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

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

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).

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.

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.

Signalling

Signalling uses SCCP/IPA/TCP/IP stacking. Considering SCCP as payload, this adds 3 (IPA) + 20 (TCP) + 20 (IP) = 43 bytes overhead for every signalling message, plus of course the 40-byte-sized TCP ACK sent in the opposite direction.

While trying to look for alternatives, we consider that none of the standard IP layer 4 protocols are suitable for this application. We detail the reasons why:

- TCP is a streaming protocol aimed at maximizing the throughput of a stream within the constraints of the underlying transport layer. This feature is not really required for the low-bandwidth and low-pps GSM signalling. Moreover, TCP is stream oriented and does not conserve message boundaries. As such, the IPA header has to serve as a boundary between messages in the stream. Moreover, assuming a generally quite idle signalling link, the assumption of a pure TCP ACK (without any data segment) is very likely to happen.
- Raw IP or UDP as alternative is not a real option, as it does not recover lost packets.
- SCTP preserves message boundaries and allows for multiple streams (multiplexing) within one connection, but it has too much overhead.

For that reason, we propose the use of LAPD for this task. This protocol was originally specified to be used on top of E1 links for the A interface, who do not expose any kind of noticeable latency. LAPD resolves (albeit not as good as TCP does) packet loss and copes with packet re-ordering.

LAPD has a very small header (3-5 octets) compared to TCPs 20 bytes. Even if LAPD is put inside UDP, the combination of 11 to 13 octets still saves a noticeable number of bytes per packet. Moreover, LAPD has been modified for less reliable interfaces such as the GSM Um interface (LAPDm), as well as for the use in satellite systems (LAPsat in ETSI GMR).

OSmux protocol

The OSmux protocol is the core of our proposed solution. This protocol operates over UDP or, alternatively, over raw IP. The designated default UDP port number and IP protocol type have not been yet decided.

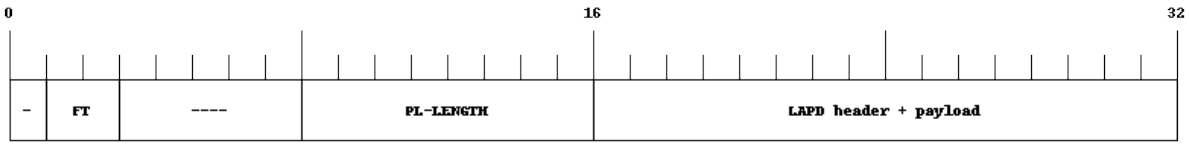
Every OSmux message starts with a control octet. The control octet contains a 2-bit Field Type (FT) and its location starts on the 2nd bit for backward compatibility with older versions (used to be 3 bits). The FT defines the structure of the remaining header as well as the payload.

The following FT values are assigned:

- FT == 0: LAPD Signalling
- FT == 1: AMR Codec
- FT == 2: Dummy
- FT == 3: Reserved for Fture Use

There can be any number of OSmux messages batched up in one underlying packet. In this case, the multiple OSmux messages are simply concatenated, i.e. the OSmux header control octet directly follows the last octet of the payload of the previous OSmux message.

LAPD Signalling (0)

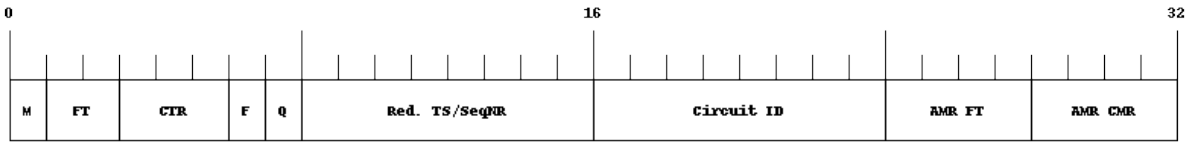


Field Type (FT): 2 bits
The Field Type allocated for LAPD Signalling frames is "0".

This frame type is not yet supported inside OsmoCom and may be subject to change in future versions of the protocol.

AMR Codec (1)

This OSmux packet header is used to transport one or more RTP-AMR packets for a specific RTP stream identified by the Circuit ID field.



Marker (M): 1 bit
This is a 1:1 mapping from the RTP Marker (M) bit as specified in RFC3550 Section 5.1 (RTP) as well as RFC3267 Section 4.1 (RTP-AMR). In AMR, the Marker is used to indicate the beginning of a talk-spurt, i.e. the end of a silence period. In case more than one AMR frame from the specific stream is batched into this OSmux header, it is guaranteed that the first AMR frame is the first in the talkspurt.

Field Type (FT): 2 bits
The Field Type allocated for AMR Codec frames is "1".

Frame Counter (CTR): 2 bits
Provides the number of batched AMR payloads (starting 0) after the header. For instance, if there are 2 AMR payloads batched, CTR will be "1".

AMR-F (F): 1 bit
This is a 1:1 mapping from the AMR F field in RFC3267 Section 4.3.2. In case there are multiple AMR codec frames with different F bit batched together, we only use the last F and ignore any previous F.

AMR-Q (Q): 1 bit
This is a 1:1 mapping from the AMR Q field (Frame quality indicator) in RFC3267 Section 4.3.2. In case there are multiple AMR codec frames with different Q bit batched together, we only use the last Q and ignore any previous Q.

Circuit ID Code (CIC): 8 bits
Identifies the Circuit (Voice call), which in RTP is identified by {srcip, srcport, dstip, dstport, ssrc}.

Reduced/Combined Timestamp and Sequence Number (RCTS): 8 bits
Resembles a combination of the RTP timestamp and sequence number. In the GSM system, speech codec frames are generated at a rate of 20ms. Thus, there is no need to have independent timestamp and sequence numbers (related to a 8kHz clock) as specified in AMR-RTP.

AMR Codec Mode Request (AMR-FT): 4 bits
This is a mapping from the AMR FT field (Frame type index) in RFC3267 Section 4.3.2. The length of each codec frame needs to be determined from this field. It is thus guaranteed that all frames for a specific stream in an OSmux batch are of the same AMR type.

AMR Codec Mode Request (AMR-CMR): 4 bits
The RTP AMR payload header as specified in RFC3267 contains a 4-bit CMR field. Rather than transporting it in a separate octet, we squeeze it in the lower four bits of the clast octet. In case there are multiple AMR codec frames with different CMR, we only use the last CMR and ignore any previous CMR.

Additional considerations

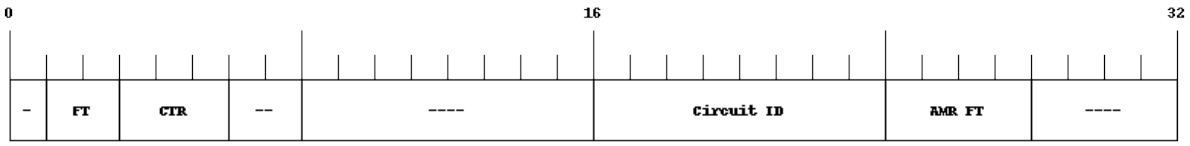
- It can be assumed that all OSmux frames of type AMR Codec contain at least 1 AMR frame.
- Given a batch factor of N frames (N>1), it can not be assumed that the amount of AMR frames in any OSmux frame will always be N, due to some restrictions mentioned above. For instance, a sender can decide to send before queueing the expected N frames due to timing issues, or to conform with the restriction that the first AMR frame in the batch must be the first in the talkspurt (Marker M bit).

Dummy (2)

This kind of frame is used for NAT traversal. If a peer is behind a NAT, its source port specified in SDP will be a private port not accessible from the outside. Before other peers are able to send any packet to it, they require the mapping between the private and the public port to be set by the firewall, otherwise the firewall will most probably drop the incoming messages or send it to a wrong destination. The firewall in most cases won't create a mapping until the peer behind the NAT sends a packet to the peer residing outside.

In this scenario, if the peer behind the nat is expecting to receive but never transmit audio, no packets will ever reach him. To solve this, the peer sends dummy packets to let the firewall create the port mapping. When the other peers receive this dummy packet, they can infer the relation between the original private port and the public port and start sending packets to it.

When opening a connection, the peer is expected to send dummy packets until it starts sending real audio, at which point dummy packets are not needed anymore.



Field Type (FT): 2 bits
The Field Type allocated for Dummy frames is "2".

Frame Counter (CTR): 2 bits
Provides the number of dummy batched AMR payloads (starting 0) after the header. For instance, if there are 2 AMR payloads batched, CTR will be "1".

Circuit ID Code (CIC): 8 bits
Identifies the Circuit (Voice call), which in RTP is identified by {srcip, srcport, dstip, dstport, ssrc}.

AMR Codec Mode Request (AMR-FT): 4 bits
This field must contain any valid value described in the AMR FT field (Frame type index) in RFC3267 Section 4.3.2.

Additional considerations

- After the header, additional padding needs to be allocated to conform with CTR and AMR FT fields. For instance, if CTR is 0 and AMR FT is AMR 6.9, a padding of 17 bytes is to be allocated after the header.
- On receival of this kind of OSmux frame, it's usually enough for the reader to discard the header plus the calculated padding and keep operating.

Sequence Charts

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.

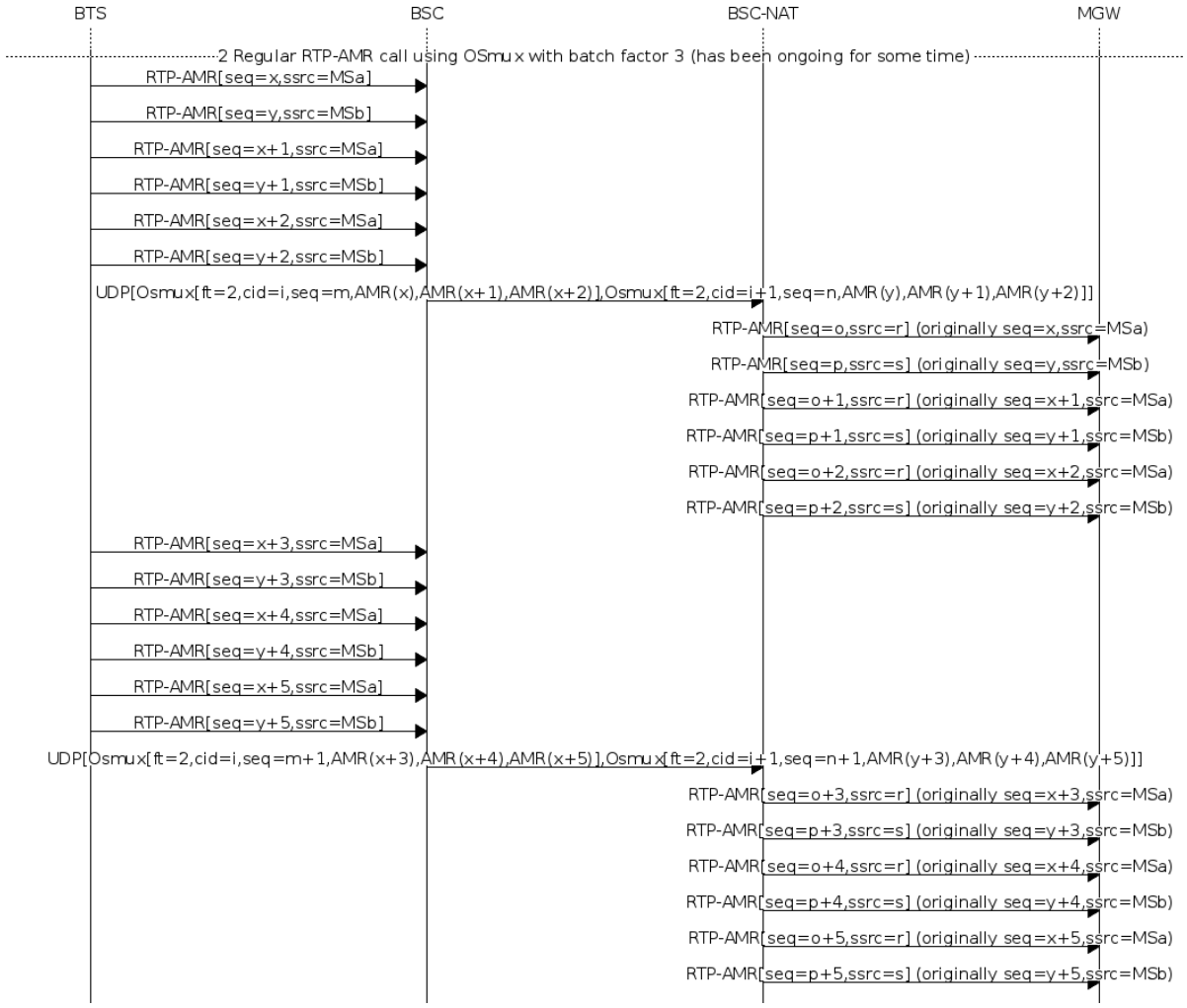
Batching

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).

Trunking and Batching

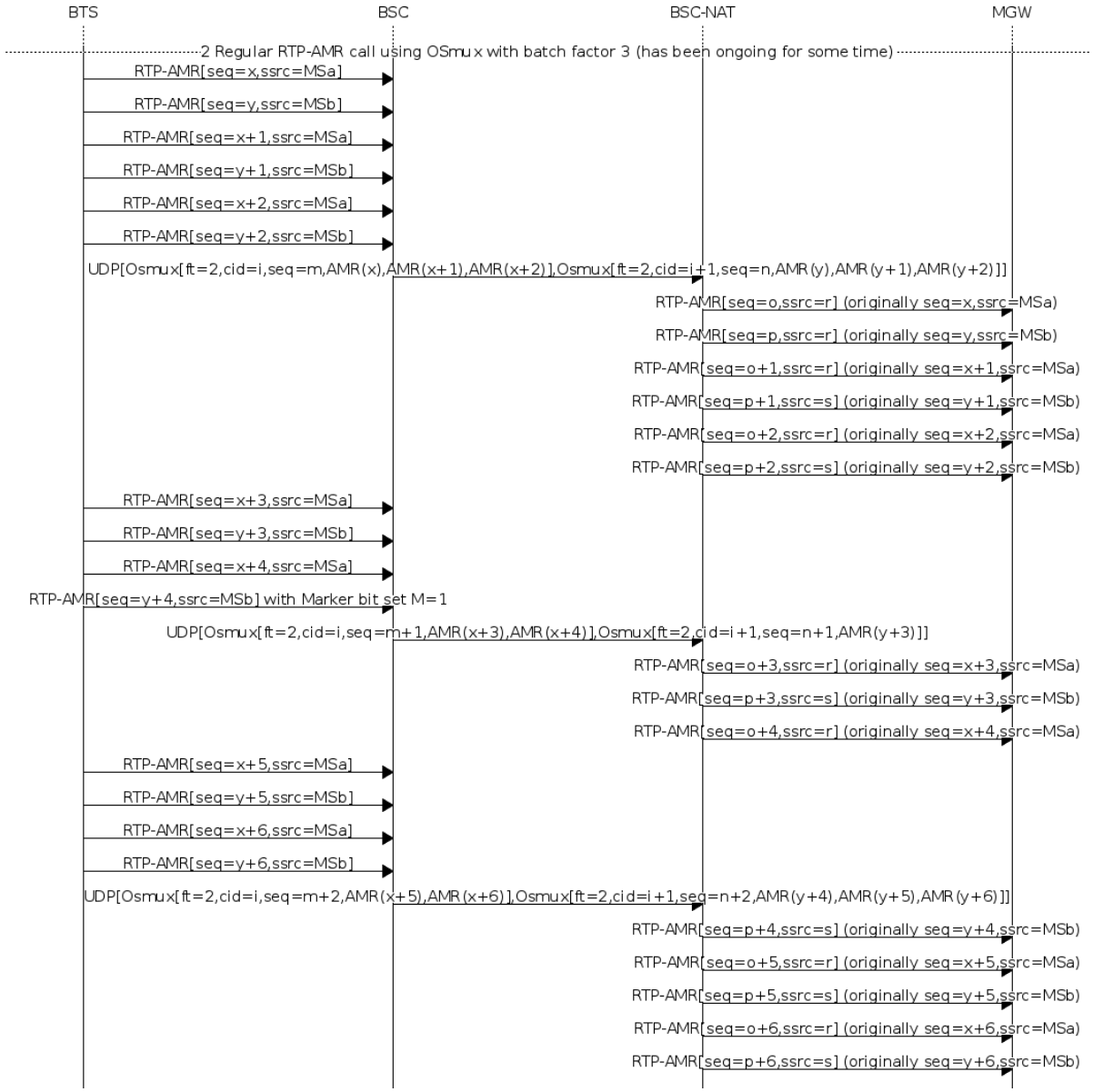
Following chart shows how trunking and batching work together. The chart shows 2 concurrent calls from different MS on a given BTS, and BSC is configured with a batch factor of 3. Again only uplink data is shown, but downlink follows the same idea. Batching factor is set to 1 to easily illustrate trunking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



Marker bit

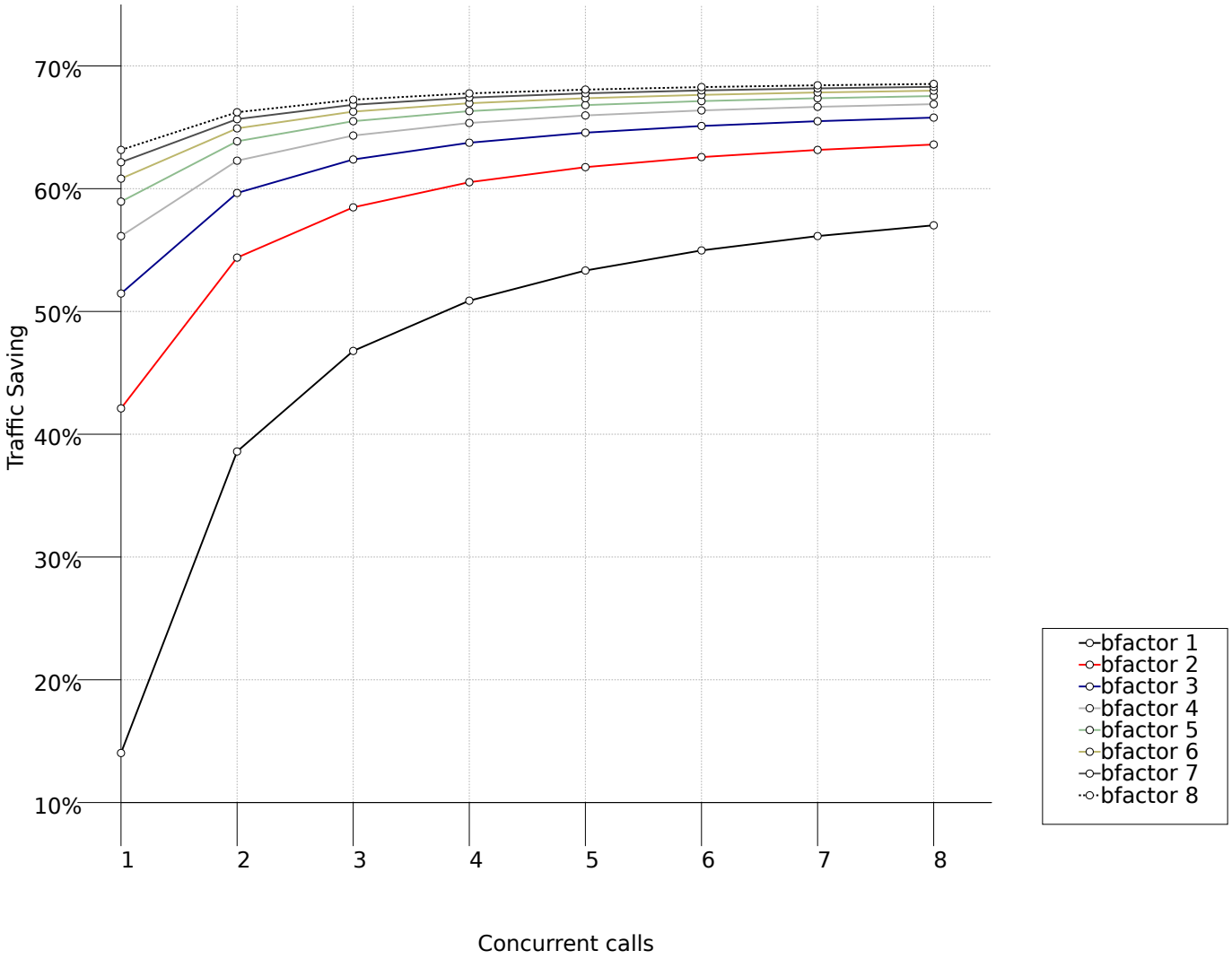
As described earlier, the Marker bit is always expected to relate to the first AMR payload of an OSmux frame. Thus, special considerations may be followed when the OSmux encoder receives an RTP packet with a marker bit. For instance, previously enqueued RTP packets may be sent even if the configured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

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



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.

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.

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 (20ms * 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).

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.

