# ndn-tools - Feature #4289

Feature # 1624 (In Progress): Design and Implement Congestion Control

## ndncatchunks: React to congestion marks

09/15/2017 04:18 PM - Klaus Schneider

| | | | |
|---|---|---|---|
| **Status:** | Closed | **Start date:** | |
| **Priority:** | Normal | **Due date:** | |
| **Assignee:** | Chavoosh Ghasemi | **% Done:** | 100% |
| **Category:** | | **Estimated time:** | 0.00 hour |
| **Target version:** | | | |

**Description**

ndncatchunks should be able to react to the congestion marks implemented in feature #3797.

The reaction should be similar to the one of a timeout, and the **window adaptation** code should separated and called on both timeouts and received congestion marks.

**Related issues:**

| | | |
|---|---|---|
| Blocked by NFD - Feature #3797: Congestion Control: generic congestion marks | **Closed** | |
| Blocked by ndn-tools - Feature #4299: ndncatchunks: Print summary for all pip... | **Closed** | **09/22/2017** |
| Blocks ndn-cxx - Feature #4364: Implement congestion control in SegmentFetcher | **Closed** | |

## History

**#1 - 09/15/2017 04:19 PM - Klaus Schneider**

We agreed that Chavoosh can work on this.

Please let me know if any of the design is still unclear.

**#2 - 09/15/2017 04:19 PM - Klaus Schneider**

*- Related to Feature #3797: Congestion Control: generic congestion marks added*

**#3 - 09/15/2017 04:19 PM - Klaus Schneider**

*- Related to Feature #1624: Design and Implement Congestion Control added*

**#4 - 09/15/2017 05:15 PM - Davide Pesavento**

*- Start date deleted (09/15/2017)*

is this related to #3860 in any way?

**#5 - 09/15/2017 05:16 PM - Davide Pesavento**

*- Tracker changed from Task to Feature*

**#6 - 09/15/2017 05:24 PM - Klaus Schneider**

Yeah, but these issues don't depend upon each other.

Any of the rate adaptation algorithms (AIMD, Bic, Cubic) should be able to use the congestion marks.

**#7 - 09/15/2017 05:28 PM - Klaus Schneider**

Since Eric finished the implementation of congestion marks, we think that this feature is more important than implementing BIC/Cubic.

**#8 - 10/17/2017 04:02 PM - Davide Pesavento**

*- Parent task set to #1624*

**#9 - 10/17/2017 04:05 PM - Davide Pesavento**

*- Blocked by Feature #3797: Congestion Control: generic congestion marks added*

**#10 - 10/17/2017 04:06 PM - Davide Pesavento**

*- Related to deleted (Feature #3797: Congestion Control: generic congestion marks)*

**#11 - 10/26/2017 03:00 PM - Klaus Schneider**

*- Related to Feature #4299: ndncatchunks: Print summary for all pipeline types added*

**#12 - 10/26/2017 03:00 PM - Klaus Schneider**

Related to #4299 since Chavoosh needs to finish that task first.

**#13 - 10/26/2017 03:51 PM - Davide Pesavento**

*- Related to deleted (Feature #4299: ndncatchunks: Print summary for all pipeline types)*

**#14 - 10/26/2017 03:51 PM - Davide Pesavento**

*- Blocked by Feature #4299: ndncatchunks: Print summary for all pipeline types added*

**#15 - 10/27/2017 01:52 PM - Klaus Schneider**

*- Blocks Feature #4364: Implement congestion control in SegmentFetcher added*

**#16 - 10/28/2017 05:03 PM - Chavoosh Ghasemi**

In our weekly meeting we decided to take the following steps to enable ndncatchunks to react to the congestion marks:

- Figuring out whether there is a congestion mark in the tags of the received data packet
- Finding and not finding a congestion mark in the received data packet should trigger different methods, consequently
- We think the best place to implement this feature would be in "PipelineInterests.cpp" file, so all pipelines can use it

**#17 - 10/28/2017 06:09 PM - Klaus Schneider**

Just a note: you can get the right bullet point formatting by writing:

- One
- Two
- Three

(there needs to be a blank line between text and bullet points)

**#18 - 11/04/2017 11:49 AM - Chavoosh Ghasemi**

To make sure we are all agree about the implementation details, here I want to discuss details of mentioned steps:

- By using getTag<lp::CongestionMarkTag>() method we can determine whether the data packet has congestion mark in its tags or not. This method should be called for each received data packet.
- We need to add a new method to pipeline-interests.hpp file, named checkCongestionMark(). If you think this name is not informative enough, suggest your name.
- If there was not any congestion mark, we should call PipelineInterestsAimd::increaseWindow() method. Otherwise, PipelineInterestsAimd::decreaseWindow() method should be triggered. Worth to mention that we need to move these methods to pipeline-interests.hpp file and make it virtual, such that each subclass can implement it, consequently. BTW, if we need to change the way PipelineInterestsAimd::decreaseWindow() should react to congestion mark, let me know.
- Again, if you think pipeline-interests.hpp is not a good place to put this functionality, discuss it.

**#19 - 11/04/2017 12:26 PM - Davide Pesavento**

Chavoosh Ghasemi wrote:

> - By using getTag<lp::CongestionMarkTag>() method we can determine whether the data packet has congestion mark in its tags or not.

No, you should use getCongestionMark().

> - We need to add a new method to pipeline-interests.hpp file, named checkCongestionMark().

What will it do?

> - Again, if you think pipeline-interests.hpp is not a good place to put this functionality, discuss it.

I'm not convinced about this. But I don't have a strong opinion either, so you might convince me if you have good reasons to do it. My rationale against is that I'm not sure how much code can actually be shared among different pipelines. It's hard to see also because we currently have only one pipeline with a dynamic window, and I can't picture how potential future pipelines might look like.

**#20 - 11/04/2017 01:12 PM - Chavoosh Ghasemi**

> > What will it do?
>
> The input of this method is a data packet and its output would be boolean (i.e. whether congestion mark exists or not). Anyway, we can shift this functionality into handleData() method; however, for readability of the code we can make a new method. BTW, I am OK with either one.
>
> > I'm not convinced about this. But I don't have a strong opinion either, so you might convince me if you have good reasons to do it. My rationale against is that I'm not sure how much code can actually be shared among different pipelines. It's hard to see also because we currently have only one pipeline with a dynamic window, and I can't picture how potential future pipelines might look like.
>
> Regardless of the pipeline's mechanisms (current and future ones), we need to add a virtual method in order to react to congestion mark. It is up to each pipeline to use it or not. A pipeline can react to congestion mark (e.g. AIMD) or it can ignore it (e.g. fixed-window). In the first case, the pipeline simply does not do anything. However, for the second case the pipeline implement this virtual method and will react to congestion mark.

**#21 - 11/04/2017 01:14 PM - Chavoosh Ghasemi**

Sorry. I am still kind of confused how to properly use this **edit** tool.

**#22 - 11/04/2017 01:28 PM - Davide Pesavento**

Chavoosh Ghasemi wrote:

> Sorry. I am still kind of confused how to properly use this **edit** tool.

You can edit your comment to fix the formatting. Also note that there is a "preview" button next to "submit" that you can use...

**#23 - 11/04/2017 02:22 PM - Klaus Schneider**

> Regardless of the pipeline's mechanisms (current and future ones), we need to add a virtual method in order to react to congestion mark. It is up to each pipeline to use it or not. A pipeline can react to congestion mark (e.g. AIMD) or it can ignore it (e.g. fixed-window). In the first case, the pipeline simply does not do anything. However, for the second case the pipeline implement this virtual method and will react to congestion mark.

I would move "PipelineInterestsAimd::recordTimeout()" to "PipelineInterests::recordTimeout()". Maybe rename to "handleTimeout()".

Then add a function "PipelineInterests::handleCongEvent()". Where "congestion event" means either congestion mark or timeout.

handleTimeout() should call handleCongEvent():

```
PipelineInterests::handleTimeout()
{
  handleCongEvent();
  retransmitInterest();
}
```

I think this function should work for practically all pipelines, except the fixed pipeline. PipelineFixed should overwrite handleTimeout() to do nothing.

**#24 - 11/04/2017 02:28 PM - Klaus Schneider**

> > Again, if you think pipeline-interests.hpp is not a good place to put this functionality, discuss it.
>
> I'm not convinced about this. But I don't have a strong opinion either, so you might convince me if you have good reasons to do it. My rationale against is that I'm not sure how much code can actually be shared among different pipelines. It's hard to see also because we currently have only one pipeline with a dynamic window, and I can't picture how potential future pipelines might look like.

Here's some assumptions that I would make about future pipelines:

- All of them will consider timeouts: after a timeout, they will both retransmit and reduce their sending rate.
- All of should have a function to consider congestion marks, but may choose not to use it (ignore congestion marks)
- Most of them will be *window-based* (example: BIC, Cubic), but I can also imagine rate-based ones.

**#25 - 11/04/2017 05:42 PM - Davide Pesavento**

Chavoosh Ghasemi wrote:

> > The input of this method is a data packet and its output would be boolean (i.e. whether congestion mark exists or not). Anyway, we can shift this functionality into handleData() method; however, for readability of the code we can make a new method. BTW, I am OK with either one.

This doesn't provide any additional logic compared to getCongestionMark().

> > Regardless of the pipeline's mechanisms (current and future ones), we need to add a virtual method in order to react to congestion mark. It is up to each pipeline to use it or not. A pipeline can react to congestion mark (e.g. AIMD) or it can ignore it (e.g. fixed-window). In the first case, the pipeline simply does not do anything. However, for the second case the pipeline implement this virtual method and will react to congestion mark.

The question is how much logic is abstracted into the base class method. If it's just one simple "if", I don't think it's worth it.

**#26 - 11/04/2017 06:02 PM - Chavoosh Ghasemi**

Davide Pesavento wrote:

> The question is how much logic is abstracted into the base class method. If it's just one simple "if", I don't think it's worth it.

Reacting to congestion control highly depends on pipeline's mechanism and we cannot shift too much functionalities to the base class. So, maybe we only need to define a virtual method in the base class that checks the very basic conditions and after that calls a proper method, due to some conditions. However, since different pipelines can have different logic we cannot abstract too much logic into the base class.

**#27 - 11/04/2017 06:06 PM - Chavoosh Ghasemi**

Klaus Schneider wrote:

> I would move "PipelineInterestsAimd::recordTimeout()" to "PipelineInterests::recordTimeout()". Maybe rename to "handleTimeout()".
>
> Then add a function "PipelineInterests::handleCongEvent()". Where "congestion event" means either congestion mark or timeout.
>
> handleTimeout() should call handleCongEvent():
>
> ```
> PipelineInterests::handleTimeout()
> {
>   handleCongEvent();
>   retransmitInterest();
> }
> ```
>
> I think this function should work for practically all pipelines, except the fixed pipeline. PipelineFixed should overwrite handleTimeout() to do nothing.

I am wondering whether it is reasonable to define these functions in the base class or not. For instance, AIMD might handle congestion mark totally different from another pipeline like Cubic. Thus, how these functions should be defined in the base class?

**#28 - 11/04/2017 07:22 PM - Klaus Schneider**

Chavoosh Ghasemi wrote:

> Klaus Schneider wrote:
>
> > I would move "PipelineInterestsAimd::recordTimeout()" to "PipelineInterests::recordTimeout()". Maybe rename to "handleTimeout()".
> >
> > Then add a function "PipelineInterests::handleCongEvent()". Where "congestion event" means either congestion mark or timeout.
> >
> > handleTimeout() should call handleCongEvent():
> >
> > ```
> > PipelineInterests::handleTimeout()
> > {
> >   handleCongEvent();
> >   retransmitInterest();
> > }
> > ```
> >
> > I think this function should work for practically all pipelines, except the fixed pipeline. PipelineFixed should overwrite handleTimeout() to do nothing.

> I am wondering whether it is reasonable to define these functions in the base class or not. For instance, AIMD might handle congestion mark totally different from another pipeline like Cubic. Thus, how these functions should be defined in the base class?

The function definition (the .hpp part) of handleCongEvent(); should be in the base class, the actual implementation (the .cpp part) should only be in the sub-classes.

> Reacting to congestion control highly depends on pipeline's mechanism and we cannot shift too much functionalities to the base class. So, maybe we only need to define a virtual method in the base class that checks the very basic conditions and after that calls a proper method, due to some conditions. However, since different pipelines can have different logic we cannot abstract too much logic into the base class.

This paragraph is very long, yet contains little information. Please be more specific.

It poses the following questions:

- What are "too much functionalities"?
- What are "very basic conditions"?
- Which is the "proper method"?
- What are "some conditions"?
- How much is "too much logic"?

I agree that the base class should only have a virtual method for handleCongEvent();

But I think the base class can and should contain the full implementation of handleTimeout() and retransmitInterest();

**#29 - 11/04/2017 09:53 PM - Chavoosh Ghasemi**

Klaus Schneider wrote:

> The function definition (the .hpp part) of handleCongEvent(); should be in the base class, the actual implementation (the .cpp part) should only be in the sub-classes.

You still did not answer my question, why we need to have them in the base class? Maybe you mean defining two interfaces and implement them in the subclass. That's the point?

>> Reacting to congestion control highly depends on pipeline's mechanism and we cannot shift too much functionalities to the base class. So, maybe we only need to define a virtual method in the base class that checks the very basic conditions and after that calls a proper method, due to some conditions. However, since different pipelines can have different logic we cannot abstract too much logic into the base class.
>
> This paragraph is very long, yet contains little information. Please be more specific.
>
> It poses the following questions:
>
> - What are "too much functionalities"?
> - What are "very basic conditions"?
> - Which is the "proper method"?
> - What are "some conditions"?
> - How much is "too much logic"?

The reason of ambiguity of this comment is we do not know details of implementation (e.g. which functions should be called facing different cases, what are their names, where to implement them, etc.).

Anyway, as I mentioned in previous comments "proper methods" are those that should be called upon finding or not finding congestion mark (e.g. for decreasing or increasing window size, re-transmissions, or any other mechanism that the corresponded pipeline needs to take into account according to its implementation).

"Basic conditions" (from my point of view) include simple tasks like checking the timestamps of the received packets and some if statements (that I am not sure about their context).

I do not have any unit of "how much is too much logic!" but when you have no idea about the future pipeline implementations (your mentioned assumptions do not clear their implementation) their logic can be totally different. Putting logic of AIMD in the pipeline and expecting (or hoping) the other pipelines to follow them is not reasonable.

> I agree that the base class should only have a virtual method for handleCongEvent();
>
> But I think the base class can and should contain the full implementation of handleTimeout() and retransmitInterest();

handleTimeout() and retransmitInterest() can differ from one pipeline to another one, how do you define full implementation?

**#30 - 11/04/2017 10:22 PM - Klaus Schneider**

Chavoosh Ghasemi wrote:

> Klaus Schneider wrote:
>
> > The function definition (the .hpp part) of handleCongEvent(); should be in the base class, the actual implementation (the .cpp part) should only be in the sub-classes.
>
> You still did not answer my question, why we need to have them in the base class? Maybe you mean defining two interfaces and implement them in the subclass. That's the point?

We should have the interface in the base class, since all potential pipelines would want to implement some version of handleCongEvent();

I think the most common implementation should be to perform a window decrease.

Regarding your example: Even if AIMD handles congestion mark differently from Cubic, the interface stays the same.

> > I agree that the base class should only have a virtual method for handleCongEvent();
> >
> > But I think the base class can and should contain the full implementation of handleTimeout() and retransmitInterest();
>
> handleTimeout() and retransmitInterest() can differ from one pipeline to another one, how do you define full implementation?

This depends on the implementation of these two functions; you can implement both functions in a way that they work for all possible pipelines.

If the handleTimeout() follows the implementation that I list above, then it *does not* differ for each pipeline. It calls functions that differ for each pipeline, but the handleTimeout() implementation itself stays the same.

The same goes for retransmitInterests. The closest to this function is the current function enqueueForRetransmission(), here printed in full:

```
void
PipelineInterestsAimd::enqueueForRetransmission(uint64_t segNo)
{
  BOOST_ASSERT(m_nInFlight > 0);
  m_nInFlight--;
  m_retxQueue.push(segNo);
  m_segmentInfo.at(segNo).state = SegmentState::InRetxQueue;
}
```

This function seems generic enough to be used for all possible pipelines.

**#31 - 11/04/2017 10:49 PM - Davide Pesavento**

Chavoosh Ghasemi wrote:

> Davide Pesavento wrote:
>
> > The question is how much logic is abstracted into the base class method. If it's just one simple "if", I don't think it's worth it.
>
> Reacting to congestion control highly depends on pipeline's mechanism and we cannot shift too much functionalities to the base class. So, maybe we only need to define a virtual method in the base class that checks the very basic conditions and after that calls a proper method, due to some conditions. However, since different pipelines can have different logic we cannot abstract too much logic into the base class.

That's my point. If you abstract just one "if" condition in the base class (e.g. pkt.getCongestionMark() > 0), then it's definitely not worth it. It would *add* complexity instead of removing it.

**#32 - 11/04/2017 10:52 PM - Klaus Schneider**

Davide Pesavento wrote:

> That's my point. If you abstract just one "if" condition in the base class (e.g. pkt.getCongestionMark() > 0), then it's definitely not worth it. It would *add* complexity instead of removing it.

Yeah, I agree that the "checkCongestionMark()" function would likely be too much overhead, and best replaced by an if-condition.

**#33 - 11/04/2017 10:56 PM - Davide Pesavento**

Klaus Schneider wrote:

> We should have the interface in the base class, since all potential pipelines would want to implement some version of handleCongEvent();

To what purpose? This is not a library, and the pipeline is not a component that can be used "from outside" (except for the initial run()). Having a common interface among pipelines is a *non-goal*. You should put things in the base class only if it simplifies the *implementation* of the various subclasses, e.g. common/repeated logic.

> [...]
> This function seems generic enough to be used for all possible pipelines.

Let's not speculate. I strongly suggest adding this functionality to the AIMD pipeline only for now. When we're about to add other pipeline types, we can re-evaluate whether some pieces can be shared among them.

**#34 - 11/04/2017 11:51 PM - Klaus Schneider**

Davide Pesavento wrote:

> Klaus Schneider wrote:
>
> > We should have the interface in the base class, since all potential pipelines would want to implement some version of handleCongEvent();
>
> To what purpose? This is not a library, and the pipeline is not a component that can be used "from outside" (except for the initial run()). Having a common interface among pipelines is a *non-goal*. You should put things in the base class only if it simplifies the *implementation* of the various subclasses, e.g. common/repeated logic.

So does this mean that there should not be any function declaration in the base class, unless the base class also implements the given function?

> > [...]
> > This function seems generic enough to be used for all possible pipelines.
>
> Let's not speculate. I strongly suggest adding this functionality to the AIMD pipeline only for now. When we're about to add other pipeline types, we can re-evaluate whether some pieces can be shared among them.

I agree with the general point (of not speculating), but there might be less speculation involved than you think.

If we are talking about adding the BIC or Cubic pipeline (which we planned to do earlier), then the current enqueueForRetransmission() function will definitely work.

The main difference between AIMD, BIC, and Cubic can be put in two functions: windowIncrease() and windowDecrease(). windowIncrease() is called on every normal Data packet; windowDecrease() is called on every congestion mark and timeout.

But we could also implement BIC and Cubic as sub-classes of AIMD. I don't have any strong preference either way.

**#35 - 11/05/2017 08:49 AM - Davide Pesavento**

Klaus Schneider wrote:

> Davide Pesavento wrote:
>
> > To what purpose? This is not a library, and the pipeline is not a component that can be used "from outside" (except for the initial run()). Having a common interface among pipelines is a *non-goal*. You should put things in the base class only if it simplifies the *implementation* of the various subclasses, e.g. common/repeated logic.
>
> So does this mean that there should not be any function declaration in the base class, unless the base class also implements the given function?

Basically yes, but it's not a strict rule, there can be exceptions, e.g. run/doRun, template method pattern, etc...

> But we could also implement BIC and Cubic as sub-classes of AIMD. I don't have any strong preference either way.

This sounds like a good idea.

**#36 - 11/05/2017 08:56 AM - Chavoosh Ghasemi**

Davide Pesavento wrote:

> Klaus Schneider wrote:

> > Davide Pesavento wrote:
> >
> > > To what purpose? This is not a library, and the pipeline is not a component that can be used "from outside" (except for the initial run()). Having a common interface among pipelines is a *non-goal*. You should put things in the base class only if it simplifies the *implementation* of the various subclasses, e.g. common/repeated logic.
> >
> > So does this mean that there should not be any function declaration in the base class, unless the base class also implements the given function?
>
> Basically yes, but it's not a strict rule, there can be exceptions, e.g. run/doRun, template method pattern, etc...

I highly suggest to specifically talk about the details of each function:

- what functions we need to define and implement (choose a name for them, so we can reference them in our discussion)
- the context of each function (so we can figure out whether to put it in the base class or not - based on Davide's comment)

> > But we could also implement BIC and Cubic as sub-classes of AIMD. I don't have any strong preference either way.
>
> This sounds like a good idea.

Is this something that should be done in this issue??

**#37 - 11/05/2017 09:00 AM - Davide Pesavento**

Chavoosh Ghasemi wrote:

> I highly suggest to specifically talk about the details of each function:

You haven't pushed any code yet, so it's hard to be more specific than this.

> > > But we could also implement BIC and Cubic as sub-classes of AIMD. I don't have any strong preference either way.
> >
> > This sounds like a good idea.
>
> Is this something that should be done in this issue??

No. There are separate issues for BIC and Cubic.

**#38 - 11/05/2017 09:24 AM - Chavoosh Ghasemi**

Davide Pesavento wrote:

> Chavoosh Ghasemi wrote:
>
> > I highly suggest to specifically talk about the details of each function:
>
> You haven't pushed any code yet, so it's hard to be more specific than this.

I think when we did not even conclude our discussion pushing any code would be to no avail.
Anyway, here is what we need to add to the source code, after our discussion. Note that I am assuming we are implementing this functionality just for AIMD pipeline in this issue.

```
PipelineInterestsAimd::handleData
{
    ...
    if (data.getCongestionMark())
    {
        decreaseWindow();
        ..
    }
    ..
}
```

I think we do not need to touch retransmission mechanism as congestion mark does not trigger packet retransmission.

About handleCongEvent() method, if we do not implement this functionality in the base class, having such method is not necessary.

**#39 - 11/05/2017 01:07 PM - Klaus Schneider**

Chavoosh Ghasemi wrote:

> Davide Pesavento wrote:
>
> > Chavoosh Ghasemi wrote:
> >
> > > I highly suggest to specifically talk about the details of each function:
> >
> > You haven't pushed any code yet, so it's hard to be more specific than this.
>
> I think when we did not even conclude our discussion pushing any code would be to no avail.
> Anyway, here is what we need to add to the source code, after our discussion. Note that I am assuming we are implementing this functionality just for AIMD pipeline in this issue.
>
> ```
> PipelineInterestsAimd::handleData
> {
>     ...
>     if (data.getCongestionMark())
>     {
>         decreaseWindow();
>         ..
>     }
>     ..
> }
> ```

Looks good to me.

> I think we do not need to touch retransmission mechanism as congestion mark does not trigger packet retransmission.

Makes sense.

> About handleCongEvent() method, if we do not implement this functionality in the base class, having such method is not necessary.

I think for all window-based congestion control schemes (AIMD, BIC, CUBIC, etc.), handleCongEvent() is equivalent to decreaseWindow(). So no need to have this extra function.

I think this outline is well-specified enough for you to start coding. Or do you have any other questions?

@Davide: Do we need to wait for your current change to get merged before Chavoosh can start coding?

**#40 - 11/05/2017 01:19 PM - Davide Pesavento**

Klaus Schneider wrote:

> @Davide: Do we need to wait for your current change to get merged before Chavoosh can start coding?

No. Chavoosh can download change 4343 onto his machine and start developing on top of it.

**#41 - 11/05/2017 01:44 PM - Chavoosh Ghasemi**

How can I create a new issue on gerrit for this change? Does it require any permission?

**#42 - 11/06/2017 01:43 PM - Chavoosh Ghasemi**

Chavoosh Ghasemi wrote:

> How can I create a new issue on gerrit for this change? Does it require any permission?

Since I did not hear from you, I asked Eric about my question. Apparently the only thing that I need to do is pushing my changes to gerrit and everything else will be handled by gerrit itself. Previously, I thought I need to create a new issue (or ticket) on gerrit for this change.

**#43 - 11/06/2017 01:48 PM - Klaus Schneider**

Yes, the "git-review" command will create the new issue, just like you created it last time :)

**#44 - 11/06/2017 01:50 PM - Klaus Schneider**

Btw, it seems like you edited your earlier comment, instead of adding a new one.

This is why we didn't get an email notification, and likely why we didn't reply.

**#45 - 11/06/2017 02:06 PM - Chavoosh Ghasemi**

Klaus Schneider wrote:

> Btw, it seems like you edited your earlier comment, instead of adding a new one.
>
> This is why we didn't get an email notification, and likely why we didn't reply.

I see. I am not sure what exactly happened. Thanks for your comment, though.

**#46 - 11/06/2017 06:34 PM - Klaus Schneider**

Chavoosh Ghasemi wrote:

> Klaus Schneider wrote:
>
> > Btw, it seems like you edited your earlier comment, instead of adding a new one.
> >
> > This is why we didn't get an email notification, and likely why we didn't reply.
>
> I see. I am not sure what exactly happened. Thanks for your comment, though.

There is an edit button (pencil) next to your comment, and one at the bottom of all comments. It's quite possible to confuse them.

**#47 - 11/07/2017 01:42 PM - Klaus Schneider**

*- Status changed from New to Code review*

**#48 - 11/17/2017 12:00 AM - Klaus Schneider**

*- % Done changed from 0 to 100*

**#49 - 11/20/2017 07:34 AM - Davide Pesavento**

*- Status changed from Code review to Closed*