NFD - Bug #3484

High PIT usage after NdnCon conference

02/28/2016 09:02 AM - Jeff Burke

Status:	Closed	**Start date:**	02/28/2016
Priority:	Normal	**Due date:**	
Assignee:	Junxiao Shi	**% Done:**	100%
Category:	Forwarding	**Estimated time:**	4.00 hours
Target version:	v0.5		

Description

The NDN seminar on 20160224 used *NdnCon* on the NDN testbed.

During the conference, two producers peter and confbridge are connected to REMAP gateway router, their prefixes are setup with nfd-autoreg, and the namespace has AccessStrategy.

After the conference ends, on REMAP gateway router, nfd-status -v shows that the PIT contains 52185 entries, and the NameTree contains 96278 entries, despite that there's very low traffic volume; these counters stay at high levels even after several hours.

This situation is not observed on other testbed gateway routers, where there are at most one producer relying on nfd-autoreg.

History

#1 - 02/28/2016 11:22 AM - Peter Gusev

The maximum Interest lifetime:

- chasing & state initial states: 3000 ms (https://github.com/remap/ndnrtc/blob/master/cpp/src/pipeliner.cpp#L302)
- other states: whatever is set in configuration - default is 2000ms (https://github.com/remap/ndnrtc/blob/master/cpp/src/pipeliner.cpp#L310)

#2 - 02/28/2016 12:21 PM - Alex Afanasyev

I have taken the coredump on REMAP node, so we can analyze the memory and see what specific entries are in memory. To do actual analysis, we need somebody to write gdb scripts to print out memory structures in readable form.

Relevant resources for that: http://stackoverflow.com/questions/14698086/gdb-python-scripting-any-samples-iterating-through-c-c-struct-fields and http://docs.adacore.com/gdb-docs/html/gdb_24.html.

#3 - 02/28/2016 12:25 PM - Alex Afanasyev

On a separate note. I am curios not about minimal lifetime size, but rather maximum lifetime size set in NDN-RTC interests. This is an artifact of the current protocol definition and implementation, but if lifetime is set too long, the PIT entry can legitimately persist in NFD for a long period of time.

Another separate note. The excessive number of entries seem to be only with REMAP hub. UCLA, Arizona, and Memphis hubs (were part of ndnrtc test) have small numbers of PIT and NameTree entries.

#4 - 03/01/2016 07:06 AM - Junxiao Shi

- Category set to Tables

It's probable that a ndncon component other than NDN-RTC (such as ndn-cpp's ChronoSync) is using large InterestLifetime.

#5 - 03/01/2016 01:33 PM - Jeff Burke

Junxiao believes may be related to #3219. Alex looked at memory dump from REMAP node and says (typical?) interest lifetime is 2 sec.

#6 - 03/01/2016 02:02 PM - Zhehao Wang

For other components of NdnCon mentioned by Junxiao: we also uses ChronoSync from ndn-cpp, and a discovery for conferences, users, and chatrooms.

The interests sent by these components should have 5s maximum lifetime. (Most of those interests have 2s lifetime, as mentioned by Alex).

A few other places that set the interest lifetime in the application code:
Sync interest
Sync interest timeout
Sync interest update
Entity discovery interest

Entity discovery heartbeat interest

And default lifetime configuration.

And a few occurrences such as this in ChronoSync2013 in ndn-cpp.

Please note that the occasional "/local/timeout" interest in the code is used as a timer in ndn-cpp, and is not propagated to nfd.

#7 - 03/01/2016 03:11 PM - Junxiao Shi

- Tracker changed from Task to Bug

- Subject changed from Debug unexpectedly large entries in NameTree and PIT to High PIT usage after NdnCon conference

- Description updated

I'm rewriting this issue as a Bug with more background factual information.

Original description:

After 2/24 NDN seminar test of NDN-RTC:
"For example, when I look at REMAP right now I see this:
nNameTreeEntries=96278
nPitEntries=52185
Perhaps our nfd developers can tell us if these values make sense with a “quiet” system. Or should they have returned to near 0 when traffic goes away. Why do we have 52K PIT entries at this point? Shouldn’t they have timed out and gone away…"

This is not normal.... In normal cases, PIT entries should have timed out (we haven't added yet a guard against the case when Interests specifies large value for Interest lifetime, but this may or may not be the case here.)

@Peter, what is max Interest lifetime in NDN-RTC?
@NFD developers, is there / can we have a mechanism to dump these entries from a running NFD?

#8 - 03/01/2016 03:33 PM - Peter Gusev

The maximum lifetime for the Interest is 3 seconds in NDN-RTC.

ndncon also uses ConferenceDiscovery library for chat and user discovery, as well as for chat messaging. I don't think it uses Interests with lifetimes larger than 5 sec. @Zhehao can confirm on this.

#9 - 03/01/2016 03:34 PM - Junxiao Shi

- Assignee set to Junxiao Shi

- Target version set to v0.5

Interests with large InterestLifetime can legitimately stay in the PIT for a long time.
There are discussions about imposing an upper bound of allowable InterestLifetime, but such practice violates NDN service semantics and should be settled with #2551.

However, note-1 note-5 note-6 can mostly rule out this Bug being caused by Interests with large InterestLifetime.

During 20160301 call, Alex reveals that from the coredump he took on REMAP gateway router several hours after the end of NdnCon conference, he could see that InterestLifetime of most packets are 2 seconds, and **both *unsatisfy timer* and *straggler timer* are empty**.

By design of forwarding pipelines, a PIT entry should have either timer set at all times (exception: if pipeline is executing when coredump is taken, the one PIT entry used in the executing pipeline is exempt).
Therefore, I suspect this is a bug in forwarding pipelines: in certain conditions, **there exists a code path that causes a PIT entry not to have either timer**; afterwards, this **PIT entry is leaked** if there isn't another incoming packet that causes a pipeline execution on this PIT entry.

#10 - 03/02/2016 06:25 PM - Junxiao Shi

- File 2016-03-02 19.09.52.jpg added

- Description updated

- Category changed from Tables to Forwarding

I found one code path that can lead to a PIT entry left without either timer.

2016-03-02 19.09.52.jpg
The photo shows 6 forwarding pipelines that contain operations of setting or cancelling the timers, highlighted in red.
The blue line indicates the portions of this control flow where neither timer is set.
As we can see, there is one blue line that continues toward the termination of forwarding pipelines, and this is where a PIT entry leak can happen.

The control flow that can leak a PIT entry is:

1. in incoming Interest pipeline, a new PIT entry is created (which comes without timers)
2. a duplicate Nonce is detected, so control flow goes to Interest loop pipeline
3. no timer is set in Interest loop pipeline

This is a design problem introduced with Dead Nonce List (DNL, #1953).

Without DNL, duplicate Nonce detection relies on the PIT entry, and it's impossible for an incoming Interest to be detected as having duplicate Nonce if the PIT entry is newly created, so step2 cannot happen.

With DNL, a duplicate Nonce can be detected from DNL even if the PIT entry is new, and then PIT entry is leaked.

PIT entries leaked by the above manner should have no in-record and no out-record.

@Alex, can you look at the coredump and confirm there is no in-record and no out-record?

#11 - 03/03/2016 01:10 AM - Alex Afanasyev

I can confirm, for a few records I looked, in and out records are empty

```
(gdb) print (*$tree.m_buckets[1131].m_entry.get().m_pitEntries[0].get())
$14 = {
  <nfd::StrategyInfoHost> = {
    m_items = std::map with 0 elements
  },
  <boost::noncopyable_::noncopyable> = {<No data fields>},
  members of nfd::pit::Entry:
  m_unsatisfyTimer = std::shared_ptr<ndn::util::scheduler::EventIdImpl> (empty) to 0x0,
  m_stragglerTimer = std::shared_ptr<ndn::util::scheduler::EventIdImpl> (empty) to 0x0,
  m_interest = std::shared_ptr<const ndn::Interest> (usecount 1, weakcount 1) to 0x1607d1d0 = {
    Use count = 1,
    Weak count = 1,
    Managed value = 0x1607d1d0
  },
  m_inRecords = empty std::list,
  m_outRecords = empty std::list,
  static LOCALHOST_NAME = {
    <std::enable_shared_from_this<ndn::Name>> = {
      _M_weak_this = std::weak_ptr<ndn::Name> (empty) to 0x0
    },
    members of ndn::Name:
    static npos = 18446744073709551615,
    m_nameBlock = {
      m_buffer = std::shared_ptr<const ndn::Buffer> (empty) to 0x0,
      m_type = 7,
      m_begin = <error reading variable>
  },
  static LOCALHOP_NAME = {
    <std::enable_shared_from_this<ndn::Name>> = {
      _M_weak_this = std::weak_ptr<ndn::Name> (empty) to 0x0
    },
    members of ndn::Name:
    static npos = 18446744073709551615,
    m_nameBlock = {
      m_buffer = std::shared_ptr<const ndn::Buffer> (empty) to 0x0,
      m_type = 7,
      m_begin = <error reading variable>
  },
  m_nameTreeEntry = std::shared_ptr<nfd::name_tree::Entry> (usecount 3, weakcount 1) to 0x9a6cd40 = {
    Use count = 3,
    Weak count = 1,
    Managed value = 0x9a6cd40
  }
}
```

For reference, the recorded interests

```
$2 = {
  <ndn::TagHost> = {
    m_tags = std::map with 1 elements = {
      [10] = std::shared_ptr<ndn::Tag> (usecount 1, weakcount 0) to 0x151e6670 = {
        Use count = 1,
        Weak count = 0,
        Managed value = 0x151e6670
```

```
    }
  }
},
<std::enable_shared_from_this<ndn::Interest>> = {
  _M_weak_this = std::weak_ptr<ndn::Interest> (usecount 1, weakcount 1) to 0x1607d1d0 = {
    Use count = 1,
    Weak count = 1,
    Managed value = 0x1607d1d0
  }
},
members of ndn::Interest:
m_name = "/ndn/edu/ucla/remap/ndnrtc/user/test/streams/camera/hi/delta/135017/data/%00%06",
m_selectors = {
  m_minSuffixComponents = -1,
  m_maxSuffixComponents = -1,
  m_publisherPublicKeyLocator = {
    m_type = ndn::KeyLocator::KeyLocator_None,
    m_name = ,
    m_keyDigest = {
      m_buffer = std::shared_ptr<const ndn::Buffer> (empty) to 0x0,
      m_type = 4294967295,
      m_begin = <error reading variable>,
    m_wire = {
      m_buffer = std::shared_ptr<const ndn::Buffer> (empty) to 0x0,
      m_type = 4294967295,
      m_begin = <error reading variable>
  },
  m_exclude = {
    m_exclude = std::map with 0 elements,
    m_wire = {
      m_buffer = std::shared_ptr<const ndn::Buffer> (empty) to 0x0,
      m_type = 4294967295,
      m_begin = <error reading variable>
  },
  m_childSelector = -1,
  m_mustBeFresh = true,
  m_wire = {
    m_buffer = std::shared_ptr<const ndn::Buffer> (usecount 23, weakcount 0) to 0x1b3beb80 = {
      Use count = 23,
      Weak count = 0,
      Managed value = 0x1b3beb80
    },
    m_type = 9,
    m_begin = 9 '\t',
    m_end = 10 '\n',
    m_size = 4,
    m_value_begin = 18 '\022',
    m_value_end = 10 '\n',
    m_subBlocks = std::vector of length 1, capacity 1 = {{
        m_buffer = std::shared_ptr<const ndn::Buffer> (usecount 23, weakcount 0) to 0x1b3beb80 = {
          Use count = 23,
          Weak count = 0,
          Managed value = 0x1b3beb80
        },
        m_type = 18,
        m_begin = 18 '\022',
        m_end = 10 '\n',
        m_size = 2,
        m_value_begin = 10 '\n',
        m_value_end = 10 '\n',
        m_subBlocks = std::vector of length 0, capacity 0
      }}
  }
},
m_nonce = {
  m_buffer = std::shared_ptr<const ndn::Buffer> (usecount 23, weakcount 0) to 0x1b3beb80 = {
    Use count = 23,
    Weak count = 0,
    Managed value = 0x1b3beb80
  },
  m_type = 10,
  m_begin = 10 '\n',
  m_end = 12 '\f',
  m_size = 6,
  m_value_begin = 31 '\037',
```

```
    m_value_end = 12 '\f',
    m_subBlocks = std::vector of length 0, capacity 0
  },
  m_interestLifetime = {
    rep_ = 2000
  },
  m_link = {
    m_buffer = std::shared_ptr<const ndn::Buffer> (empty) to 0x0,
    m_type = 4294967295,
    m_begin = <error reading variable>,
  m_linkCached = std::shared_ptr<ndn::Link> (empty) to 0x0,
  m_selectedDelegationIndex = 18446744073709551615,
  m_wire = {
    m_buffer = std::shared_ptr<const ndn::Buffer> (usecount 23, weakcount 0) to 0x1b3beb80 = {
      Use count = 23,
      Weak count = 0,
      Managed value = 0x1b3beb80
    },
    m_type = 5,
    m_begin = 5 '\005',
    m_end = 8 '\b',
    m_size = 107,
    m_value_begin = 7 '\a',
    m_value_end = 8 '\b',
    m_subBlocks = std::vector of length 4, capacity 4 = {{
        m_buffer = std::shared_ptr<const ndn::Buffer> (usecount 23, weakcount 0) to 0x1b3beb80 = {
          Use count = 23,
          Weak count = 0,
          Managed value = 0x1b3beb80
        },
        m_type = 7,
        m_begin = 7 '\a',
        m_end = 9 '\t',
        m_size = 91,
        m_value_begin = 8 '\b',
        m_value_end = 9 '\t',
        m_subBlocks = std::vector of length 0, capacity 0
      }, {
        m_buffer = std::shared_ptr<const ndn::Buffer> (usecount 23, weakcount 0) to 0x1b3beb80 = {
          Use count = 23,
          Weak count = 0,
          Managed value = 0x1b3beb80
        },
        m_type = 9,
        m_begin = 9 '\t',
        m_end = 10 '\n',
        m_size = 4,
        m_value_begin = 18 '\022',
        m_value_end = 10 '\n',
        m_subBlocks = std::vector of length 0, capacity 0
      }, {
        m_buffer = std::shared_ptr<const ndn::Buffer> (usecount 23, weakcount 0) to 0x1b3beb80 = {
          Use count = 23,
          Weak count = 0,
          Managed value = 0x1b3beb80
        },
        m_type = 10,
        m_begin = 10 '\n',
        m_end = 12 '\f',
        m_size = 6,
        m_value_begin = 31 '\037',
        m_value_end = 12 '\f',
        m_subBlocks = std::vector of length 0, capacity 0
      }, {
        m_buffer = std::shared_ptr<const ndn::Buffer> (usecount 23, weakcount 0) to 0x1b3beb80 = {
          Use count = 23,
          Weak count = 0,
          Managed value = 0x1b3beb80
        },
        m_type = 12,
        m_begin = 12 '\f',
        m_end = 8 '\b',
        m_size = 4,
        m_value_begin = 7 '\a',
        m_value_end = 8 '\b',
```

```
        m_subBlocks = std::vector of length 0, capacity 0
      }}
  }
}
```

#12 - 03/06/2016 10:23 AM - Junxiao Shi

- Status changed from New to In Progress

I propose the following solution:

- In **incoming Interest pipeline**, split duplicate Nonce detection to two steps.
  - Before PIT entry, DNL is queried. If DNL detects a duplicate Nonce, go straight to **Interest loop pipeline**.
    Setting straggler timer isn't helpful in this case, because straggler timer on a PIT entry can help with duplicate Nonce detection only if an in-record or out-record is created, but we can't create in-record otherwise Data would be mistakenly returned to the requester.
  - After PIT insert, PIT entry is queried for duplicate Nonce. If PIT entry contains a duplicate Nonce, this implies it's an old PIT entry with at least one in-record or out-record, and this condition should be asserted. Afterward, go to Interest loop pipeline as before.

- Change **Interest loop pipeline** to not require a PIT entry as parameter.

#13 - 03/06/2016 10:17 PM - Alex Afanasyev

The logic looks good to me.

#14 - 03/08/2016 04:52 AM - Junxiao Shi

- % Done changed from 0 to 30

- Estimated time set to 4.00 h

http://gerrit.named-data.net/2758

patchset1 has test case only.

#15 - 03/10/2016 03:43 AM - Junxiao Shi

- File forwarding-pipelines_20160310.pptx added

- % Done changed from 30 to 60

Pipelines design is updated.

NFD devguide is also updated in nfd-docs:commit:e5421bb278f4fa6c5ed628b584fe64697ce7f9e9.

#16 - 03/10/2016 03:56 AM - Junxiao Shi

- Status changed from In Progress to Code review

- % Done changed from 60 to 100

Implementation is updated as patchset2.

#17 - 03/10/2016 12:20 PM - Junxiao Shi

- Status changed from Code review to Closed

Files

2016-03-02 19.09.52.jpg	2.15 MB	03/03/2016	Junxiao Shi
forwarding-pipelines_20160310.pptx	92.4 KB	03/10/2016	Junxiao Shi