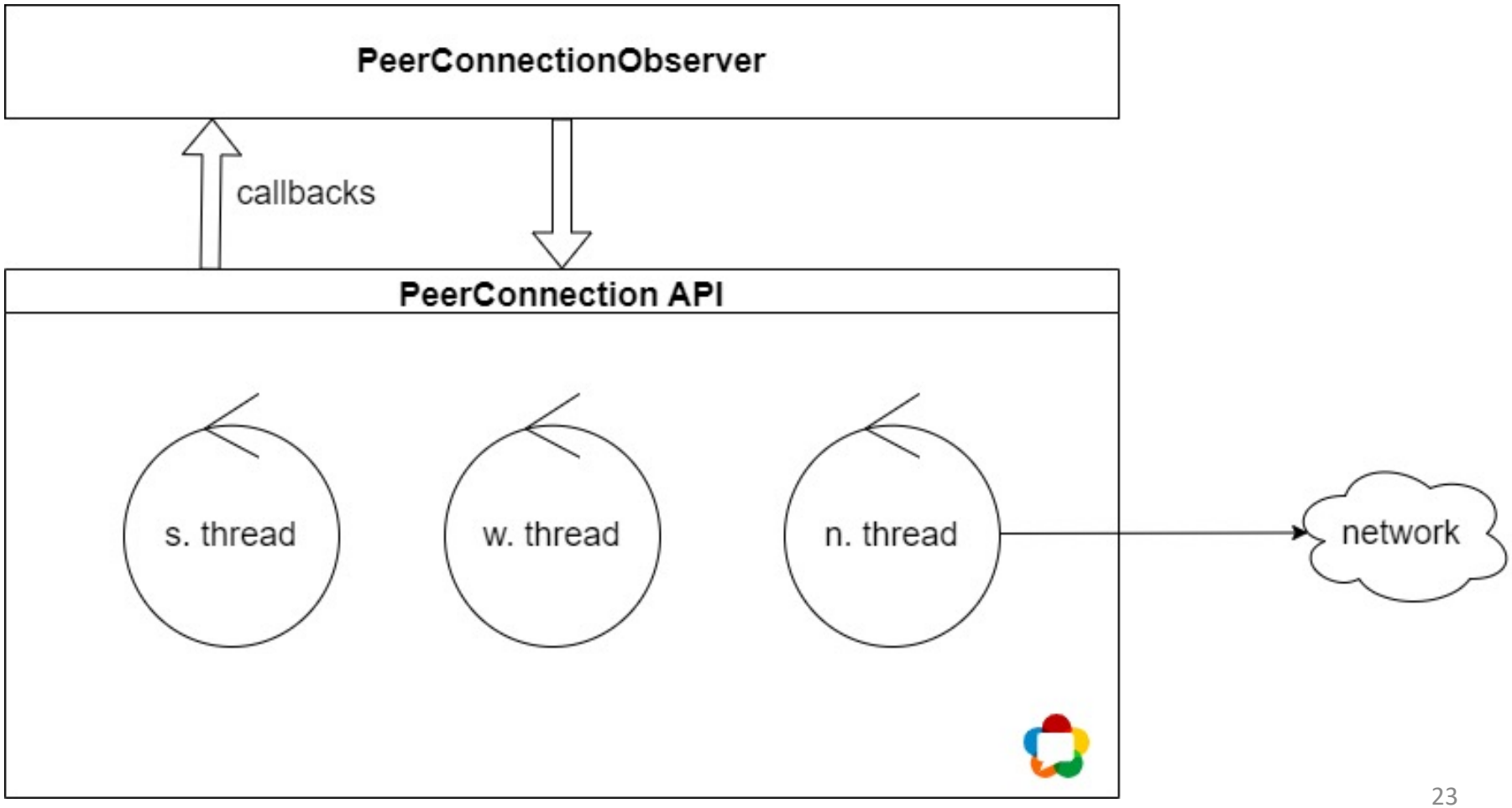
Janus	IMP (мы)
1.9-2.1%	4.7-5.0%

	raw_syscalls:sys_enter	sched:sched_switch	sched:sched_migrate_task	cache-misses
Janus	91,031	4,485	376	56,640,793
IMP	19,654	20,476	1,258	212,674,982

Много переключений контекста!

Откуда столько переключений и ожидания?

Потоковая модель. Вид с птичьего полета



Наводим резкость. Отправка

6 переключений на пакет

Насколько глубока кроличья нора?

Входящий пакет

Глубина стека: **38**

```
Thread-11
cricket::BaseChannel::OnRtpPacket channel.cc:477
webrtc::RtpDemuxer::OnRtpPacket rtp_demuxer.cc:177
webrtc::RtpTransport::DemuxPacket rtp_transport.cc:195
webrtc::SrtpTransport::OnRtpPacketReceived srtp_transport.cc:231
webrtc::RtpTransport::OnReadPacket rtp_transport.cc:269
sigslot::_opaque_connection::emitter<webrtc::RtpTransport, rtc::PacketTransportIntern
sigslot::_opaque_connection::emit<rtc::PacketTransportInternal*, char const*, unsigne
sigslot::signal_with_thread_policy<sigslot::single_threaded, rtc::PacketTransportInte
sigslot::signal_with_thread_policy<sigslot::single_threaded, rtc::PacketTransportInte
cricket::DtlsTransport::OnReadPacket dtls_transport.cc:622
sigslot::_opaque_connection::emitter<cricket::DtlsTransport, rtc::PacketTransportInte
sigslot::_opaque_connection::emit<rtc::PacketTransportInternal*, char const*, unsigne
sigslot::signal_with_thread_policy<sigslot::single_threaded, rtc::PacketTransportInte
sigslot::signal_with_thread_policy<sigslot::single_threaded, rtc::PacketTransportInte
cricket::P2PTransportChannel::OnReadPacket p2p_transport_channel.cc:2021
sigslot::_opaque_connection::emitter<cricket::P2PTransportChannel, cricket::Connectio
sigslot::_opaque_connection::emit<cricket::Connection*, char const*, unsigned long, l
sigslot::signal_with_thread_policy<sigslot::single_threaded, cricket::Connection*, ch
sigslot::signal_with_thread_policy<sigslot::single_threaded, cricket::Connection*, ch
cricket::Connection::OnReadPacket connection.cc:458
cricket::UDPPort::OnReadPacket stun_port.cc:419
cricket::UDPPort::HandleIncomingPacket stun_port.cc:348
cricket::AllocationSequence::OnReadPacket basic_port_allocator.cc:1609
sigslot::_opaque_connection::emitter<cricket::AllocationSequence, rtc::AsyncPacketSoc
sigslot::_opaque_connection::emit<rtc::AsyncPacketSocket*, char const*, unsigned long
sigslot::signal_with_thread_policy<sigslot::single_threaded, rtc::AsyncPacketSocket*,
sigslot::signal_with_thread_policy<sigslot::single_threaded, rtc::AsyncPacketSocket*,
rtc::AsyncUDPSocket::OnReadEvent async_udp_socket.cc:134
sigslot::_opaque_connection::emitter<rtc::AsyncUDPSocket, rtc::AsyncSocket*> sigslot.
sigslot::_opaque_connection::emit<rtc::AsyncSocket*> sigslot.h:327
sigslot::signal_with_thread_policy<sigslot::multi_threaded_local, rtc::AsyncSocket*>:
sigslot::signal_with_thread_policy<sigslot::multi_threaded_local, rtc::AsyncSocket*>:
rtc::SocketDispatcher::OnEvent physical_socket_server.cc:790
rtc::ProcessEvents physical_socket_server.cc:1379
rtc::PhysicalSocketServer::WaitEpoll physical_socket_server.cc:1620
rtc::PhysicalSocketServer::Wait physical_socket_server.cc:1328
rtc::Thread::Get thread.cc:468
rtc::Thread::ProcessMessages thread.cc:1021
rtc::Thread::Run thread.cc:809
webrtc_rtp_gw2::thread_wrapper<webrtc_rtp_gw2::worker_thread_tag>::Run thread_wrapper
rtc::Thread::PreRun thread.cc:798
```

Оптимизация. Зеленая зона. Избавление от клиентской специфики.

- Оставляем **один поток**
- Удаляем ставшую избыточной **синхронизацию** – меньше критических секций
- Удаляем **посылку сообщений**, делаем вызовы **напрямую**

Цифири

IMP (orig)	IMP (opt1)
4.7-5.0 %	3.5-4.0 %

	sched_switch	sched_migrate_task	epoll_wait	enter_futex	raw_syscalls: sys_enter	cache-misses
opt1	8,911	266	17,202	4,151	50,482	108,829,173
orig	20,476	1,258	25,566	14,772	91,031	212,674,982

Цифири

IMP (orig)	IMP (opt1)
4.7-5.0 %	3.5-4.0 %

-20%

	sched_switch	sched_migrate_task	epoll_wait	enter_futex	raw_syscalls: sys_enter	cache-misses
opt1	8,911	266	17,202	4,151	50,482	108,829,173
orig	20,476	1,258	25,566	14,772	91,031	212,674,982

Оптимизация, оранжевая зона?

Голоса в голове шепчут: «переключений по-прежнему много!»

- Заменяем все **repeating_task** в каждом компоненте на вызов из одного цикла
- **Кешируем** передаемые паметры за счет **протекающей абстракции** – у нас нет длительных вычислений, априорное знание
- Более экономное обращение с памятью – использование **пула**

Сводим воедино

Janus	IMP (no opt)	IMP (opt1)	IMP (opt2)
1.9-2.1%	4.7-5.0%	3.5-4.0%	1.6-1.7%

	sched_switch	sched_migrate_task	epoll_wait	enter_futex	raw_syscalls: sys_enter	cache-misses
Opt 2	6,178	161	6,720	1,257	20,217	58,160,307
Janus	4,485	376	-	-	19,654	56,640,793
Opt 1	8,911	266	17,202	4,151	50,482	108,829,173
original	20,476	1,258	25,566	14,772	91,031	212,674,982

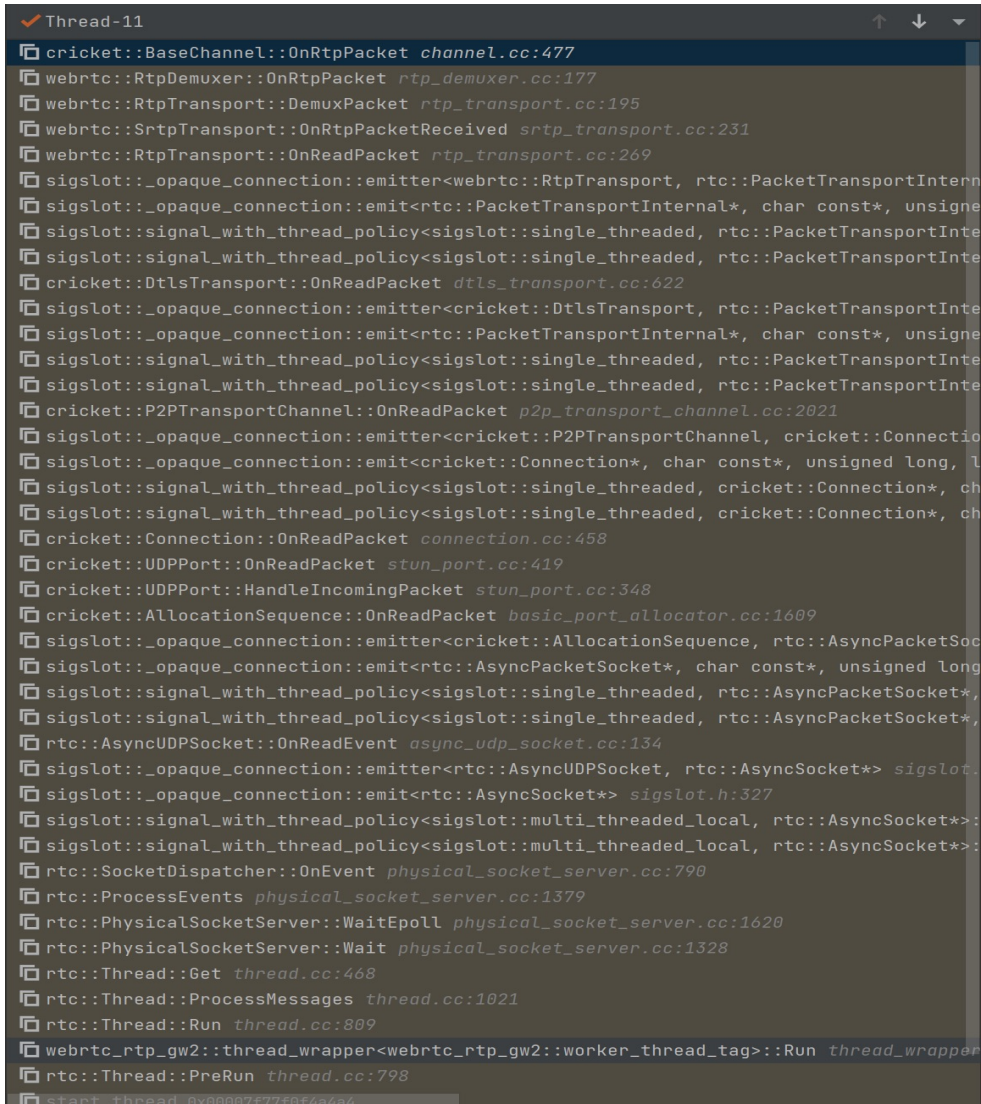
~~И чо?!~~ Что дальше?

Два параллельных направления:

- Дальнейшая оптимизация
- Работы по улучшению качества

Оптимизация.
Куда копать, с чем сравнивать, что измерять?

- Еще один заход в **профилирование**
- Переписывание **слоев работы с сокетами** для сокращения накладных расходов
- Оптимизация **работы с памятью**, использование арен



Качество.
Критерии, от чего отталкиваться, что измерять

Что именно проверять:

- Оценка **пропускной способности**
- **Адаптация**: скорость и глубина падения, скорость роста, адекватность переключений между слоями вниз, вверх
- **Компенсация задержек** в аудио и видео (JB).
  - Задержки могут быть **разными** (см. слайд ниже)
- **Синхронизация** аудио и видео

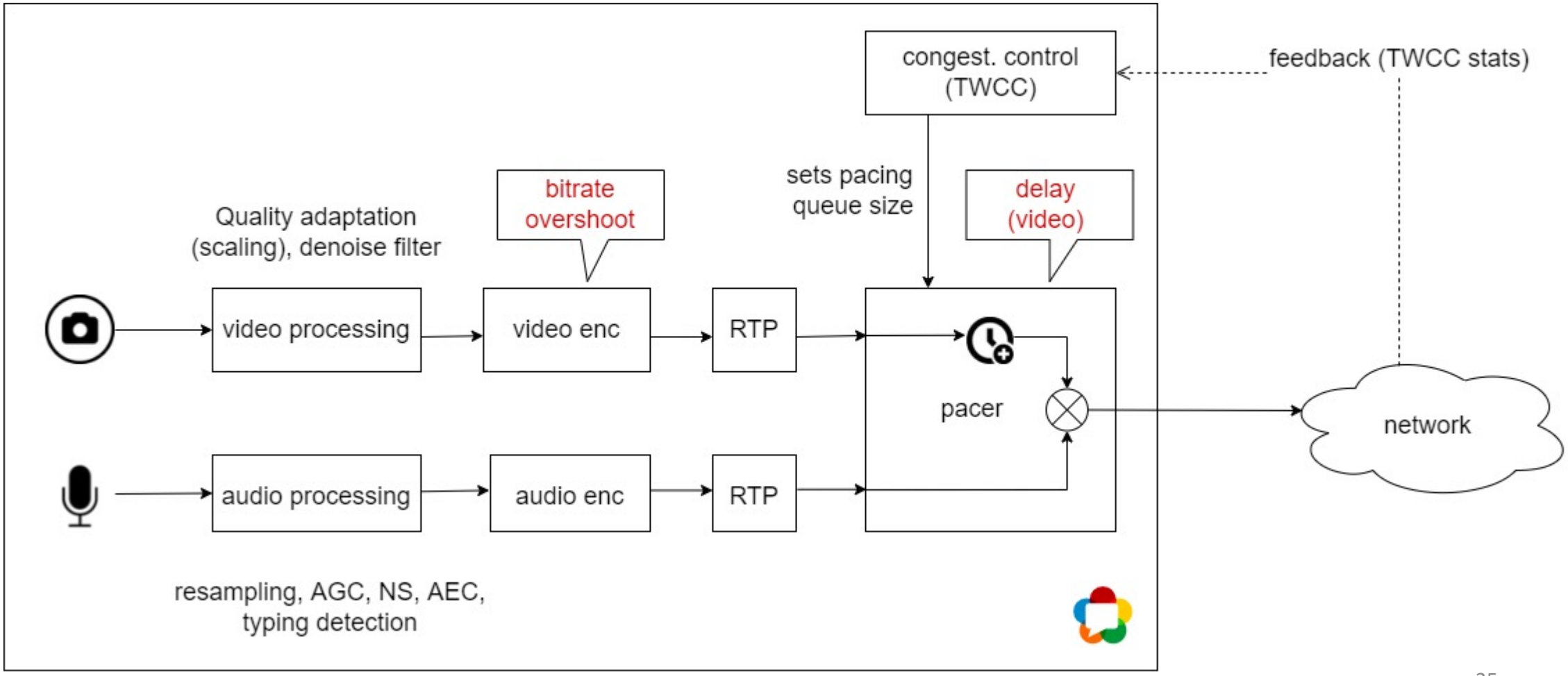
Тестовые условия:

- 1280x720@30, 640x480@30, 320x240@30 + opus
- Фиксированная пропускная способность: 3600mbps, 2.4mbps, 1mbps, 600kbps, 150kbps
- случайные потери: 0.1, 0.5, 1, 1.5, 2, 10%
- фиксированная задержка (+ в том числе RTCP):

Метрики:

- PESQ MOS
- скорость падения и роста, мс

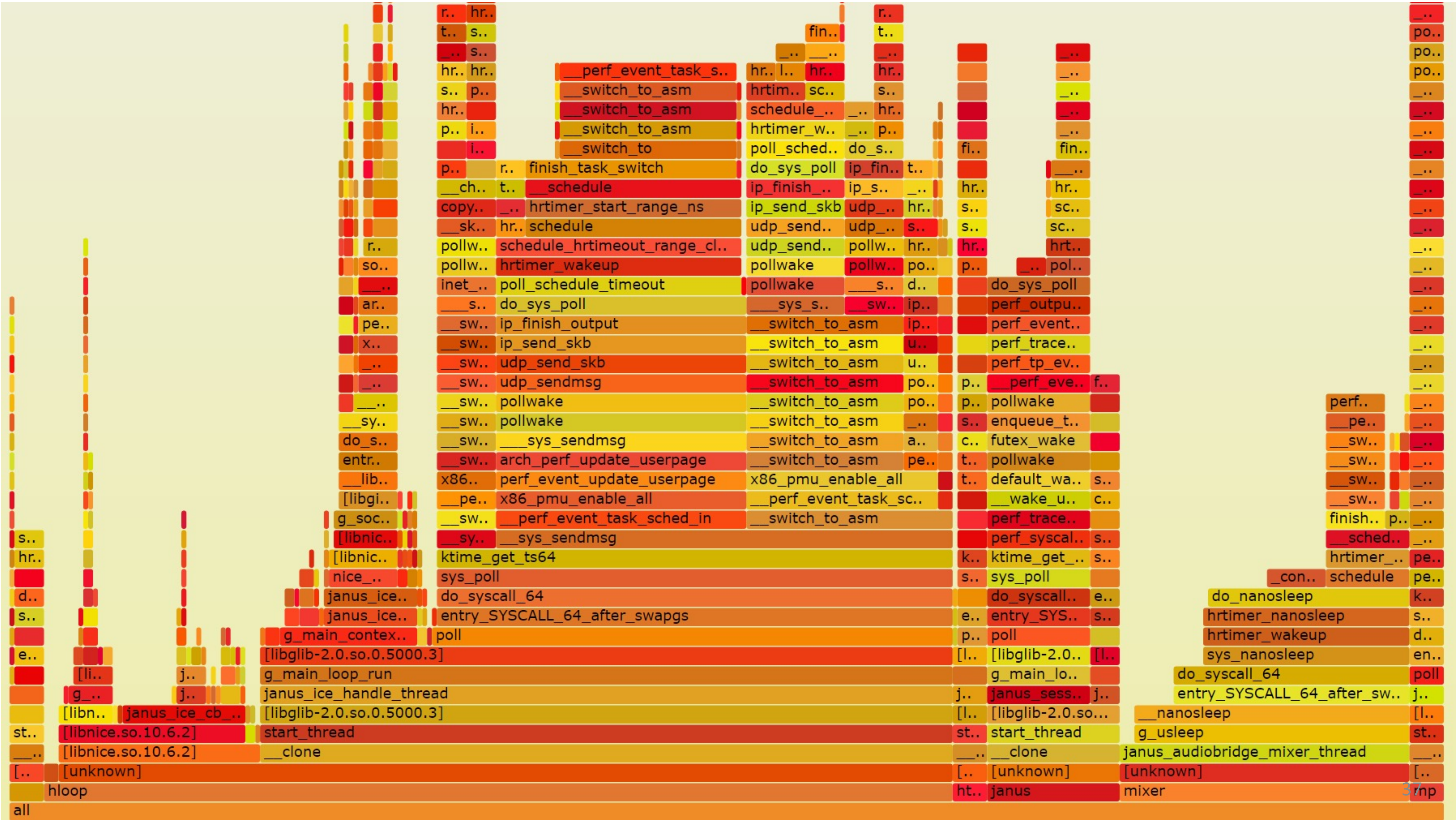
Для любопытных.
Появление задержки видео на отправляющей стороне (в клиенте)



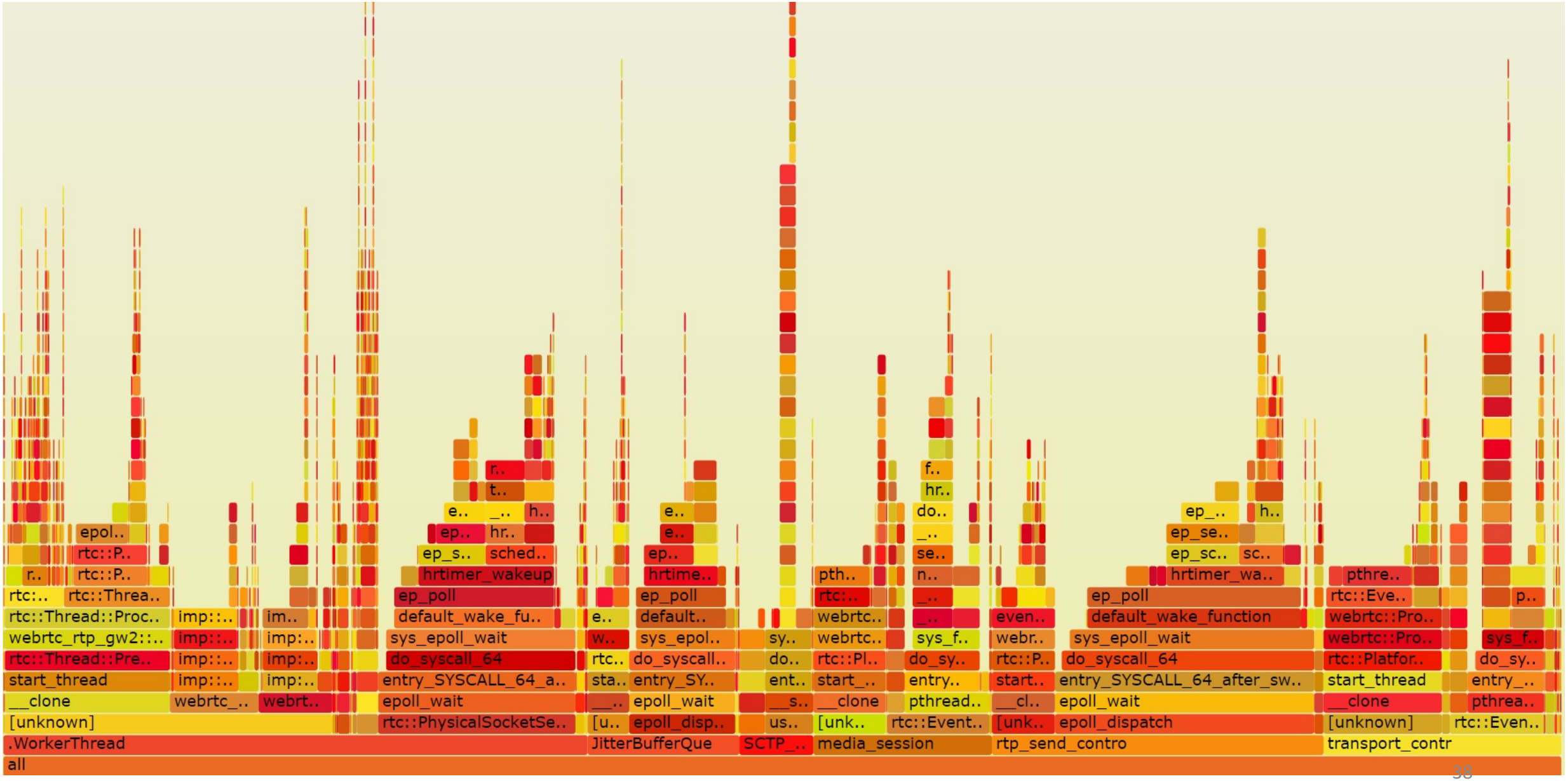
Нескомпенсированная задержка видео и Janus

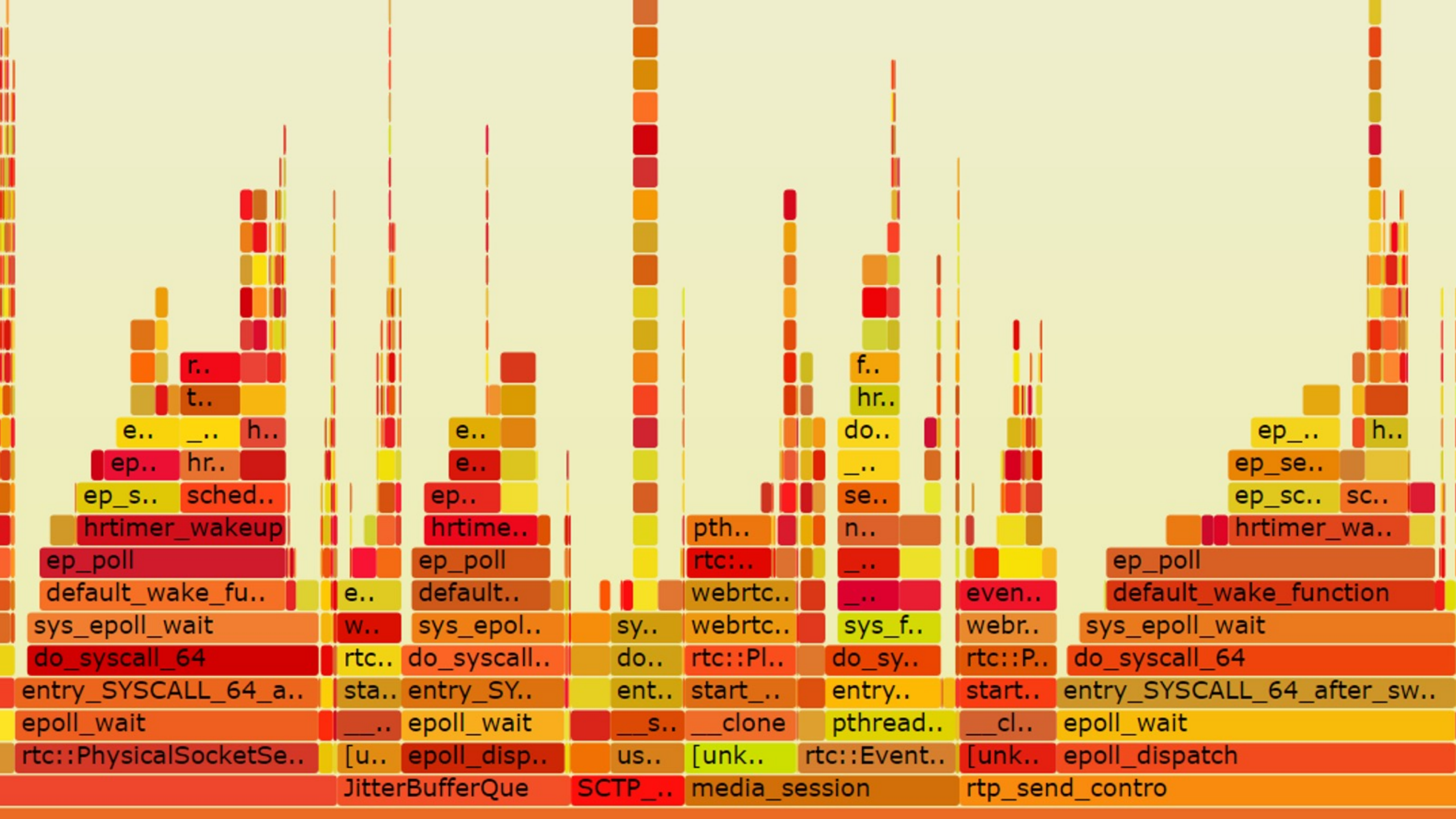
При микшировании аудио и видео потоков на сервере нескомпенсированная задержка с клиента приводит к **рассинхронизации**.

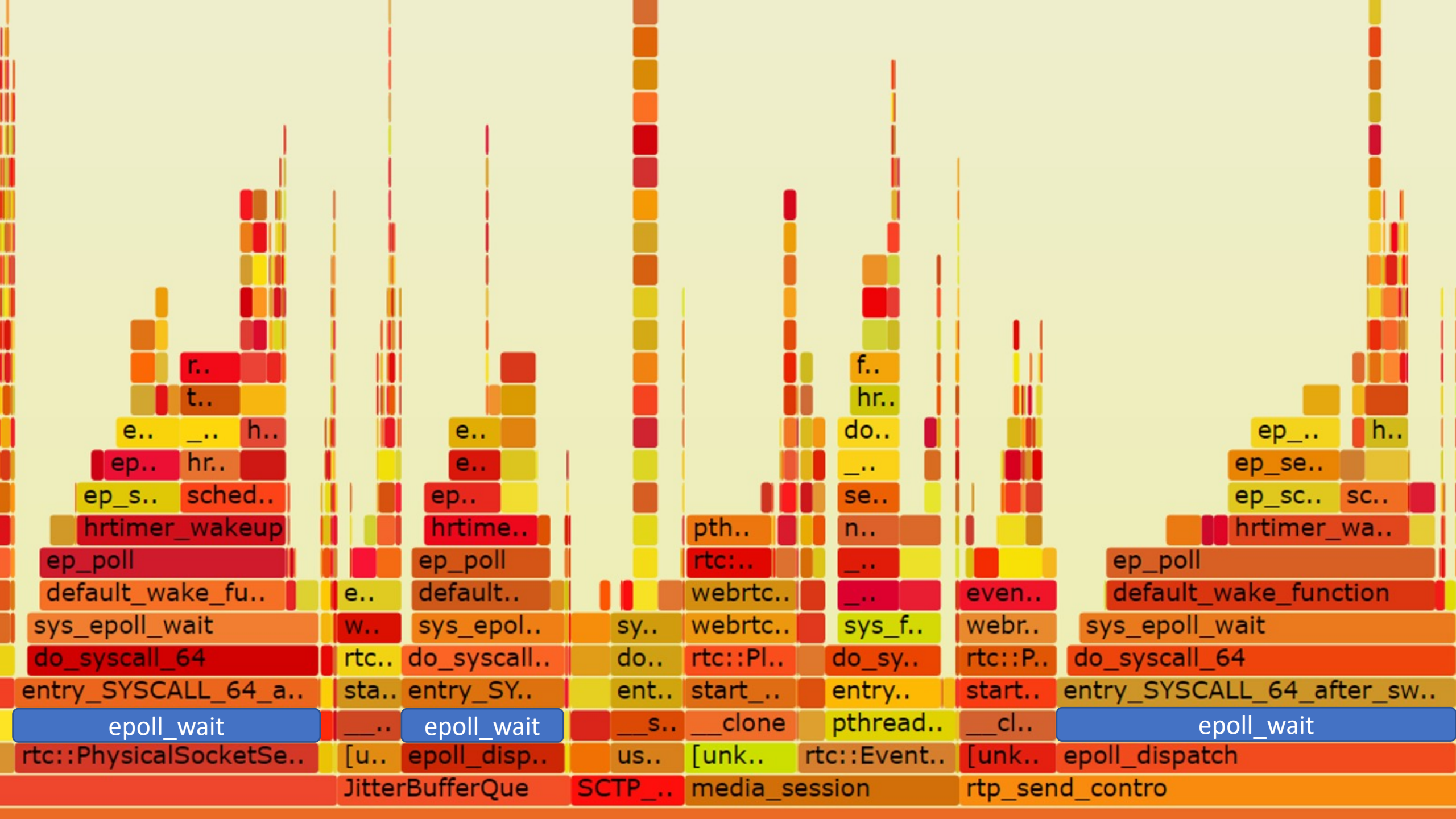
Для любопытных. Janus



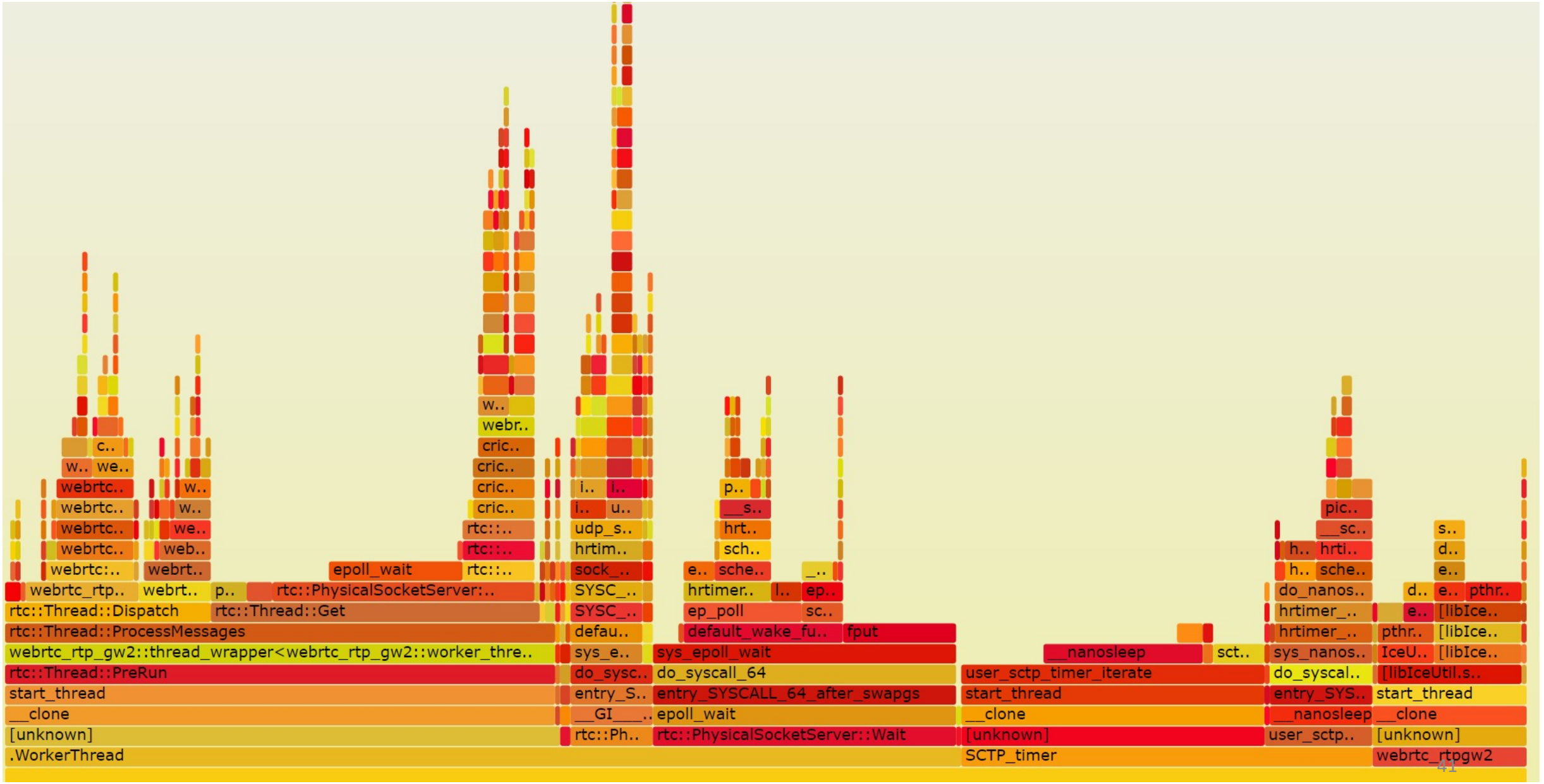
No opt


r..
t..
e..
_..
h..
epol..
ep..
hr..
rtc::P..
ep_s..
sched..
r..
rtc::P..
hrtimer_wakeup
rtc:..
rtc::Threa..
ep_poll
rtc::Thread::Proc..
imp::..
im..
default_wake_fu..
webrtc_rtp_gw2::..
imp::..
imp:..
sys_epoll_wait
rtc::Thread::Pre..
imp::..
imp:..
do_syscall_64
start_thread
imp::..
imp:..
entry_SYSCALL_64_a..
__clone
webrtc_..
webrt..
epoll_wait
[unknown]
rtc::PhysicalSocketSe..
.WorkerThread
JitterBufferQue
all
e..
e..
ep..
hrtime..
ep_poll
e..
default..
w..
sys_epol..
rtc..
do_syscall..
sta..
entry_SY..
___..
epoll_wait
[u..
epoll_disp..
f..
hr..
do..
_..
se..
pth..
n..
rtc:..
_..
webrtc..
_..
webrtc..
sys_f..
do..
rtc::Pl..
do_sy..
ent..
start_..
entry..
__s..
__clone
pthread..
us..
[unk..
rtc::Event..
SCTP_..
media_session
ep_..
h..
ep_se..
ep_sc..
sc..
hrtimer_wa..
ep_poll
default_wake_function
sys_epoll_wait
do_syscall_64
entry_SYSCALL_64_after_sw..
epoll_wait
epoll_dispatch
even..
webr..
rtc::P..
start..
__cl..
[unk..
rtp_send_contro
pthre..
rtc::Eve..
webrtc::Pro..
webrtc::Pro..
rtc::Platfor..
start_thread
__clone
[unknown]
transport_contr
p..
sys_f..
do_sy..
entry_..
pthrea..
rtc::Even..

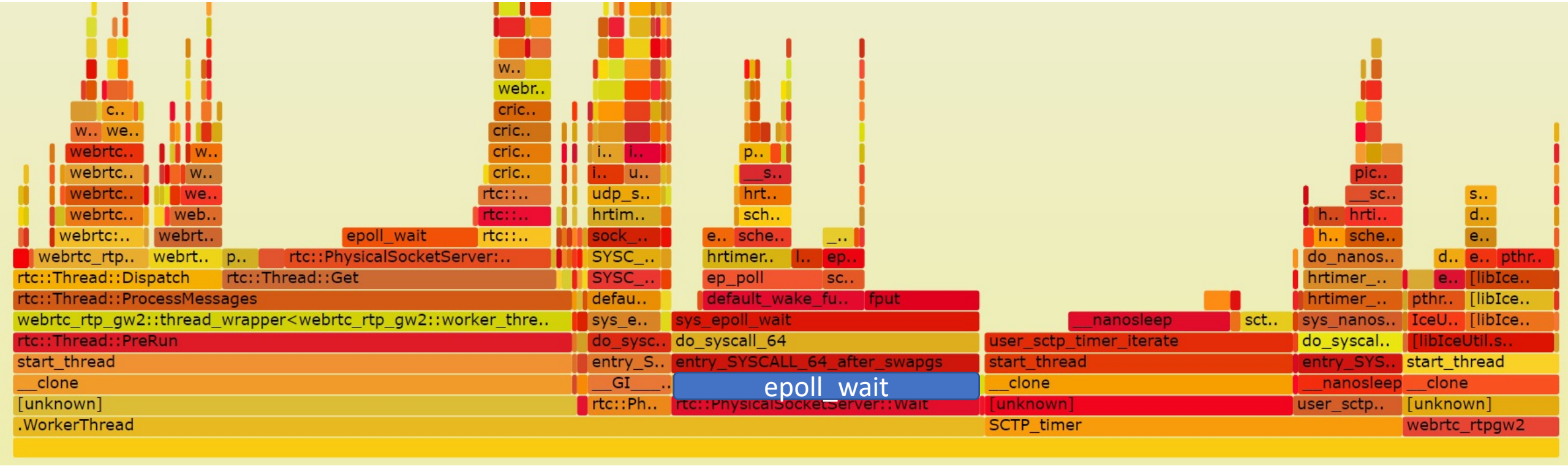

r..
t..
e..
_..
h..
ep..
hr..
ep_s..
sched..
hrtimer_wakeup
ep_poll
default_wake_fu..
sys_epoll_wait
do_syscall_64
entry_SYSCALL_64_a..
epoll_wait
rtc::PhysicalSocketSe..
e..
w..
rtc..
sta..
___..
[u..
JitterBufferQue
e..
e..
ep..
hrtime..
ep_poll
default..
sys_epol..
do_syscall..
entry_SY..
epoll_wait
epoll_disp..
sy..
do..
ent..
__s..
us..
SCTP_..
f..
hr..
do..
_..
se..
n..
pth..
rtc:..
webrtc..
webrtc..
rtc::Pl..
start_..
__clone
[unk..
media_session
do_sy..
entry..
pthread..
rtc::Event..
even..
webr..
rtc::P..
start..
__cl..
[unk..
rtp_send_contro
ep_..
h..
ep_se..
ep_sc..
sc..
hrtimer_wa..
ep_poll
default_wake_function
sys_epoll_wait
do_syscall_64
entry_SYSCALL_64_after_sw..
epoll_wait
epoll_dispatch


r..
t..
e..
_..
h..
ep..
hr..
ep_s..
sched..
hrtimer_wakeup
ep_poll
default_wake_fu..
sys_epoll_wait
do_syscall_64
entry_SYSCALL_64_a..
epoll_wait
rtc::PhysicalSocketSe..
e..
w..
rtc..
sta..
___..
[u..
JitterBufferQue
e..
e..
ep..
hrtime..
ep_poll
default..
sys_epol..
do_syscall..
entry_SY..
epoll_wait
epoll_disp..
sy..
do..
ent..
___s..
us..
SCTP_..
pth..
rtc:..
webrtc..
webrtc..
rtc::Pl..
start_..
__clone
[unk..
media_session
f..
hr..
do..
_..
se..
n..
_..
sys_f..
do_sy..
entry..
pthread..
rtc::Event..
even..
_..
webr..
rtc::P..
start..
__cl..
[unk..
rtp_send_contro
ep_..
h..
ep_se..
ep_sc..
sc..
hrtimer_wa..
ep_poll
default_wake_function
sys_epoll_wait
do_syscall_64
entry_SYSCALL_64_after_sw..
epoll_wait
epoll_dispatch

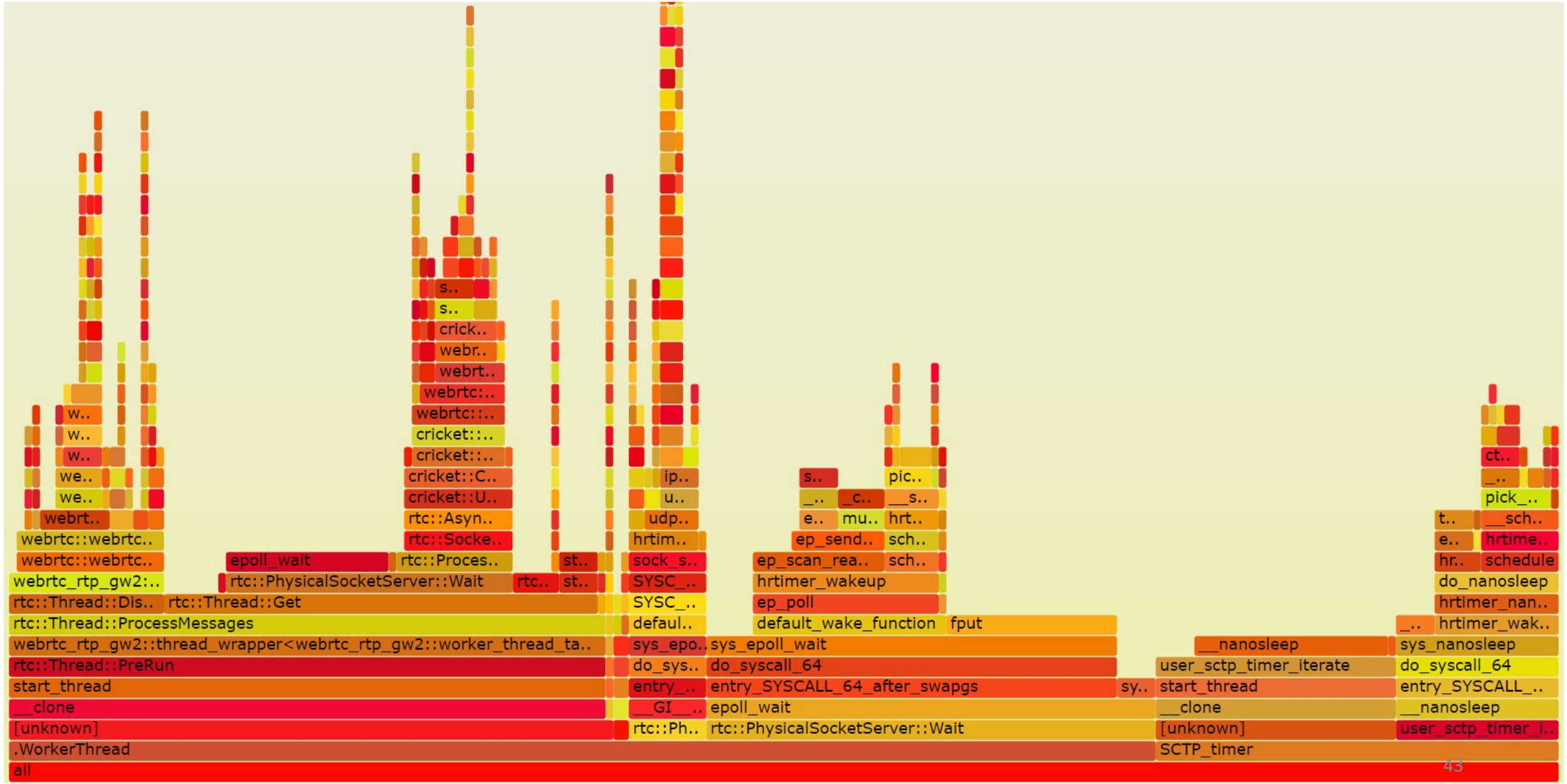
1 оптимизация

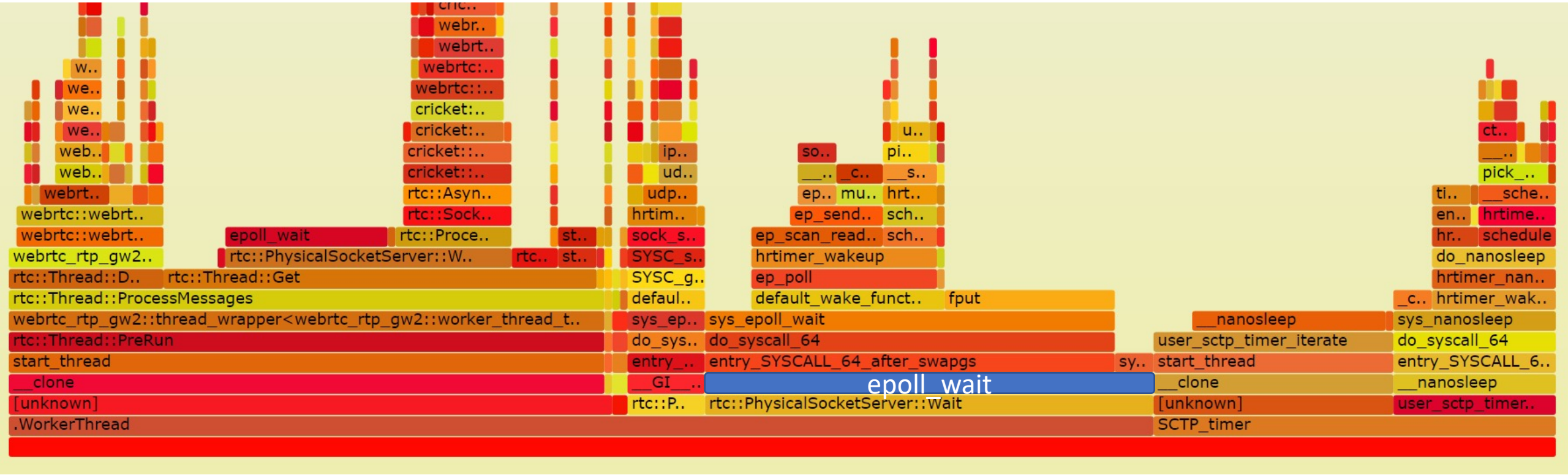


1 оптимизация



2 оптимизация




webr..
webrt..
webrtc:..
webrtc::..
cricket:..
cricket:..
cricket::..
cricket::..
rtc::Asyn..
rtc::Sock..
epoll_wait
rtc::Proce..
rtc::PhysicalSocketServer::W..
rtc..
st..
st..
ip..
ud..
udp..
hrtim..
sock_s..
SYSC_s..
SYSC_g..
defaul..
sys_ep..
do_sys..
entry_..
__GI__..
rtc::P..
so..
u..
pi..
__..
_c..
__s..
ep..
mu..
hrt..
ep_send..
sch..
ep_scan_read..
sch..
hrtimer_wakeup
ep_poll
default_wake_funct..
fput
sys_epoll_wait
do_syscall_64
entry_SYSCALL_64_after_swapgs
sy..
epoll_wait
rtc::PhysicalSocketServer::Wait
__nanosleep
user_sctp_timer_iterate
start_thread
__clone
[unknown]
SCTP_timer
ct..
__..
pick_..
ti..
__sche..
en..
hrtime..
hr..
schedule
do_nanosleep
hrtimer_nan..
_c..
hrtimer_wak..
sys_nanosleep
do_syscall_64
entry_SYSCALL_6..
__nanosleep
user_sctp_timer..
webrtc::webrt..
webrtc::webrt..
webrtc_rtp_gw2..
rtc::Thread::D..
rtc::Thread::Get
rtc::Thread::ProcessMessages
webrtc_rtp_gw2::thread_wrapper<webrtc_rtp_gw2::worker_thread_t..
rtc::Thread::PreRun
start_thread
__clone
[unknown]
.WorkerThread
w..
we..
we..
we..
web..
web..
webrt..

Итоги

- Из WebRTC **можно** сделать SFU сервер
- Для этого «**выбрать калибровку**», на которую ориентироваться при проведении оптимизации
- Подобрать **критерии качества** и **условия тестирования**

Благодарю за внимание!
Готов ответить на вопросы