

sysmocom

sysmocom - s.f.m.c. GmbH



OSmux: reduce of SAT uplink costs by protocol optimizations

by Holger Freyther, Harald Welte, and Pablo Neira Ayuso

Copyright © 2012-2017 sysmocom - s.f.m.c. GmbH

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with the Invariant Sections being just 'Foreword', 'Acknowledgements' and 'Preface', with no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

The AsciiDoc source code of this manual can be found at <http://git.osmocom.org/osmo-gsm-manuals/>

HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
0.1	11 June 2012	Initial version of the proposal for internal discussion.	Pablo Neira Ayuso
0.2	11 June 2012	Second version after comments from Holger and Harald: Include figures that provide expect traffic savings (in %). Change licensing terms (owned by OnWaves and consultants). Adjust work from 200 to 150 hours, remove details on how the implementation	Pablo Neira Ayuso
0.3	20 June 2017	Improve and extenend for osmo-gsm-manuals inclusion from Pau Espin: Convert to asciidoc. Update frame bits according to implementation.	Pau Espin Pedrol
0.4	19 July 2017	Add sequence charts, generate packet header structure with packetdiag, generate traffic saving plot with pychart.	Pau Espin Pedrol

Contents

1 Problem	1
2 Proposed solution	1
2.1 Voice	1
2.2 Signalling	2
3 OSmux protocol	2
3.1 LAPD Signalling (0)	2
3.2 AMR Codec (1)	3
3.2.1 Additional considerations	4
3.3 Dummy (2)	4
3.3.1 Additional considerations	4
4 Sequence Charts	5
4.1 Trunking	5
4.2 Batching	5
4.3 Trunking and Batching	6
4.4 Marker bit	7
5 Evaluation: Expected traffic savings	8
6 Other proposed follow-up works	9
6.1 Encryption	9
6.2 Multiple OSmux messages in one packet	10

1 Problem

In case of satellite based GSM systems, the transmission cost on the back-haul is relatively expensive. The billing for such SAT uplink is usually done in a pay-per-byte basis. Thus, reducing the amount of bytes transferred would significantly reduce the cost of such uplinks. In such environment, even seemingly small protocol optimizations, eg. message batching and trunking, can result in significant cost reduction.

This is true not only for speech codec frames, but also for the constant background load caused by the signalling link (A protocol). Optimizations in this protocol are applicable to both VSAT back-haul (best-effort background IP) as well as Inmarsat based links (QoS with guaranteed bandwidth).

2 Proposed solution

In order to reduce the bandwidth consumption, this document proposes to develop a multiplex protocol that will be used to proxy voice and signalling traffic through the SAT links.

2.1 Voice

For the voice case, we propose a protocol that provides:

- **Batching:** that consists of putting multiple codec frames on the sender side into one single packet to reduce the protocol header overhead. This batch is then sent as one RTP/UDP/IP packet at the same time. Currently, AMR 5.9 codec frames are transported in a RTP/UDP/IP protocol stacking. This means there are 15 bytes of speech codec frame, plus a 2 byte RTP payload header, plus the RTP (12 bytes), UDP (8 bytes) and IP (20 bytes) overhead. This means we have 40 byte overhead for 17 byte payload.
- **Trunking:** in case of multiple concurrent voice calls, each of them will generate one speech codec frame every 20ms. Instead of sending only codec frames of one voice call in a given IP packet, we can *interleave* or trunk the codec frames of multiple calls into one IP. This further increases the IP packet size and thus improves the payload/overhead ratio.

Both techniques should be applied without noticeable impact in terms of user experience. As the satellite back-haul has very high round trip time (several hundred milliseconds), adding some more delay is not going to make things significantly worse.

For the batching, the idea consists of batching multiple codec frames on the sender side, A batching factor (B) of 4 means that we will send 4 codec frames in one underlying protocol packet. The additional delay of the batching can be computed as $(B-1)*20ms$ as 20ms is the duration of one codec frame. Existing experimentation has shown that a batching factor of 4 to 8 (causing a delay of 60ms to 140ms) is acceptable and does not cause significant quality degradation.

The main requirements for such voice RTP proxy are:

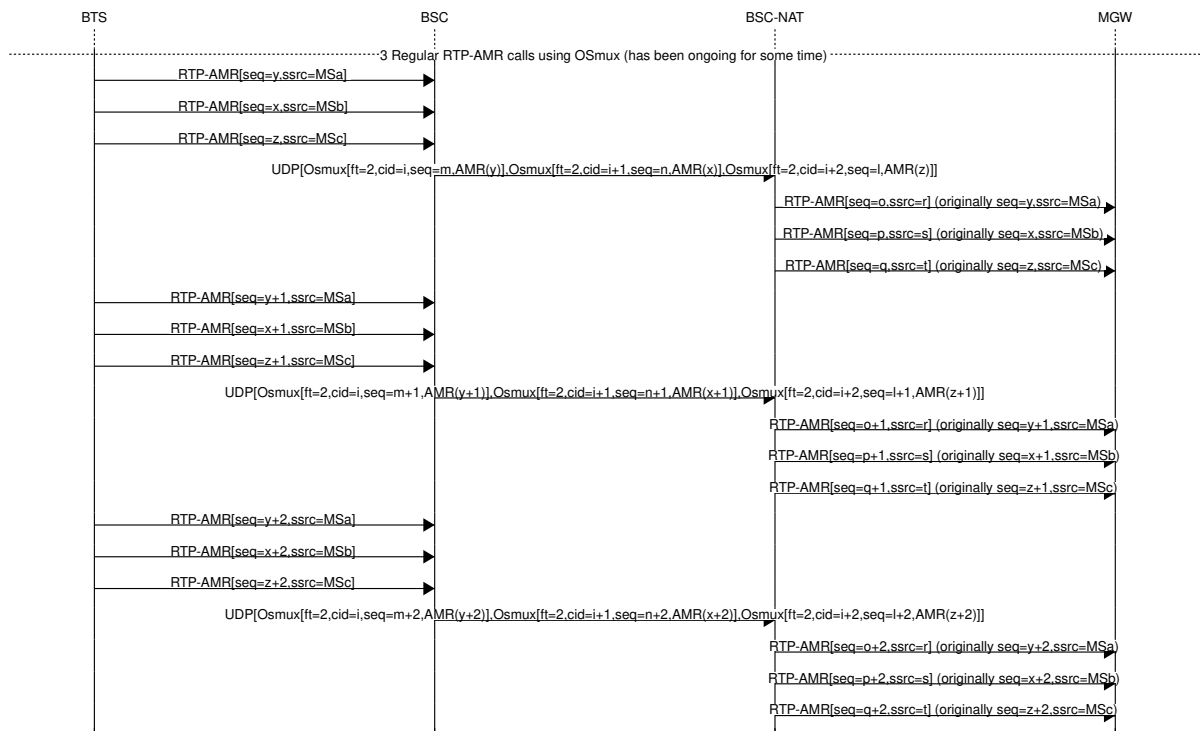
- Always batch codec frames of multiple simultaneous calls into single UDP message.
- Batch configurable number codec frames of the same call into one UDP message.
- Make sure to properly reconstruct timing at receiver (non-bursty but one codec frame every 20ms).
- Implementation in libosmo-netif to make sure it can be used in osmo-bts (towards osmo-bsc), osmo-bsc (towards osmo-bts and osmo-bsc_nat) and osmo-bsc_nat (towards osmo-bsc)
- Primary application will be with osmo-bsc connected via satellite link to osmo-bsc_nat.
- Make sure to properly deal with SID (silence detection) frames in case of DTX.
- Make sure to transmit and properly re-construct the M (marker) bit of the RTP header, as it is used in AMR.
- Primary use case for AMR codec, probably not worth to waste extra payload byte on indicating codec type (amr/hr/fr/efr). If we can add the codec type somewhere without growing the packet, we'll do it. Otherwise, we'll skip this.

4 Sequence Charts

4.1 Trunking

Following chart shows how trunking works for 3 concurrent calls from different MS on a given BTS. In this case only uplink data is shown, but downlink follows the same idea. Batching factor is set to 1 to easily illustrate trunking mechanism.

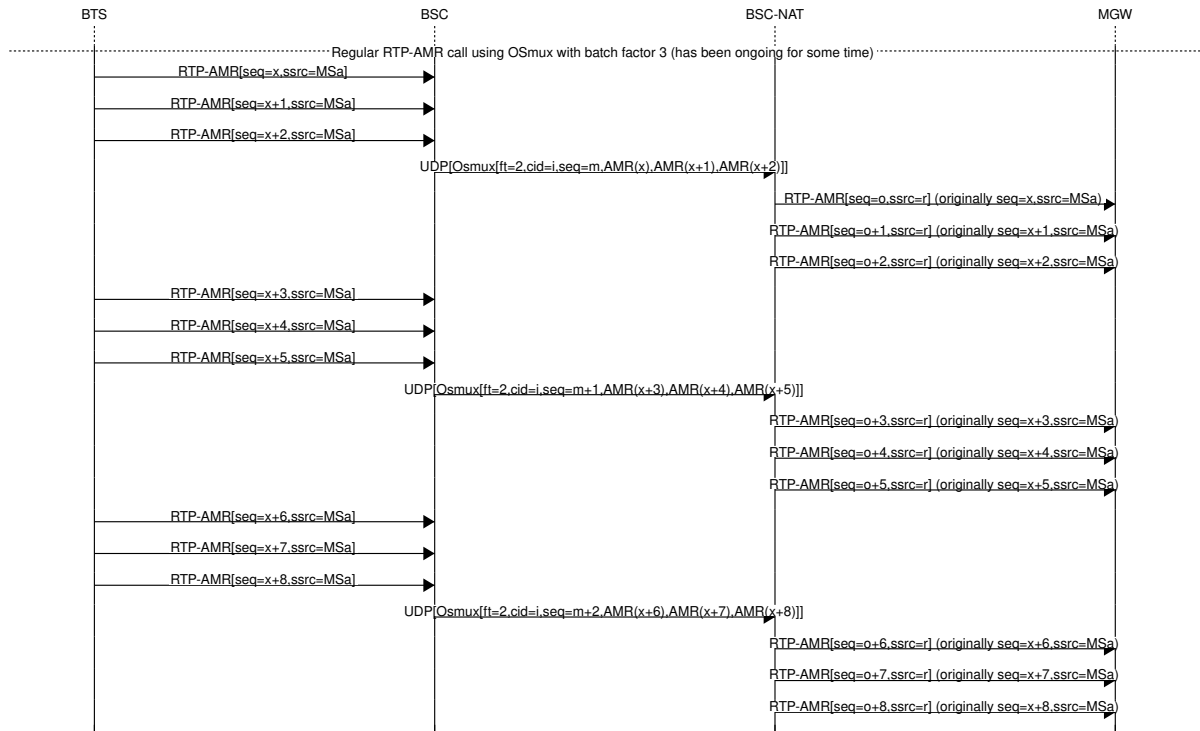
It can be seen how 3 RTP packets from 3 different Ms (a, b, and c) arrive to the BSC from the BTS. The BSC generates 3 OSmux frames and stores and sends them together in one UDP packet to the BSC-NAT. The BSC-NAT decodes the three OSmux frames, identifies each of them through CID values and transform them back to RTP before sending them to the MGW.



4.2 Batching

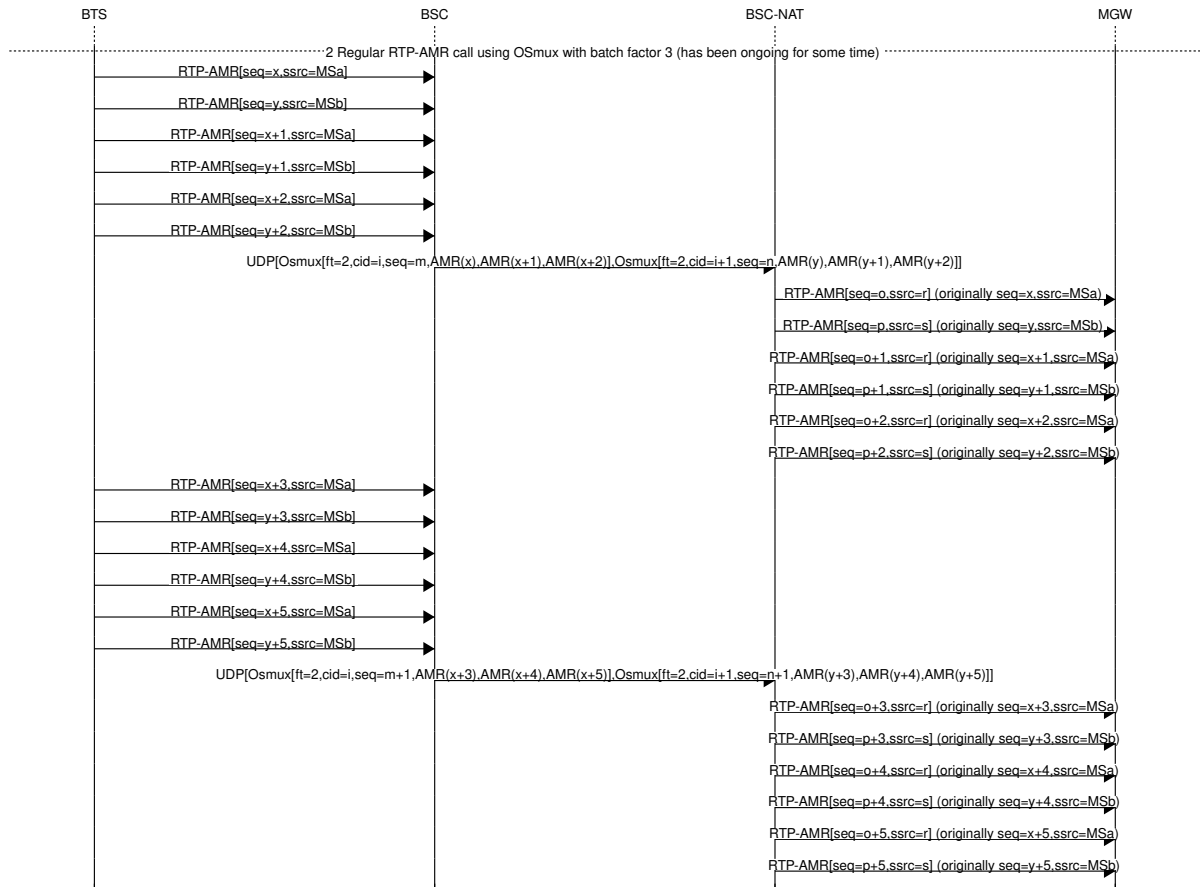
Following chart shows how batching with a factor of 3 works. To easily illustrate batching, only uplink and one concurrent call is considered.

It can be seen how 3 RTP packets from MSa arrive to the BSC from the BTS. The BSC queues the 3 RTP packets and once the batchfactor is reached, an OSmux frame is generated and sent to the BSC-NAT. The BSC-NAT decodes the OSmux frames, transforms each AMR payload into an RTP packet and each RTP packet is scheduled for delivery according to expected proportional time delay (and timestamp field is set accordingly).



4.3 Trunking and Batching

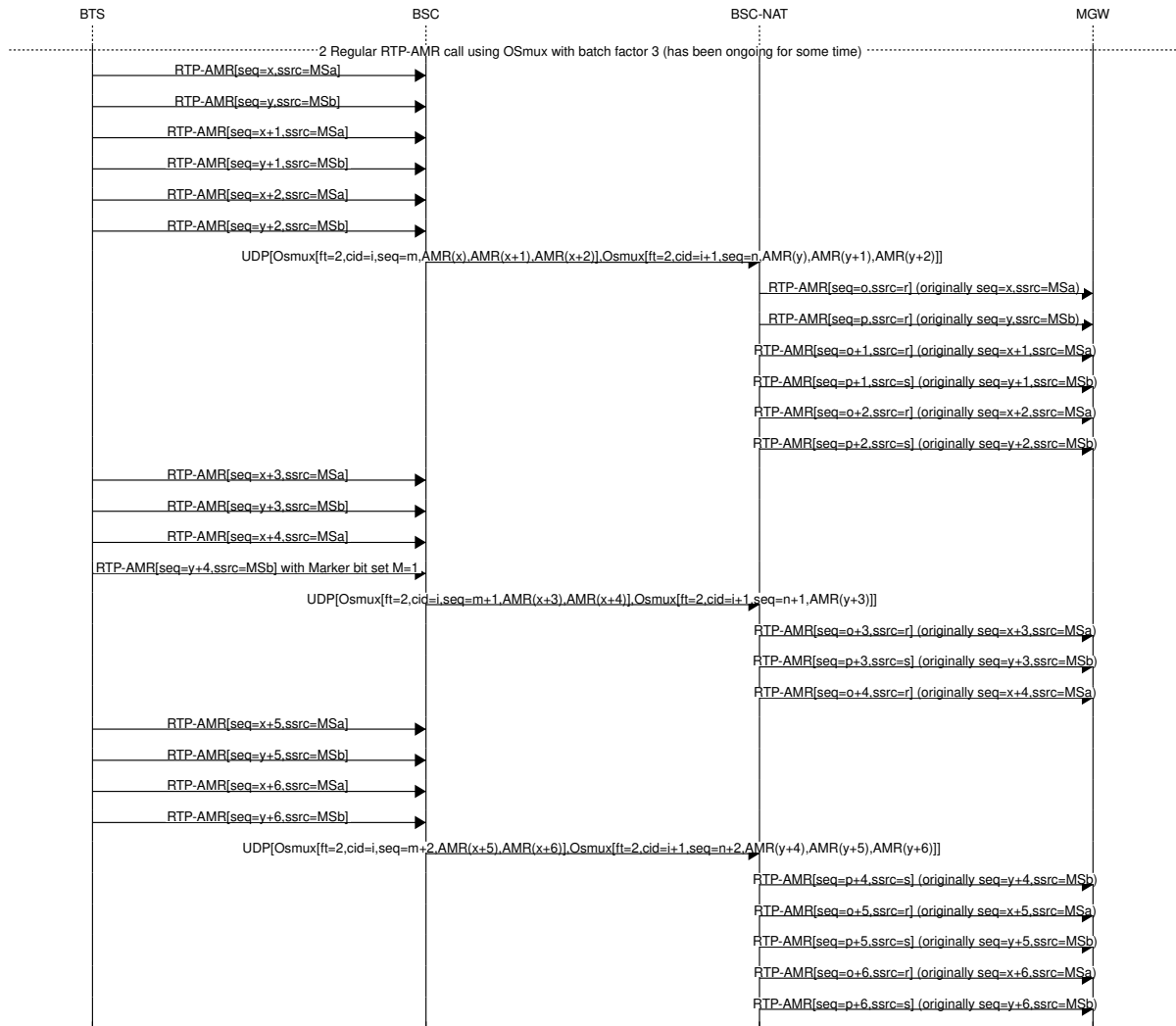
Following chart shows how trunking and batching work together. The chart shows 2 concurrent calls from different MS on a given BTS, and BSC is configured with a batch factor of 3. Again only uplink data is shown, but downlink follows the same idea. Batching factor is set to 1 to easily illustrate trunking mechanism.



4.4 Marker bit

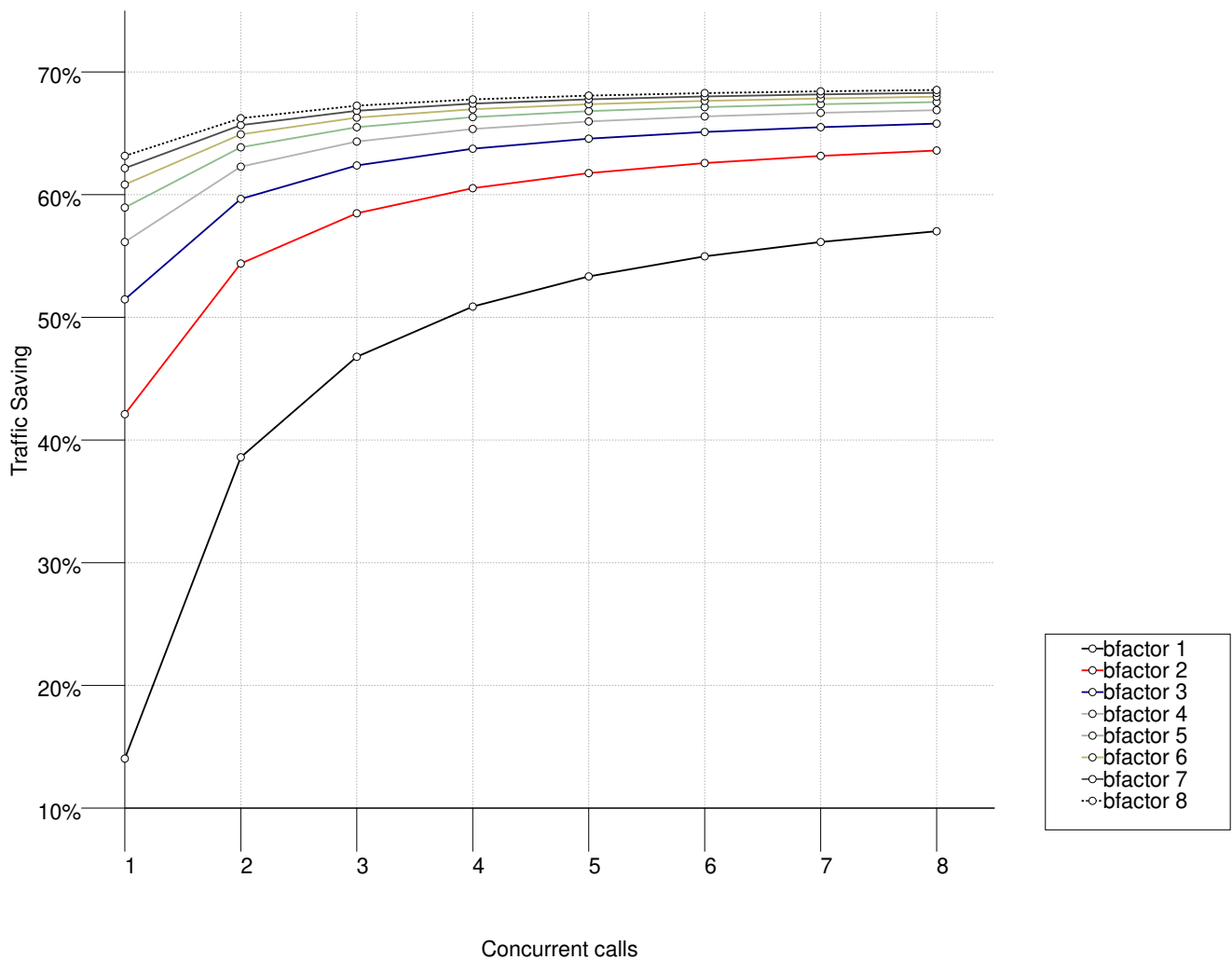
As described earlier, the Marker bit is always expected to relate to the first AMR payload of an Osmux frame. Thus, special considerations may be followed when the Osmux encoder receives an RTP packet with a marker bit. For instance, previously enqueued RTP packets may be sent even if the configured batch factor is not reached.

We again use the scenario with 2 concurrent calls and a batch factor of 3.



5 Evaluation: Expected traffic savings

The following figure shows the growth in traffic saving (in %) depending on the number of concurrent numbers of callings for a given set of batching factor values:



The results show a saving of 15.79% with only one concurrent call and with batching disabled (bfactor 1), that quickly improves with more concurrent calls (due to trunking).

By increasing the batching of messages to 4, the results show a saving of 56.68% with only one concurrent call. Trunking slightly improves the situation with more concurrent calls.

A batching factor of 8 provides very little improvement with regards to batching 4 messages. Still, we risk to degrade user experience. Thus, we consider a batching factor of 3 and 4 is adequate.

6 Other proposed follow-up works

The following sections describe features that can be considered in the mid-run to be included in the OSmux infrastructure. They will be considered for future proposals as extensions to this work. Therefore, they are NOT included in this proposal.

6.1 Encryption

Voice streams within OSmux can be encrypted in a similar manner to SRTP (RFC3711). The only potential problem is the use of a reduced sequence number, as it wraps in $(20\text{ms} * 2^{256} * B)$, i.e. 5.12s to 40.96s. However, as the receiver knows at which rate the codec frames are generated at the sender, he should be able to compute how much time has passed using his own timebase.

Another alternative can be the use of DTLS (RFC 6347) that can be used to secure datagram traffic using TLS facilities (libraries like openssl and gnutls already support this).

6.2 Multiple OSmux messages in one packet

In case there is already at least one active voice call, there will be regular transmissions of voice codec frames. Depending on the batching factor, they will be sent every 70ms to 140ms. The size even of a batched (and/or trunked) codec message is still much lower than the MTU.

Thus, any signalling (related or unrelated to the call causing the codec stream) can just be piggy-backed to the packets containing the voice codec frames.