Detect deadline miss of Path (composed of multiple nodes) #3176

nabetetsu started this conversation in **Design**

nabetetsu on Jan 6, 2023 edited

Summary

Through this discussion, I'd like to introduce a new timing violation monitoring feature to the Localization of Autoware.universe. Timing violation means the event when response time is larger than expected.

After adding this feature, Autoware will be able to detect one of the causes of localization failure. Then, Autoware will stop the vehicle when the risk is detected before it causes localization failure. Even when localization failure happens, log outputted from timing violation monitor will help the operator or developer to investigate the cause.

Adding the new feature will include the following changes.

- Adding new message type MessageTrackingTag
- Adding new publisher of the new message to ndt_scan_matcher/ndt_scan_matcher_core.cpp
- Adding new node to monitor timing violation

In this post, I'd like explain the policy of these changes and make agreement of the policy. Details of these changes will be explained via Pull Request.

Background

Currently, there is no common mechanism for detecting timing violation. Autoware is a real-time system, which means that the software shall save time constraints and use more fresh data. If time constraints cannot be saved, any function failed, and it may be difficult to continue the service.

For example, Localization is one of the critical features of Autoware, and it has time constraints.

Localization consists of multiple features such as point cloud filter nodes, NDT Scan Matcher, and EKF Localizer. The sequence of processes consisting of these must observe time constraints. If the time constraints are not observed, there is a concern that self-location estimation will fail.

In this post, I propose a feature to monitor the time constraints.

Category

Design

Labels

type:new-feature

4 participants

Expected achievement by this proposal

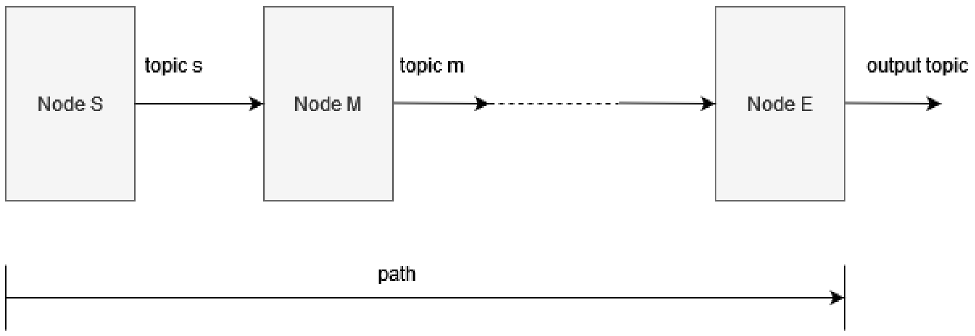
- Detection of timing violations in existing features which has time constraints
- Detection of timing violations caused by hardware, e.g., thermal throttling
- Stable timing violation detection by external monitoring

Timing violation monitor

Design Policy

Autoware consists of multiple nodes. Some data is passed to subsequent nodes using inter process communication with topic messages, and nodes process the data in order.

As shown in the figure below, the chain of nodes from the starting node (Node S) to the end node (Node E) is defined as path. The timing violation monitor verifies that the time constraints of the path are saved.



The timing violation monitor proposed here is intended to be deployed as a small start, and is designed with the following three points.

- Minimizing Autoware user code changes as much as possible
- Having the flexibility to add or delete paths to be monitored
- Running in a different Linux process from the monitored entity in order to keep the monitored entity stable
    - To prevent delays in the execution of the entity to be monitored by monitoring

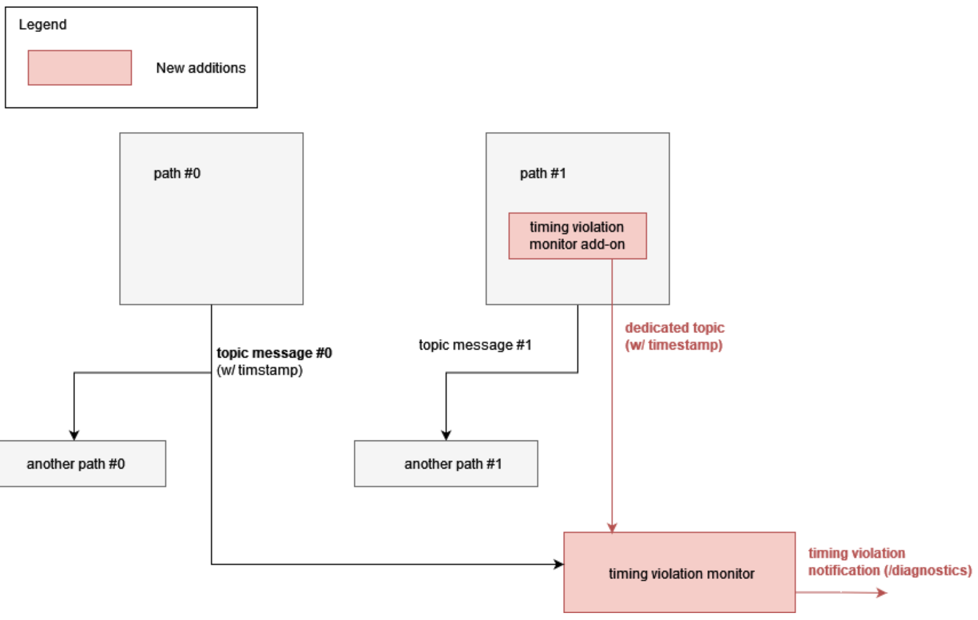
As described in more detail below, timing violation monitor refers to the header timestamp of existing topic messages as much as possible in order to avoid modifying existing Autoware user code.

On the other hand, depending on the implementation of the endpoint of the path, there is a possibility that the header timestamp cannot be referenced. In this case, a dedicated message is used instead of the header timestamp of existing topic message.

Design overview

As shown in the diagram below, the following two will be added by timing violation monitor.

- timing violation monitor node
- dedicated topic message
    - add-on function to output the topic message


Legend
New additions
path #0
path #1
timing violation monitor add-on
dedicated topic (w/ timestamp)
topic message #0 (w/ timstamp)
topic message #1
another path #0
another path #1
timing violation monitor
timing violation notification (/diagnostics)

The timing violation monitor subscribes the following two types of topic messages

- Existing topic messages with timestamp sent from the end node of the Path
- Dedicated topic messages which pass timestamp from starting node of the Path

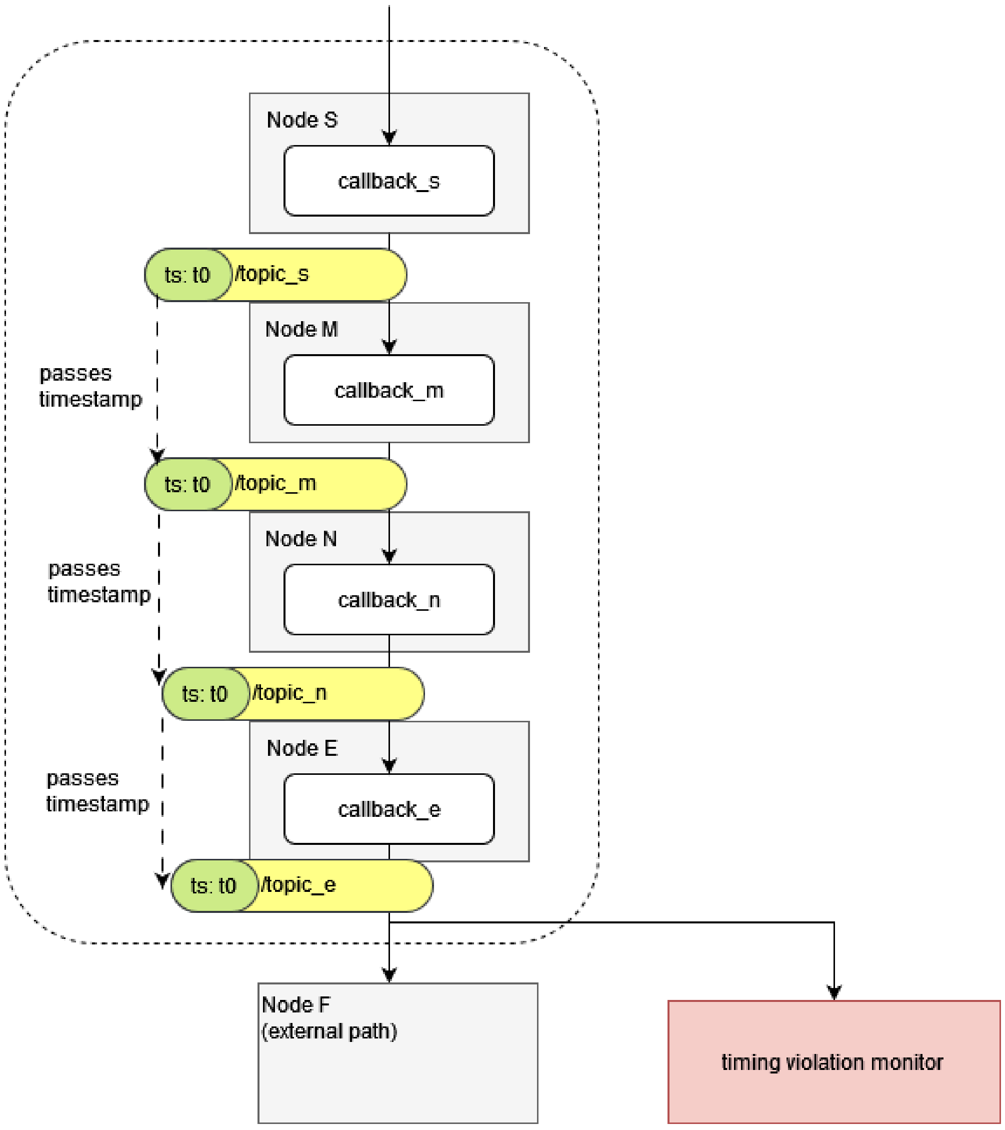
The timing violation monitor checks violation occurrence based on the timestamps in these messages. The monitor notifies timing violation occurrence to the upper-level monitor tools, for example, the `/diagnostics` topic.

Path #0 in the above diagram shows the case scenario where timing violation detection is accomplished by referencing an existing topic message, and Path #1 shows the case scenario where timing violation detection is accomplished by using a dedicated topic message.

Precondition on timestamp

In order to achieve this functionality, timestamps included in topic messages sent from the path are important.

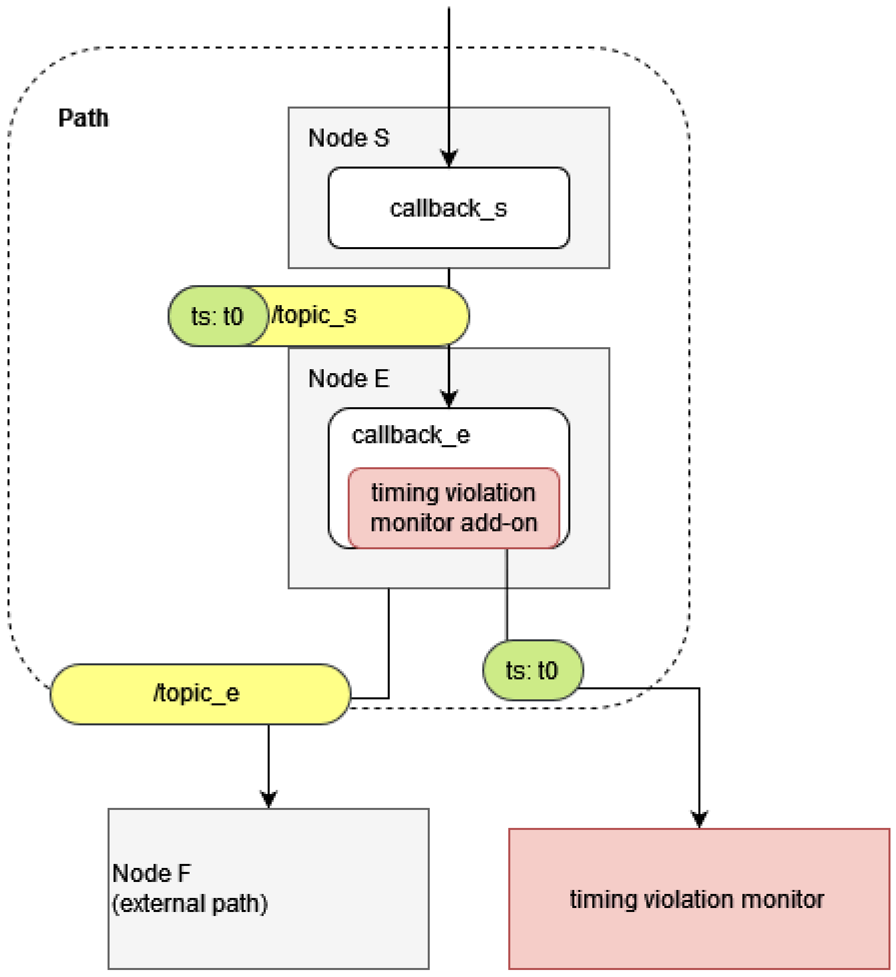
When timing violation monitor refers to an existing topic message, it is assumed that the timestamp given by the starting node of the path is pass to the end node, as shown in the following diagram.



In the diagram shown above, the timestamp sent from Node S is outputted from Node E. In Autoware, the PointCloud topics of the Sensing, Perception, and Localization nodes are transmitted in this way.

In this case, the timing violation monitor can refer to the header timestamp to know if the path save the time constraints without any change in the user code.

On the other hand, there are some cases where the header timestamp given at the starting node is not output from the end node of the path. In such cases, a dedicated message is output as shown in the diagram below.



In the diagram shown above, the timestamp output from Node S is not output from Node E, so Add-on outputs the dedicated message with the timestamp output from Node S.

If the data size of `/topic_e` is too large to be received by the timing violation monitor, the data transfer overhead can be reduced by using a dedicated message.

FYI) Comparison with other designs

I think that there are two alternative designs for detecting timing violation as below, but I'd like to recommend the proposed timing violation monitor.

- Design 1: To introduce a timing violation detection function at an end node of a path
- Design 2: To extend topic_state_monitor to support timing violation detection

The following subsection explains why I don't recommend them.

The reason why I don't adopt the design 1

At present, I am only trying to detect timing violations in localization. Someone may suggest that I introduce a timing violation detection function at an end node of a path.

However, I don't adopt the design 1 because of the following reasons.

- It will make large changes to the user code to add timing violation detection logic
- It will disperses the similar logic to multiple code, which degrades maintainability, when monitoring timing violations on multiple paths
- It will not let us separate the monitor from the end node if I'd like to execute the monitor on reliable platform

The reason why I don't adopt the design 2

My proposed timing violation monitor uses the timestamp in the header of topic messages. Its implementation may be similar to the that of existing `topic_state_monitor`. Some might suggest that I extend `topic_state_monitor` to detect timing violations.

However, I don't adopt the design 2 because of the following reasons.

- It will add new other responsibility to `topic_state_monitor` because it is only responsible for checking transmission of topic messages themselves rather than timing constraints of the paths
- It will degrade the cohesion of `topic_state_monitor` if `topic_state_monitor` has a function to detect timing violations
- It will make the configuration of `topic_state_monitor` more difficult and degraded its usability if `topic_state_monitor` has a function to detect timing violation

The implementation of timing violation monitor may be similar to the that of existing `topic_state_monitor`, but both should be implemented separately because they are based on respective unique requirements.

↑ 2 👍 5

4 comments · 11 replies

Oldest Newest Top

nabetetsu on Feb 3, 2023 Author

@mitsudome-r @kenji-miyake @takam5f2
Significant changes have been made to the submission in order to reach agreement on the design policy for the timing violation monitor. Thank you for your patience. I hope you will take the time to review it.

↑ 1 5 replies

kenji-miyake on Feb 5, 2023

@nabetetsu Thank you for your proposal.
I think @xmfcx -san is working on a similar task here:
https://github.com/orgs/autowarefoundation/discussions/3194
What's the comparison between his idea?

Also, I'm not sure what is MessageTrackingTag or etc. yet. So I'm looking forward to your proposal PR to understand more details.

nabetetsu on Feb 6, 2023 Author

@kenji-miyake Thank you for your confirmation.
If we were to add a new message, where would be the appropriate place to put it?

I think it needs to be demonstrated, so for now I am thinking it would be better to place new messages under tier4_system_msgs directory in tier4_autoware_msgs repository.

nabetetsu on Feb 6, 2023 Author

> I think @xmfcx -san is working on a similar task here:
> https://github.com/orgs/autowarefoundation/discussions/3194
> What's the comparison between his idea?

First, from an implementation standpoint, the proposal does not replace existing node classes. To minimize impact, the proposal uses existing timestamps.
From a functional standpoint, this proposal monitors the response time of paths and does not check whether a node is alive or dead, nor does it control the node itself.

xmfcx on Feb 6, 2023 Maintainer edited

Sorry, I've just started reading your proposal.

Meanwhile, could you also check https://github.com/ros-safety/software_watchdogs which uses `QoS Deadline and Liveliness policies` too?

kenji-miyake on Feb 7, 2023

> I think it needs to be demonstrated, so for now I am thinking it would be better to place new messages under tier4_system_msgs directory in tier4_autoware_msgs repository.

@nabetetsu If you want to discuss it within TIER IV, I think `tier4_autoware_msgs` is fine. If you want to discuss it also in AWF, sending a draft PR to `autoware_msgs` would be okay.

👍 1

xmfcx on Feb 6, 2023 Maintainer edited

@nabetetsu thanks for creating this proposal and comparing them to some potential implementations.

After carefully inspecting your proposal and the https://github.com/ros-safety/software_watchdogs repository I think it's a good idea to make use of deadline QoS policy.

Your current proposal includes:

- Adding new message type `MessageTrackingTag`
- Adding new publisher of the new message
- Adding new node to monitor timing violation

By using the deadline and liveliness policies, this can be simplified to either of following options, with different dynamics.

Liveliness

- Set the `liveliness` to `RMW_QOS_POLICY_LIVELINESS_MANUAL_BY_TOPIC` in both pub and sub.
- Set the `liveliness_lease_duration` in both pub and sub.
- Set the `liveliness_callback` for the subscriber like in the simple_watchdog.cpp.
  - The callback publishes the diagnostic message.

Deadline

- Set the deadline duration in the publisher
- Set a deadline callback `publisher_options.event_callbacks.deadline_callback` like in this ROS answer for the same publisher
  - The callback publishes the diagnostic message.
- Set the deadline duration in the subscriber
- You can also set a deadline callback for the subscriber side, which would achieve the same effect you are going for.

Comparison and suggestion

The liveliness option helps you to detect if there are any publishers for the subscribers at all with the checks.

The deadline option allows both the subscriber and publisher to monitor themselves.

Configuring a subscriber side callback and setting the `deadline` policies for both subscriber and publisher should have the functionality you are aiming. And I think you don't need to do anything with the `liveliness` policy.

This way you don't need to generate a new message type and a new monitoring node.

Extra information can be found in ROS QoS - Deadline, Liveliness, and Lifespan.

> **For Deadlines,** the Subscriber receives event notifications if it doesn't receive anything within the deadline and the Publisher receives event notifications if it doesn't publish anything within the deadline.
> **For Liveliness,** Subscribers receive events when there are no longer any Publishers alive to assert the topic is alive.

cc. @kenji-miyake

↑ 1 👍 2 1 reply

takam5f2 on Feb 8, 2023 Collaborator

@xmfcx
Thank for reply.
I want to add the supplement on behalf of @nabetetsu because the first post may have mislead you.

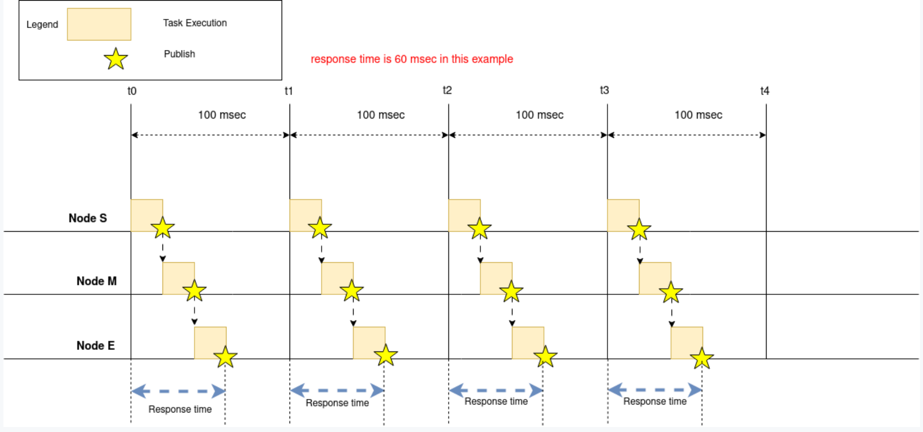
What to observe by the Timing violation monitor

Response time

The timing violation monitor will observe the response time taken by the path, and check if it is expected.
As mentioned above, a path is a series of multiple ROS 2 nodes. Their callback functions are executed in sequentially.
A node sends a topic message at the end of its callback execution, and the next node receives the message and executes the callback.
The response time is the processing time from the start of the first node to the end of the last node.

Let me show you an example that shows a path. The path consists of `Node S` , `Node M` , `Node E` . Each node is executed sequentially. `Node S` is triggered by 10 Hz timer. `Node M` is executed after `Node S` sends a topic message. `Node E` is triggered by a topic message sent by `Node M` as well.

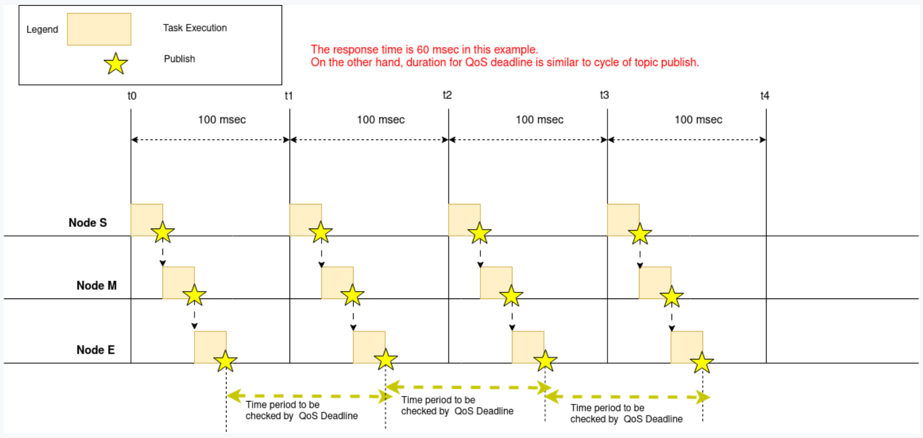
In this example, the path response time is the processing time from `Node S` to `Node E` per cycle, shown by the deep blue dotted lines `<–––>` . When each node costs 20 msec, the path response time is 60 msec. Response time is one of the most important metrics for real-time systems (see the page P.11 also).



What QoS Deadline checks

What QoS Deadline observes is different from the path response time. QoS Deadline checks the period between one publish execution to the next one. I want to give you another figure to show what period of time is being observed by QoS Deadline with the previous example.

In the following figure, the deep yellow dotted lines `<---->` show the time period between publishes on `Node E`. The time period is different from the path response time.

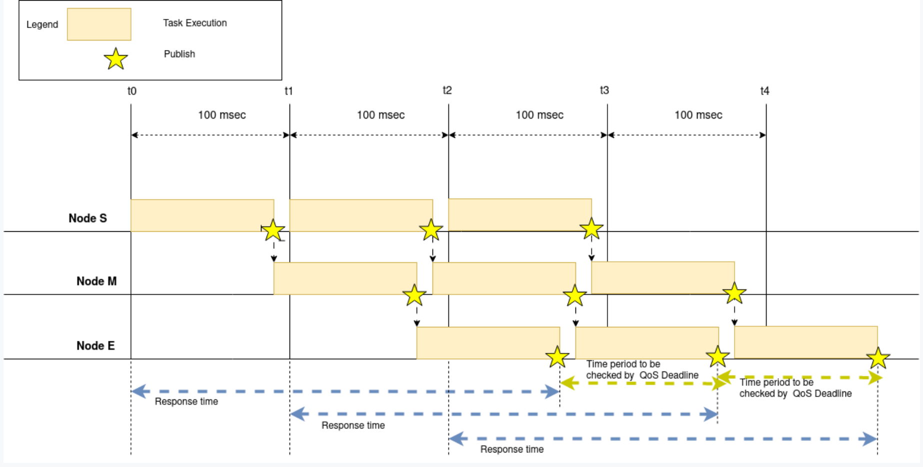

Legend
Task Execution
Publish
The response time is 60 msec in this example.
On the other hand, duration for QoS deadline is similar to cycle of topic publish.
t0
t1
t2
t3
t4
100 msec
100 msec
100 msec
100 msec
Node S
Node M
Node E
Time period to be checked by QoS Deadline
Time period to be checked by QoS Deadline
Time period to be checked by QoS Deadline

Difference between the response time and the time period

It seems to be sufficient to check only the time period with QoS deadline while the path response time does not need to be checked. However, sometimes the path response time is larger than expected while the time period is reasonable.

Let me revisit the previous example again in the following figure. The response time is 270 msec when each node costs 90 msec. Since the response time in the first example was 60 msec, 270 msec of the response time is hard to ignore.

On the other hand, the deep yellow dotted lines show that the time period is the same even if the response time increases. QoS Deadline setting is unable to detect the increase in response time.



Changes in the response time may affect the time period between publishes. However, I would like to clarify the response time in Autoware, and check if it is expected.

👍 3

takam5f2 on Feb 8, 2023 Collaborator

Here I'd like add another supplement to explain why the new message publisher is needed for the timing violation monitor.

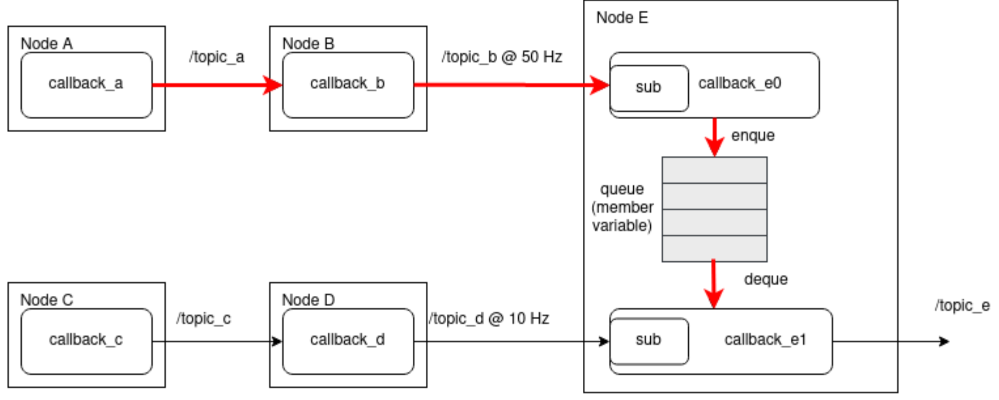
Supplements of path definition

To avoid confusion, I said that the path started at the beginning of the first node and ended at the end of the last node. That was for clarity of explanation.

I think that the path definition is not limited as explained above. The processing from creating a message to consuming it is considered as the path.

Let me give you another example to illustrate what I mean. The destination node, `Node E`, of the path has two callback functions that run at different frequencies. I can define two paths that pass `Node E`, the `Path 1` from `Node A` to `Node E` and `Path 2` from `Node C` to `Node D`. `callback_e0` is the callback that only receives the topic message and passes it to `callback_e1` via the member variable queue.

As well as explained above, the start of `Path 2` is the `callback_c` and the end is `callback_e1` where `callback_e1` sends `/topic_e`.
On the other hand, the end of `Path 1` has two candidates; `callback_e0` which receives the message and `callback_e1` which consumes it. To carefully check whether Autoware works as expected, we think, `callback_e1` should be the end of `Path 1`.



In this example, it is difficult to trace from external when the message is consumed in `callback_e1`. Then, `callback_e1` has to notify consumption of the queued message to external. That's why we want to introduce a new publisher for this kind of node.

Previously, `callbackInitialPose` callback in NDT Scan Matcher did not run at 50 Hz and it used old data as the given initial pose per cycle. NDT Scan Matcher suffered from increased computation cost due to inefficiency caused by old data, and NDT Scan Matcher sometimes failed in localization function.

↑ 1 👍 1 5 replies

xmfcx on Feb 8, 2023 Maintainer

Ok so this is something completely different.

In this context, we should be thinking about the time data takes as it travels through the pipeline. Like, tracing timings from sensor driver to the control/vehicle nodes.

I will also think about this but give me some time. Meanwhile, could you explain why we can't use CARET, a performance analysis tool for Autoware for this purpose?

I didn't investigate it completely it seems to be achieving same result, the delay in the pipeline for a given topic chain.
What prevents CARET from running in real time?

The reason I'm asking this is, to know if it's possible to track/trace the timing of the messages without adding extra messages to the pipeline.

👍 1

nabetetsu on Feb 9, 2023 Author

As you say, our objective is to monitor and check the response time as data moves through the pipeline.

About CARET:
CARET hooks into the send/receive function of a pcl or DDS layer and outputs LTTng trace points and additional information specified by CARET.
To get the response time spent on the pipeline, you need to analyze the output data.
In addition, you must write a configuration file, called an architecture file, in which you specify the node latency and the destination path before analyzing the dataset.
Because of this specification, it is not possible to get the path response time when Autoware is running in CARET.

Regarding the addition of messages:
One of the time constraints of NDT is up to data consumption, at which point no existing topics are sent.
Therefore, in our opinion, it is necessary to indicate the end of the path in one of two ways.

1. add a publish of an existing message type
2. add a publish of a new message type

👍 1 👀 1

takam5f2 on Feb 9, 2023 Collaborator

> In this context, we should be thinking about the time data takes as it travels through the pipeline. Like, tracing timings from sensor driver to the control/vehicle nodes.

Exactly.

> I will also think about this but give me some time. Meanwhile, could you explain why we can't use #2630 for this purpose?
> I didn't investigate it completely it seems to be achieving same result, the delay in the pipeline for a given topic chain.
> What prevents CARET from running in real time?

CARET allows us to observe how much time costs when data takes as it takes for data to travel through the pipeline. We don't have to change any code to do this and we can measure response time in multiple paths. We don't have to specify the paths before we measure the response time.
CARET is designed as a performance analysis tool, so that it returns the response time outside of Autoware execution, not during execution. The output format is CTF and it takes tens of seconds or minutes to parse the format.
In my opinion, CARET should be a tool used for integration testing or field testing, but not recommended for production use. CARET adds overhead to the execution of all components and we don't want to observe timing violations of all of paths and components.

However, we do want to monitor the response time of the important paths even in production. The important paths obviously have timing constraints and their number may not be many.
We also want to care about the response time when it is larger than expected, but we don't usually have to care about it . Those motivated us to propose the timing violation monitor.
When Autoware has failed in a certain function after starting the service, we are asked for accountability. The timing violation monitor will help us to explain the failure from the point of view of the response time. It will help us to investigate the cause of the failure.

👍 1

xmfcx on Feb 9, 2023 Maintainer

@nabetetsu @takam5f2 Thank you for the explanations. Could you give me a couple days to think on this a bit more? Or is it urgent?

👍 3

takam5f2 on Feb 10, 2023 Collaborator

@xmfcx
Thank for checking.

> Could you give me a couple days to think on this a bit more?

Yes, of course.
We are preparing Pull Request for more details.

nabetetsu on Feb 16, 2023 Author

To introduced this feature, it needs three PRs for adding the dedicated message, notifying end of path in `ndt_scan_matcher` and `timing_violation_monitor` node itself.

We created two PR as following:

- add the dedicated message
  - feat(TimingViolationMonitor): add new message `MessageTrackingTag` used in `timing_violation_monitor` tier4/tier4_autoware_msgs#74
- add the `timing_violation_monitor` library and add function to notify end of path
  - feat(ndt_scan_matcher & timing_violation_monitor): add function to notify end of Path to `timing_violation_monitor` autoware.universe#2638

Remained PR for `timing_violation_monitor` node implementation will be created within next week.

↑ 2 👍 1 0 replies