Guard band signalling – Does it help?

Tony Jeffree

August 2013

MENU

- Starting assumptions
- Minimum fragment size
- Possible delays for preemptive frames and implications for “guard band”
- Scenarios using “hold-req” signalling
- Scenarios using no “hold-req” signalling
- Conclusions

Disclaimer

The precise details (frame sizes, IPG sizes, timings...etc.) in this presso may or may not be strictly accurate, and will in any case flex with the precise details of the solution selected by 802.3; however, it hopefully illustrates some of the principles and effects that are relevant to further understanding of how preemption can/will operate in different scenarios. Therefore, please focus on the message, not the nits.

Starting point for this presso

- The “possible architecture” for 802.3 preemption (as presented in the July tutorial)
- P802.1Qbv/D0.2
- P802.1Qbu/D0.0, with the additional change to Qbv that 8.6.8 (c) does not apply if preemption is supported (i.e., if the gate is open for a queue, it can transmit, regardless of whether the frame fits in the time available before the next gate close event).

Minimum fragment sizes

- The “possible architecture” for 802.3 preemption (presented by Pat in the July tutorial) states that:
  - Preemptable frame fragments cannot be <64 octets
  - A standard 802.3 IPG (96 bit times, or 12 octet times) always separate adjacent frames/frame fragments
  - Frame fragments cannot be “padded” to meet the 64 octet rule
- Therefore, a preemptable frame (or frame fragment) that is shorter than 128 octets in length cannot be further subdivided, because doing so would result in one fragment or both fragments being <64 octets long

Possible delays before transmitting a preemptive frame:

- 0 bit times (no delay) if the last preemptable frame finished more than 96 bit-times ago (i.e., >1 IPG ago)
- 96 bit-times (1 IPG) if:
  - there is a preemptable frame/fragment transmission in progress, and
  - >=64 octets of the frame/fragment have been transmitted, and
  - there are >=64 octets still to transmit
- 608 bit times (96 bit-times + up to 64 octets) if:
  - there is a preemptable frame/fragment transmission in progress, and
  - >=64 octets of the frame/fragment have been transmitted, and
  - there are <64 octets still to transmit
- 608 bit times (96 bit-times + up to 64 octets) if:
  - there is a preemptable frame/fragment transmission in progress, and
  - <64 octets of the frame/fragment have been transmitted, and
  - the total size of the remaining preemptable frame/fragment (already transmitted plus still to transmit) is >=128 octets
- 1176 bit times (96 bit-times + up to 127 octets) if:
  - there is a preemptable frame/fragment transmission in progress, and
  - the total size of the remaining preemptable frame/fragment (already transmitted plus still to transmit) is <=127 octets

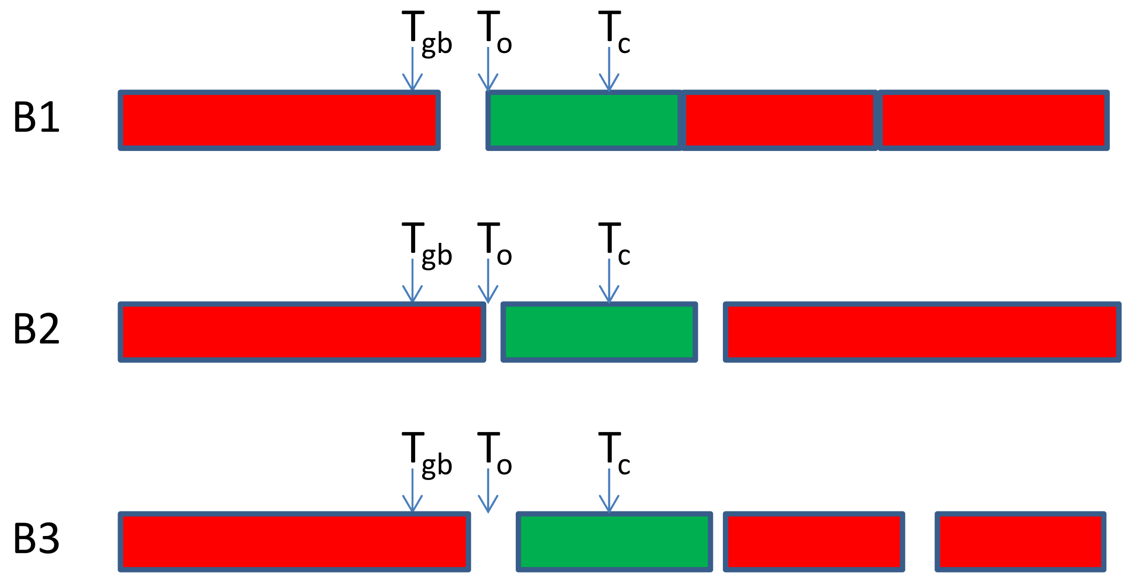
Implication for Guard Band

- If it is desired to provide preemptive traffic with a clear run within its TAS window, then the start of the Guard Band, i.e., the point at which MA_MM.request(hold_req) is issued to MAC Merge, needs to be >=1176 bit times before a Gate-open event for a preemptive traffic class, and the end of the Guard Band needs to be issued at the same time as the Gate-close event for that traffic class
- This means only the 13.7X reduction in guard band described in Norm’s tutorial preso is achievable, not the 24X (Sorry Norm)
- The 24X case would be achievable if it was possible to pad the last frame fragment out to 64 octets, or if the last fragment could be <64 octets
- Obviously, better than 24X reduction if frame fragments could all be smaller than 64 octets

Scenarios using hold_req signaling & guard band

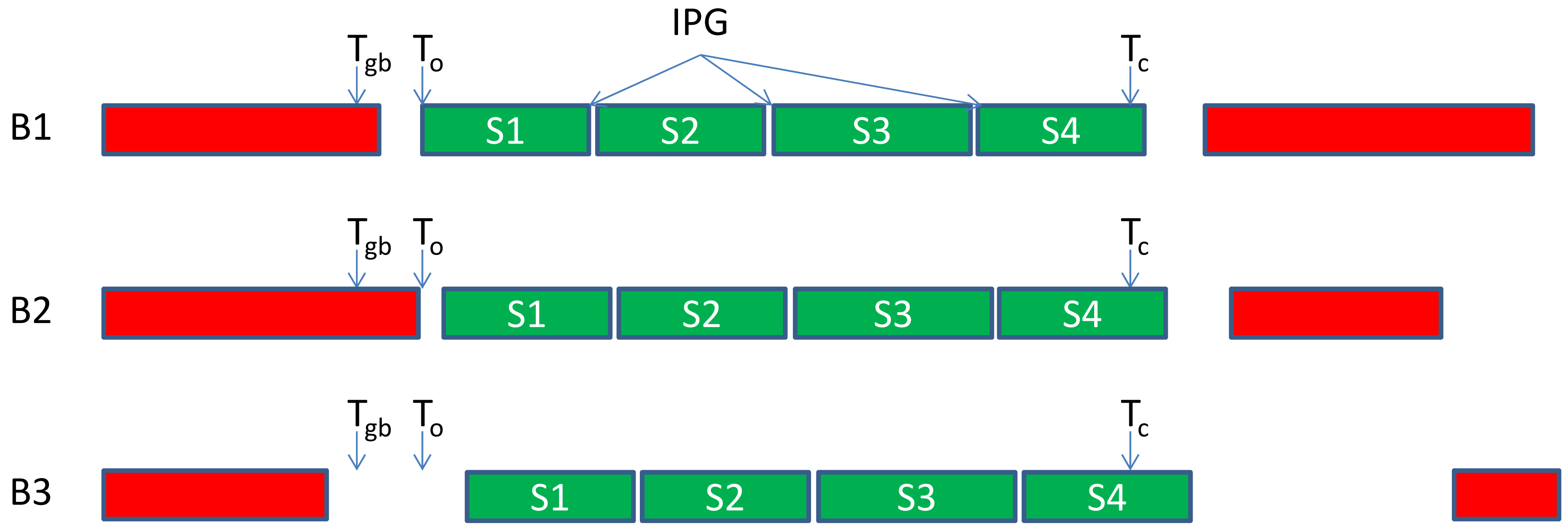
- hold_req is used in order to shorten the guard band
- how long is needed between Gate-open and Gate-close events for that traffic class?
- What is the impact on latency for preemptive frames?
- What is the effect on bandwidth available to best effort traffic?

Case 1: One frame per window



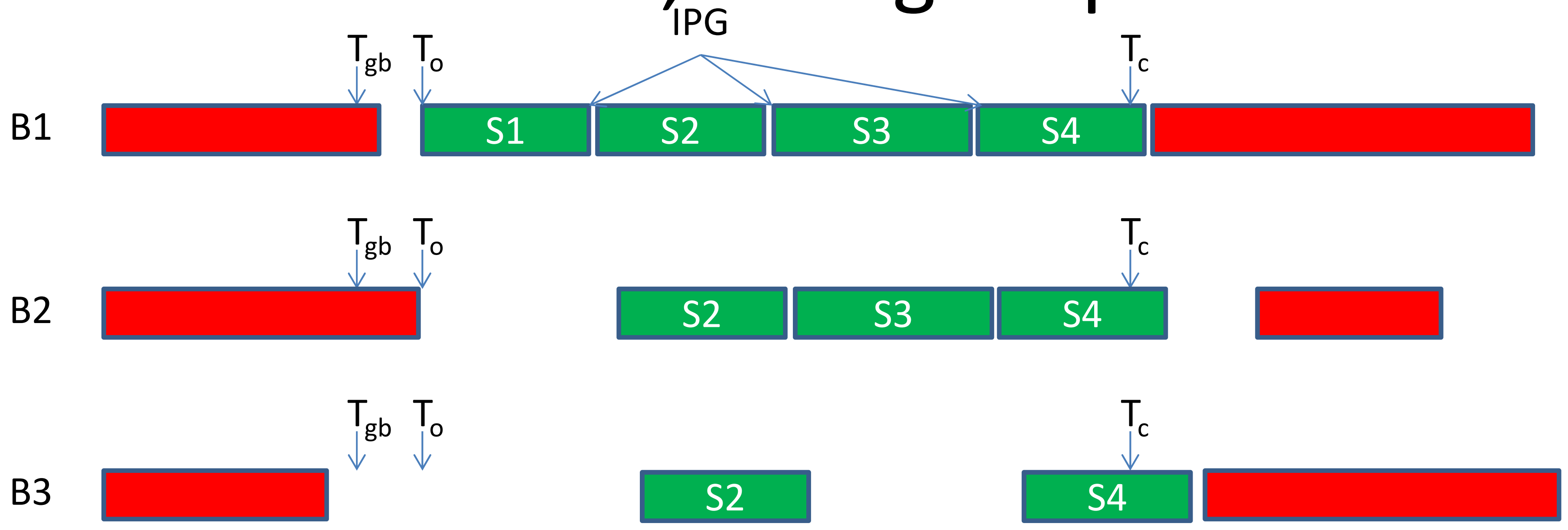
- In successive bridges, the arrival time of the preemptive (green) frames is delayed by the frame transit time (T_t, = LAN propagation delay + input queuing delay + store-and-forward delay – see 802.1Q Annex L.3)
- Latency is simply N x T_t, where N is the number of bridges transited
- T_c - T_o must be at least N x T_t but need not be much bigger
- Alternatively, the window can be moved later by T_t in each successive bridge

Case 2: More than one frame per window



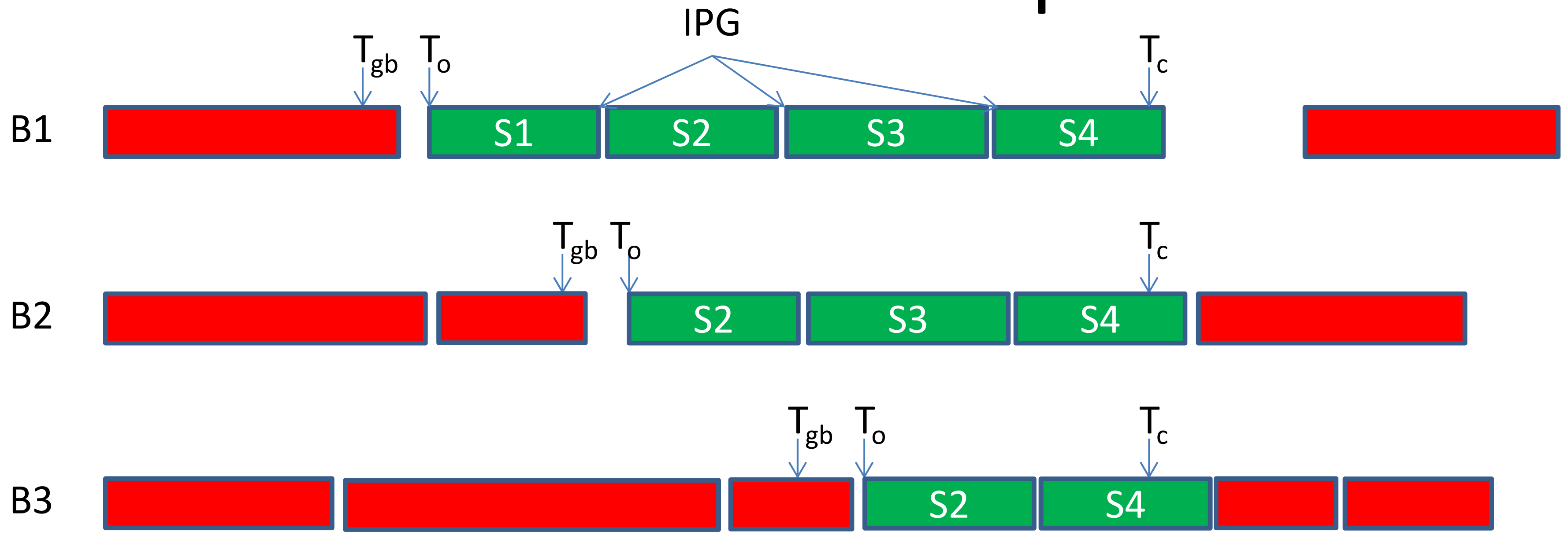
- In B1, frames S2 through S4 are delayed by 1 IPG, and if S1 through S4 arrived at 4 input ports of B1 around the same time, S2 through S4 are also delayed due to "Fan-in delay" (see 802.1Q Annex L.3)
- In successive bridges, the arrival time of the preemptive (green) frames is delayed by T_t, as before
- Latency is N x T_t plus fan-in delay, plus 1 IPG per frame.
- T_c - T_o must be at least N x T_t plus the sum of S1 through S3 and their IPGs
- If the order of the arrival times of S1 through S4 at B1 cannot be relied on, then the above calculation must be based on the lengths of the longest frames
- Again, the window can be moved later by T_t in each successive bridge

Case 3: More than one frame per window; divergent paths



- In B1, frames S2 through S4 are delayed by 1 IPG, and if S1 through S4 arrived at 4 input ports of B1 around the same time, S2 through S4 are also delayed due to "Fan-in delay" (see 802.1Q Annex L.3) in B1
- S1 is destined for a branch off the main path at B1; S3 branches off the main path at B2
- In successive bridges, the arrival time of the preemptive (green) frames is delayed by T_t, as before
- Latency is N x T_t plus fan-in delay, plus 1 IPG preceding per frame
- T_c - T_o STILL must be at least N x T_t plus the sum of S1 through S3 and their IPGs IN ALL BRIDGES ON THE COMMON PATH; there is no means of closing the gaps, short of delaying T_o by the lengths of the frames that branch off (which would have certain similarities to the peristaltic shaper).
- If the order of the arrival times of S1 through S4 at B1 cannot be relied on, then the above calculation must be based on the lengths of the longest frames
- Again, the window can be moved later by T_t in each successive bridge

Case 4: Same as Case 3, but with best-effort bandwidth optimised



- Delaying T_o in B2 and B3 makes more bandwidth available to best-effort traffic, at the expense of forcing worst case fan-in delay for the green frames.

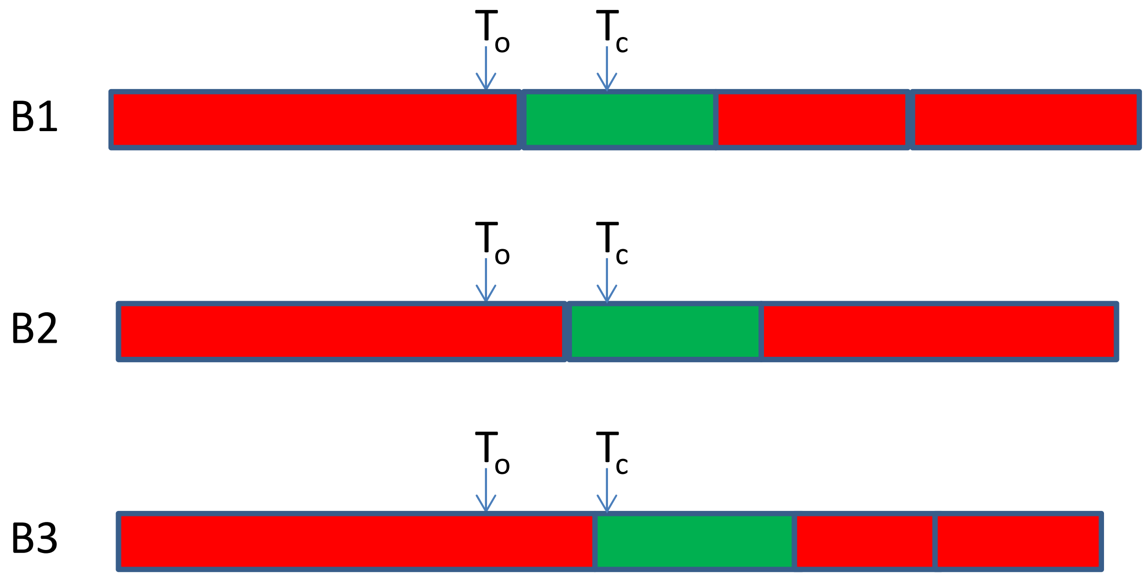
Conclusions

- A guard band of 1176 bit times, coupled with use of hold_req, allows preemptive traffic to be transmitted without any delays caused by best-effort interference. Use of hold_req and preemption gives a 13.7x improvement in guard band size relative to the guard band needed if there is no preemption
- It is no surprise that fan-in delays, caused by multiple streams using the same window, still have an effect
- If it is essential to minimise the worst case latency for a given stream, then minimise the number of streams that use each preemptive window. One frame per window reduces the latency hit to just the accumulated transit times; however, it also increases the hit on best-effort traffic (increases the proportion of guard band time)
- If it is essential to minimise impact on best effort bandwidth, use multiple streams per window (reduces the proportion of guard band time), and delay the Gate_open in successive bridges to force any gaps caused by fan-in to close up. However, this also forces worst-case fan-in delays to be imposed on the preemptive traffic
- In general, knowledge of the upstream configuration of the network is required in order to be able to configure the gate events in a given bridge

Scenarios using no guard band

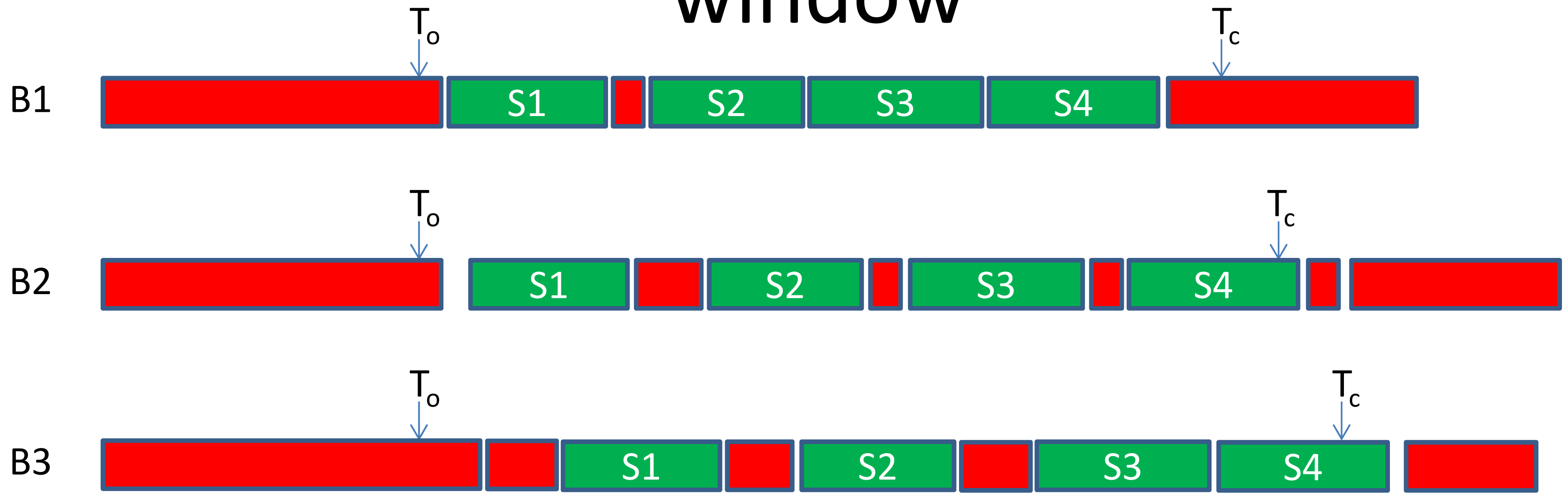
- hold_req is not used, but preemption is used.
- how long is needed between Gate-open and Gate-close events for that traffic class?
- What is the impact on latency for preemptive frames?
- What is the effect on bandwidth available to best effort traffic?

Case 1a: One frame per window



- In successive bridges, the arrival time of the preemptive (green) frames is delayed by T_t, plus up to 1176 bit times.
- Max latency is simply N x (T_t + 1176) bit times, where N is the number of bridges transited
- T_c - T_o must be at least N x (T_t + 1176) but need not be much bigger
- Alternatively, the window can be moved later by T_t in each successive bridge, in which case T_c - T_o must be at least N x 1176 bit times

Case 2a: More than one frame per window

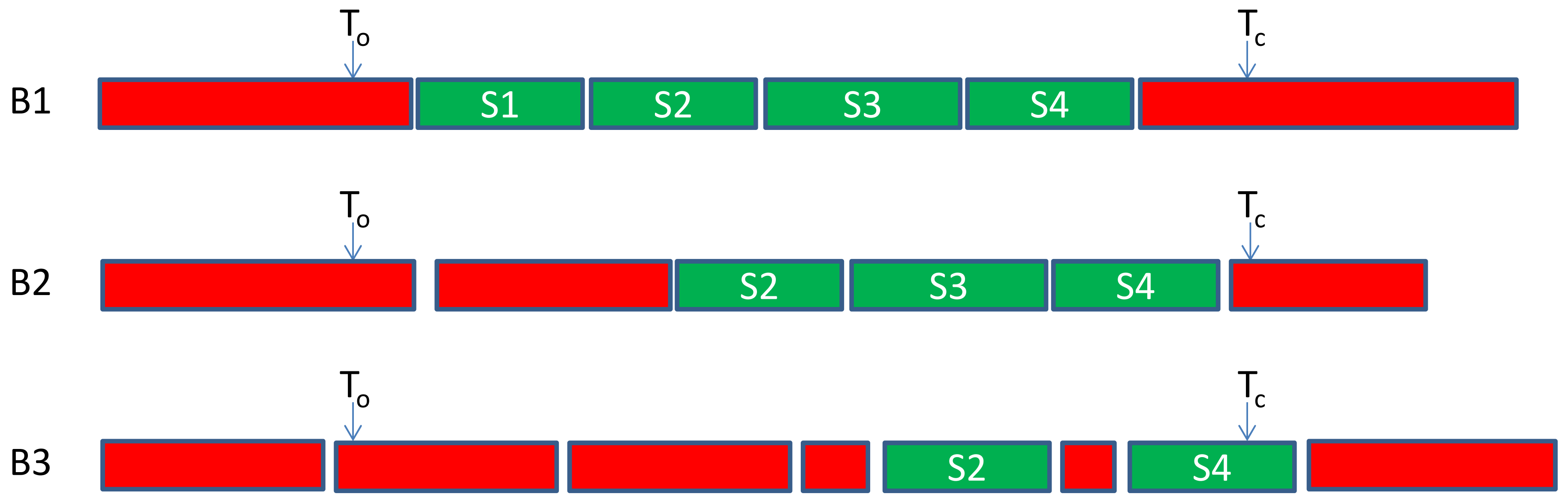


• In addition to Case 2, S1 through S4 can be delayed by frame fragments; if the timings are right (bad), a fragment can get inserted between each pair of green frames

•In successive bridges, the gaps between preemptive (green) frames can potentially “grow”, if a preempted frame gets to re-start in the gaps between green frames. However, there is a natural limit to this growth as the sum of the fragments cannot exceed the max frame size (plus headers etc). So the worst case latency in this example is as for Case 2, with the addition of a fragmented max size frame.

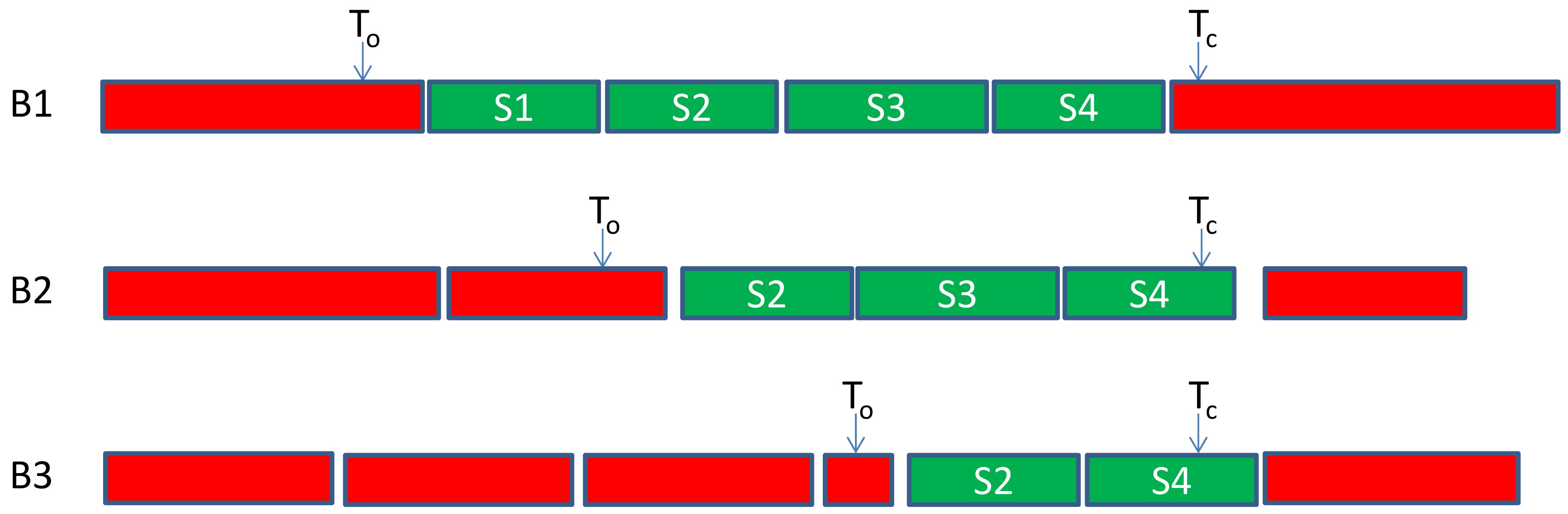
•A strategy of deliberately delaying the start of the window in successive bridges by 1175 bit times can be used to mimimize this effect (similar to Case 4), which can reduce the hit to Case 2 plus 1175 bit times per hop (a worst case fragment interposed between the start of the window and the first green frame).

Case 3a: More than one frame per window; divergent paths



- In B1, frames S2 through S4 are delayed by 1 IPG, and if S1 through S4 arrived at 4 input ports of B1 around the same time, S2 through S4 are also delayed due to "Fan-in delay" (see 802.1Q Annex L.3) in B1.
- The first "green" frame after T_o is also potentially delayed by up to 1175 bit times AT EACH HOP
- Any "gaps" caused by fan out can be extended in size at each hop by up to 1175 bit times

Case 4a: Same as Case 3a, but with latency hit improved



- By delaying T_o in successive bridges, the opportunity for the "gaps" to grow is removed; however, the start of the first green frame can still be delayed by up to 117 bit times in each successive bridge, so the latency "hit" relative to case 4 is reduced to 1175 bit times per hop (a worst case fragment interposed between the start of the window and the first green frame).

Conclusions

- Losing the explicit start-of-guard-band signal can result in up to a frame’s worth of fragments being inserted between preemptive frames in the worst case. However, if careful attention is paid to the timing of the start of the preemptive window in each bridge along the path, the additional latency (relative to the case where the start-of-guard-band signal is available) can be mitigated, reducing it to 1176 bit times per hop.
- The most obvious advantage of losing the explicit start-of-guard-band signal is elimination of the guard band, which maximises the bandwidth available to preemptable traffic. In effect, the guard band has been moved inside the “green” window, and that bandwidth is available either to red or green frames.

Conclusions from the Conclusions

- Preemption without the addition of guard band signalling carries a latency penalty (relative to preemption + guard band signalling) for preemptive traffic, but this can be kept small by appropriate choice of gate timings, and it also has the benefit of maximizing the available bandwidth for preemptable traffic
- Preemption with the addition of guard band signalling carries no latency penalty for preemptive traffic, and the bandwidth "hit" for preemptable traffic is 13.7x better than with no preemption at all (where the guard band = max frame size). If there is significant "fan out", then there is the potential for wasted bandwidth that could be used by preemptable traffic; this could be mitigated by delaying the start of the green window to account for fan out.