Blocking of misbehaving streams

Daniel Zebralla
daniel.zebralla@continental-corporation.com

IEEE 802.1 Interim, May 2016, Budapest

Acknowledgements

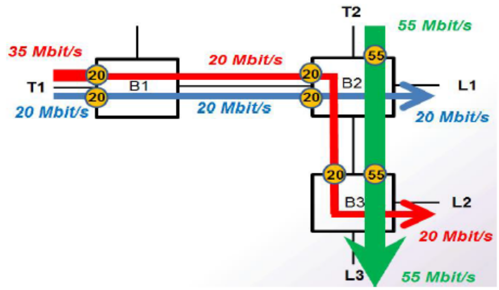
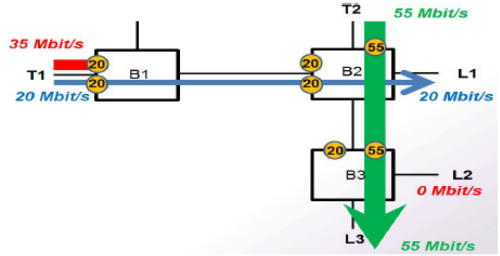
Thanks to the following individuals who contributed to this presentation

- Jens Bierschenk
- Christian Boiger
- Soheil Samii
- Johannes Specht
- Helge Zinner

Motivation

Automotive emphasized the need for complete stream blocking earlier

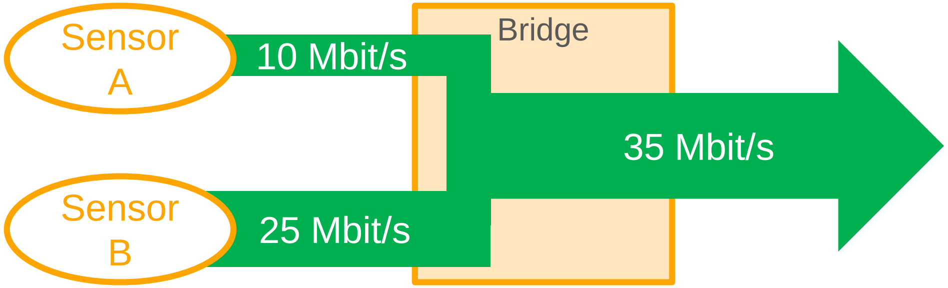
From http://www.ieee802.org/1/files/public/docs2013/new-tsn-jochim-ingress-policing-1113-v1.pdf
(“Per Class”-cases omitted):

	Per Stream *(= Potentially higher number of filters per port)*
Threshold Enforcing	 T2 55 Mbit/s 35 Mbit/s 20 Mbit/s 55 20 20 B1 20 20 B2 L1 T1 20 Mbit/s 20 Mbit/s 20 Mbit/s 20 55 B3 L2 20 Mbit/s L3 55 Mbit/s - A faulty stream sent by a faulty talker is not “silenced”. - Other streams from faulty / fault free talkers not affected.
Blocking	 T2 55 Mbit/s 35 Mbit/s 55 20 20 T1 B1 B2 L1 20 20 20 Mbit/s 20 Mbit/s 20 55 B3 L2 0 Mbit/s L3 55 Mbit/s - A faulty stream sent by faulty talker is “silenced”. - Non-faulty streams sent by faulty talker are not necessarily silenced.

	Per Stream *(= Potentially higher number of filters per port)*
Threshold Enforcing	**Not a preferred solution.** Reason: “*Per Stream X Blocking*” and “*Per Stream X Thr. Enforcing*” both require the same HW overhead associated with Per Stream IPFs, but “*Per Stream X Blocking*” exhibits preferable behavior.
Blocking	**#1 Preferred solution !!!** Very effectively addresses the Babbling Idiot problem.

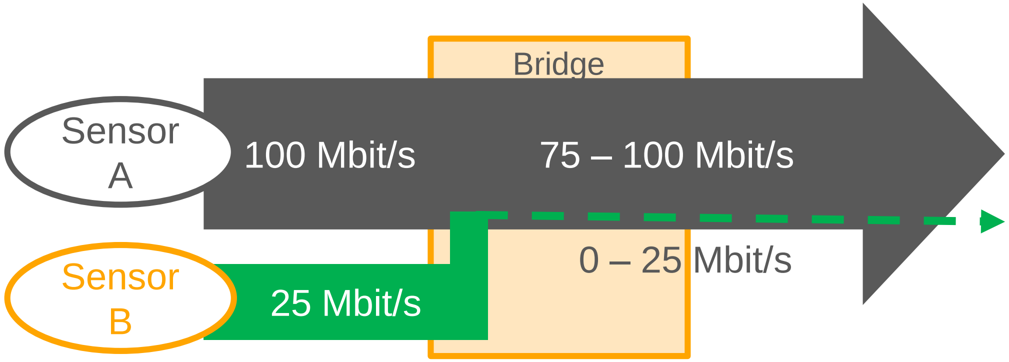
Motivation

Automotive: Babbling sensor



Normal operation

- Assuming 100 Mbit/s, everything’s fine

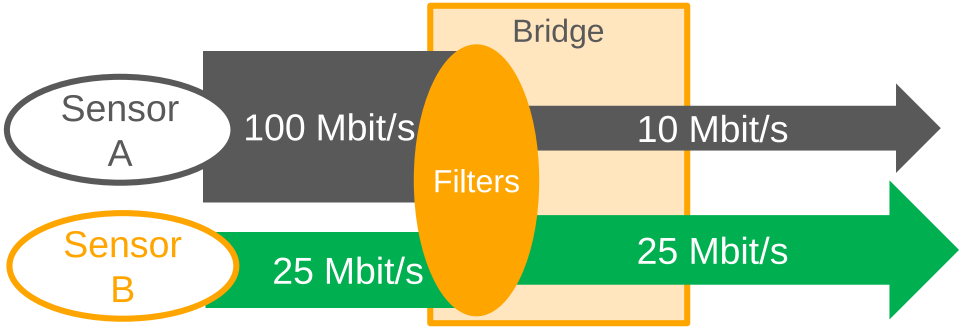


Babbling case

- Sensor A is stuck at sending
- Sensor A sends junk (probably no sensible data, thus grey color)

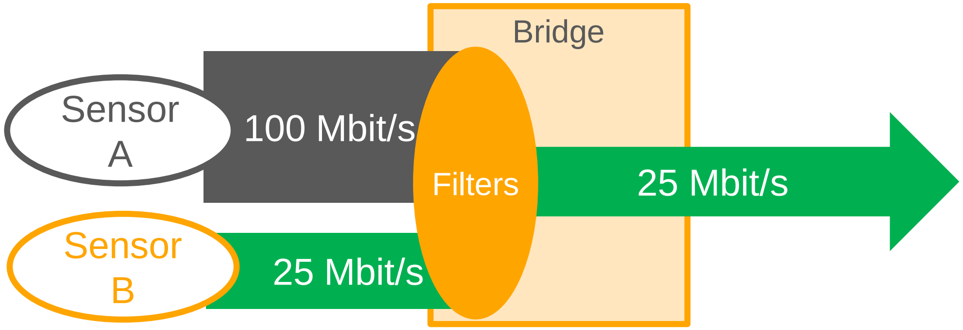
Motivation

Automotive: Babbling sensor



Mitigation using Qci D1.1

- A’s bandwidth is limited
- Still, 10 Mbit/s of *probably* junk or invalid data is sent
- This could lead to further errors down the line (cf. fault → error → failure propagation)



Blocking proposal

- A’s stream is blocked at the bridge
- Bridges/ECUs down the line are relieved from handling the *invalid* data (the failure is contained)
- The overall system can transition into a defined state, “limp home”-mode

Motivation

Automotive sensor networks

Stream blocking is **necessary** because …

- … autonomous driving functions will require defined states in the network at all times
- … lots of raw sensor data will get merged from a multitude of sensors
- … erroneous input may cause a drop in environmental model quality

Motivation

From the PAR

“Policing and filtering functions include the detection and mitigation of disruptive transmissions by other systems in a network, improving the robustness of that network.”

→ From my interpretation, “mitigation” also means having a possibility to silence the disruptive transmission

Comments against D1.1 regarding stream blocking

#8 (Mick Seaman)

Re: Editors Note. Keep it simple and do not add the envisaged actions. In particular automatically closing down ports would have to be approached exceedingly cautiously. I don't see anything in the PAR that extends scope that far.

Suggested Remedy:

Remove Editor's Note and go with what we have.

#28 (Christian Boiger)

It should be possible to drop streams or close a port based on detected misbehavior of the link partner.

Suggested Remedy:

Add an additional mechanism that can force the gate status to closed. This allows to configure either that a stream, several streams, or all traffic is dropped. It should be possible trigger the "force gate closed" by all defined mechanisms that detect misbehavior (e.g. max SDU, RED, blocked by stream gate). The gate instances associated with a "force gate closed" event, triggered by a particular detected misbehavior of a link partner, should be configurable.

Comments against D1.1 regarding stream blocking

#35 (Helge Zinner)

I would like to see the possibity to completely block misbehaving streams once an error (or a defined number of errors in a certain time frame) have occurred.

Suggested Remedy:

Add additional mitigation options (in adition to drop option):
1) block stream_identifier (until reset, until timeout, until new reservation,....)
2) close a port
3) ...(more) ?
add-on to e): ...Frames that fail a filter are discarded.... and their corresponding stream_identifier specifications may be added to a "block table"
Options:
1) if the filter is violated once then the stream_identifier specification shall be added to the "block table" (automotive use case: a diagnosis port shall never be used while driving – if packets were received from this port an attack may be in progress)
2) if the Maximum SDU size/Flow meter instance identifier is violated n times in m seconds then the stream_identifier specification shall be added to the block table
3) ...(more)?
add a new filter specification under item e)
4)
block table. Frames containing stream_identifier specifications inside this table will be dropped. Entries into this table can be added dynamically (see e) or static by configuration.

Comments against D1.1 regarding stream blocking

#40 (Soheil Samii)

One mitigation option that has been discussed in the TSN group in the past is the requirement to allow multiple (statically) configurable mitigation options (i.e., what shall the bridge/endstation do in case a stream violates its timing and/or bandwidth contract). In the list of options, we should include "blocking the stream" and "blocking the (ingress) port," as these are mechanisms to isolate faults, for example in safety-critical automotive systems.

Suggested Remedy:

Add blocking table that can be modified at runtime based on what is detected by the stream filters/input gates. The blocking table can have the same parameters as what is used to identify streams. In case a stream violates its timing or bandwidth contract (or any other property we decide to filter on, e.g., maximum payload size), an entry is added to the blocking table to prevent all future frames of that stream to be forwarded.

Approach with a “blocking table”

stream_identifier specification*
127
22
14
17
…
*
<null>
…
<null>

Please wait for the next slide to see some assumptions

*) The simple numbers are just to indicate different identifiers

Approach with a “blocking table”

Assumptions

- One table per port
- The tables are empty on startup/reset
- Table is checked from top to bottom with “first match wins”
  - This needs to be fast, maybe a smarter implementation would be needed
- Complete closing of a port can be achieved with adding a wildcard as stream_identifier specification
- Table entries can be added/modified/removed via configuration as well
  - e.g. based on time

Discussion

- Is this function reasonably achievable (complexity)? Opinion of semiconductors?
  - To reduce complexity, wildcards in the blocking table could be forbidden (an individual stream_identifier could then only exist at most once per table). Complete port blocking would then need to be realized differently.
- What’s the minimum length (rows) which a “blocking table” must support? E.g. total # of streams in the network. This is low for automotive, but could get high for professional A/V and maybe industry automation.
  - Suggestion: Let this be determined via profiles, e.g. by AVnu
- Is the possibility to block streams of interest for industry automation people as well?
- Is unblocking of a stream/port after a certain time of interest for others as well?
- Where should the check if a stream is blocked happen? Compare figure 8-12.
- Alternative approaches for stream/port blocking besides “blocking table”?
  - See for example Christian’s comment #28

Thank you

for your attention!