Monitoring Specification and Its Library Composition

Authors: Sebastian Robitzsch, Mays Al-Naday, Dirk Trossen

1. System Overview

1.1 Terminology and Workflow

The overall monitoring framework workflow is illustrated in Figure 1 and consists of the following entities:

- **Monitoring Database:** The environment to store collected monitoring data into a relational structure.
- **MOnitOring SErver (MOOSE):** The instance responsible for collecting reported monitoring data published by monitoring agents under implicitly known data points in the monitoring namespace and populate this information to the appropriate places in the monitoring database.
- **MONitoring Agent (MONA):** An instance which collects monitoring data from applications operating on the same node and publishes this information to the monitoring server under implicitly known data points using the monitoring namespace.
- **Application:** An instance which reports - if necessary aggregated - monitoring data to the MA which is eventually supposed to be reported to MS.
- **Blackader Core:** The ICN core framework handling pub/sub operations.
- **Blackadder API:** A set of predefined routines allowing an ICN application to perform pub/sub operations
- **MOnitoring LibrarY (MOLY):** A set of predefined routines to allow applications to send monitoring data to the monitoring agent
- **BlAckadder Monitoring wraPpER clasS (BAMPERS):** A set of routines for MAs to publish information under data points by using a more abstracted usage of the Blackadder API without caring about ICN related scope states.

Furthermore, Figure 1 illustrates the information flow from an application, agent and server point to view and where MOLY and BAMPERS come into play. While MOLY allows any application to report monitoring data to the monitoring agent on a particular node, BAMPERS is solely used by a monitoring agent to perform pub/sub operations without caring about any ICN related operation to exchange data with the monitoring server.

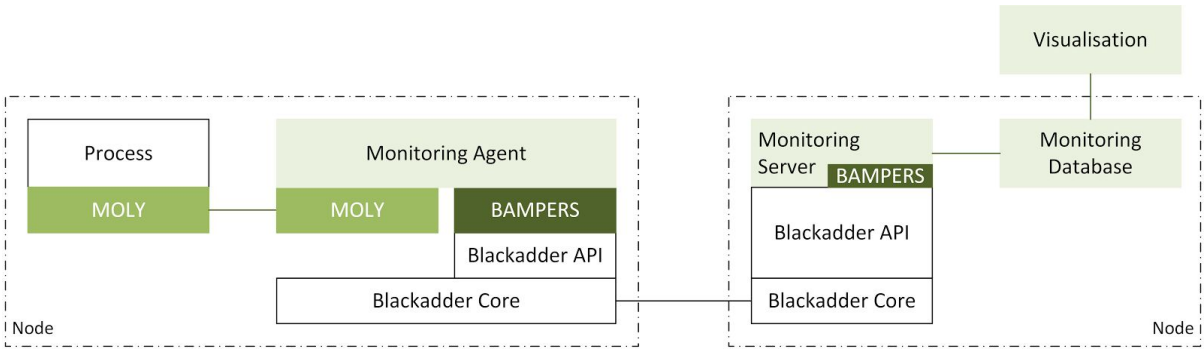

Figure 1.1: Overall workflow within the monitoring framework using MOLY and BAMPERS

1.3 Monitoring Namespace

The implemented monitoring namespace is given below which is reflected by the MOLY and BAMPERS primitives:

- /monitoring
  - /links
    - /linkId
      - /destinationNodeId
        - /linkType
          - /state
  - /nodes
    - /fn
      - /name
      - /state
    - /tm
      - /name
      - /state
    - /rv
      - /name
      - /state
    - /nap
      - /cmcGroupSizes
      - /name
      - /networkLatencyPerFqdn
      - /state
    - /gw
      - /cmcGroupSizes
      - /name
      - /networkLatencyPerFqdn
      - /state
    - /ue
      - /name
      - /state
    - /sv

- /name
- /state

1.4 Port Identifiers

The Port Identifier (PID) MA is listening on PID_MA, as provided in Table 1. All messages follow a TLV structure with the type and length encoded in the message header and the values in the payload field. A detailed list of all primitives and their content structure is provided in Section 2. All PIDs are enumerated in lib/blackadder_enums.hpp

Table 1: Port identifier of the monitoring agent

Name	Port Number	Description
PID_MA	39692	The port on which the monitoring agent is listening for datagram packets

1.5 Data Types

Types	Data Type	Description
LINK_ID	`string`	A unique string representation of a link. Note, this field always requires a LENGTH field beforehand indicating the length of the name. This length field is of data type uint32_t
NAME	`string`	User friendly name. Note, this field always requires a LENGTH field beforehand indicating the length of the name. This length field is of data type uint32_t
NODE_ID	`uint32_t`	A unique integer representation of a node
TYPE	`uint8_t`	The node type/role of a network element
STATE	`uint8_t`	The state of a network element

2. Bootstrapping

This section describes how applications, agents and the server bootstrap in order to allow a coordinated initilisation of the network elements that need to report monitoring data.

2.1 Introduction

When an application starts, it has no knowledge about the operational status of MA or MS. Furthermore, a coordinated bootstrapping of applications, MAs and MS is essential to not end up losing important monitoring data, .e.g., topology related node or link information. That is why MOLY aims at providing the bootstrapping feature as part of its initilisation process completely hidden from the application or agent. A detailed description of how to use MOLY is provided in Section 4.

2.2 Approach

The bootstrap of the monitoring network elements is divided into the initialisation of applications and their corresponding MA, described in Section 2.2.1, and the initialisation of MAs and the MS, described in Section 2.2.2. Only if both initialisations have been successfully completed, monitoring data can be reported from an application to MS.

2.2.1 Initialisation of Applications and their MA

Figure 2.1 illustrates the message sequence exchange between two applications and their monitoring agent. The steps are as follows:

1. Application 1 sends its PID on which it listens for messages from the monitoring agent to the monitoring agent encapsulated in a BOOTSTRAP_MY_PID using message type BOOTSTRAP_MY_PID
2. The monitoring agent acknowledges the correct reception of the message under Step 1 and confirms that it has added Application 1 as a process which waits to receive a trigger message using the uniform BOOTSTRAP_OK message type.
3. Application 2 sends its PID encapsulated in a BOOTSTRAP_MY_PID to the monitoring agent, similar to Application 1 in Step 1.
4. The monitoring agent acknowledges the correct reception of the message from Application 2 and confirms it has added Application 2 to the list of applications which awaits a trigger message using the uniform BOOTSTRAP_OK message type.
5. The monitoring agent has learned about its node ID through the management interface towards the ICN core node (see Section 5 for more details) and has received the trigger from the monitoring server that it can be used to report monitoring data (see Section 2.2.2 for more details).
6. The monitoring agent sends a START_REPORTING trigger message to all registered applications using their previously announced PIDs on which they are listening on.

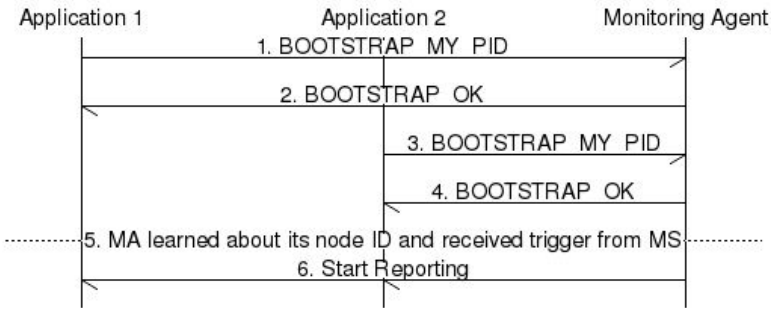

Figure 2.1: Bootstrapping of applications and their MA

2.2.2 Initialisation of MAs and their MS

Figure 2.2 illustrates the message exchange between the monitoring server and all agents which are active at the time the monitoring server starts. This communication is realised using ICN semantics. The individual steps are as follows:

1. The monitoring agent subscribes to a predefined information item /monitoring under the father scoop /management[1].
2. Once the monitoring server has initialised a connection towards the database and is ready to receive and insert reported monitoring data, it awaits a START_PUBLISH from the ICN core node which triggers a BOOTSTRAP_SERVER_UP trigger messages being published to all monitoring agents under /management/monitoring.

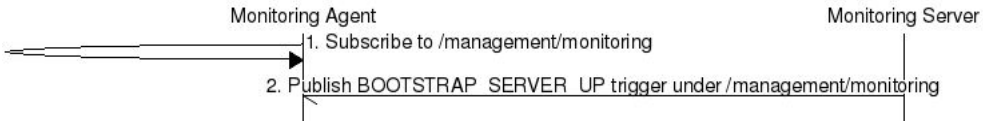

Figure 2.2: MS informs all active MAs about its availability

Figure 2.3 illustrates the case where a new node is being attached to an already operational network where MS is fully operational (sequences of Figure 2.2 have been completed). In this case the bootstrapping is as follows:

1. TM is reporting the attachment of a new node to MS using ADD_NODE().
2. MS publishes the BOOTSTRAP_SERVER_UP command under /management/monitoring information MA on the newly attached node that it can start reporting data.

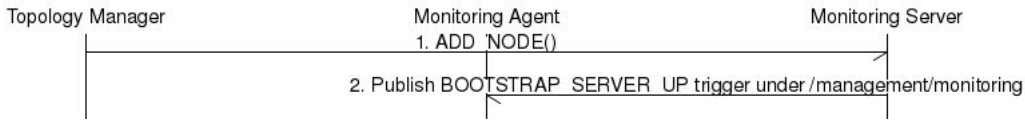

Figure 2.3: MS informs a newly attached MA about its availability

[1] The enumeration of root scope identifiers has not been agreed yet. The tentative implementation of Blackadder uses an enumeration defined in ~/blackadder/lib/moly/enum.hh or ~/blackadder/lib/blackadder_enums.hpp depending on the release candidate.

2.3 Bootstrapping Message Types

The following table lists the enumerated message types used to bootstrap MOLY applications and their agent.

Message Type	**Value**
BOOTSTRAP_OK	0
BOOTSTRAP_ERROR	1
BOOTSTRAP_MY_PID	2
BOOTSTRAP_START_REPORTING	3
BOOTSTRAP_SERVER_UP	4

3. MOnitoring LibrarY (MOLY)

3.1 Enumerations

3.1.1 Link Types

The link types are declared in lib/moly/enum.hh

Name	Value	Description
LINK_TYPE_802_3	1	
LINK_TYPE_802_11	2	
LINK_TYPE_802_11_A	3	
LINK_TYPE_802_11_B	4	
LINK_TYPE_802_11_G	5	
LINK_TYPE_802_11_N	6	
LINK_TYPE_802_11_AA	7	
LINK_TYPE_802_11_AC	8	
LINK_TYPE_SDN_802_3_Z	9	IEEE spec deployed by pica8 SDN switches for 1G ports
LINK_TYPE_SDN_802_3_AE	10	IEEE spec deployed by pica8 SDN switches for 10G optical ports
LINK_TYPE_GPRS	11	
LINK_TYPE_UMTS	12	
LINK_TYPE_LTE	13	
LINK_TYPE_LTE_A	14	
LINK_TYPE_OPTICAL	15	
LINK_TYPE_UNKNOWN	16	

3.1.3 Node Types

The states of node types are declared in lib/moly/enum.hh

Name	Value	Description
NODE_TYPE_UNKNOWN	0	The node's role is unknown
NODE_TYPE_GW	1	The node role is ICN gateway
NODE_TYPE_FN	2	The node role is forwarding node
NODE_TYPE_NAP	3	The node role is network attachment point
NODE_TYPE_RV	4	The node role is rendezvous
NODE_TYPE_SERVER	5	The node role is server
NODE_TYPE_TM	6	The node role is topology manager
NODE_TYPE_UE	7	The node role is user equipment

3.1.4 States

The states of network elements are declared in lib/moly/enum.hh

Name	Value	Description
STATE_UNKNOWN	0	The state of a network element is unknown
STATE_BOOTED	1	The state of a network element is booted
STATE_DOWN	4	The state of a network element is down
STATE_UP	8	The state of a network element is up

3.2 Primitives

ADD_LINK

When generated: This message is generated by the TM if a new link has been added to the topology.

Field	Type	Description
linkName	NAME	User friendly name of the link
linkId	LINK_ID	The identifier of a particular link
sourceNodeId	NODE_ID	The node identifier of a source network element

destinationNodeId	NODE_ID	The node identifier of a destination network element
linkType	TYPE	The link type of the reported link

Action upon arrival: The monitoring agent publishes the new scope path into the domain local RV (if not done so) and publishes the information towards the monitoring server under the corresponding monitoring namespace.

ADD_NODE

When generated: This message is generated by an application if it boots up and reports its existence.

Field	Type	Description
name	NAME	User friendly name of the node
nodeId	NODE_ID	The node identifier of the network element
nodeType	TYPE	The node type of the reported node

Action upon arrival: The monitoring agent publishes the new scope path into the domain local RV (if not done so) and publishes the information towards the monitoring server under the corresponding monitoring namespace.

ADD_NODE_TYPE

When generated: This message is generated by an application if it boots up and wants to report its existence. Note, this primitive is almost identical to ADD_NODE without the nodeId field though. Only applications which do not know their node ID should use ADD_NODE_TYPE.

Field	Type	Description
name	NAME	User friendly name of the node
nodeType	TYPE	The node type of the reported node

Action upon arrival: The monitoring agent publishes the new scope path into the domain local RV (if not done so) and publishes the information towards the monitoring server under the corresponding monitoring namespace.

CMC_GROUP_SIZE

When generated: This message is issued by the NAP application when reporting the average coincidental multicast group size over the time interval when the last report was sent. The CMC group size is averaged over the predefined reporting time interval.

Field	Type	Description
groupSize	`uint32_t`	The coincidental multicast group size

Action upon arrival: The monitoring agent publishes the new scope path into the domain local RV (if not done so) and publishes the information towards the monitoring server under the corresponding monitoring namespace:

/monitoring/topology/nodes/nap/nodeId/coincidentalMulticastGroupSize

The required node ID is known to the monitoring agent prior to the arrival of this primitive.

HTTP_REQUESTS_PER_FQDN

When generated: This message is issued by a NAP application when reporting the respective data point. The number of HTTP requests is reported over the predefined reporting time interval.

Field	Type	Description
fqdnLength	`uint32_t`	Length of the next field “fqdn”
fqdn	`string`	The FQDN for which the number of HTTP requests traversed the node
numberOfRequests	`uint32_t`	The number of requests that traversed the node

Action upon arrival: The monitoring agent publishes the scope path to the domain local RV (if not done so) and publishes the data under CID

/monitoring/topology/nodes/nap/nodeId/httpRequestsPerFqdn

or

/monitoring/topology/nodes/gw/nodeId/httpRequestsPerFqdn

depending on the node type using BAMPERS.

LINK_STATE

When generated: This message is generated by an application if the state of a link has changed.

Field	Type	Description
destinationNodeId	`NODE_ID`	The node identifier of a destination network element
linkType	`TYPE`	The link type of the reported link
state	`STATE`	The new state of the reported link

NETWORK_LATENCY_PER_FQDN

When generated: This message is issued as part of the periodic reporting of statistics by a NAP. If the network latency for a particular FQDN could not be measured within the reporting period this message is not generated. Note that only cNAPs report the network latency.

Field	Type	Description
hashedFqdn	`uint32_t`	The FQDN (hashed) for which the average network latency is being reported
networkLatency	`uint16_t`	The ICN network latency measured by the NAP

3.3 Using MOLY

As illustrated in Section 1.1, MOLY provides the required set of primitives to exchange monitoring information between applications and their monitoring agent.

3.3.1 Initialisation

The initialisation process of how the MOLY library bootstraps distinguishes between applications and agents.

3.3.1.1 Application

3.3.1.2 Agent

3.3.2 Primitives

3.3.2.1 Application

3.3.2.2 Agent

3.3.3 Examples

3.3.3.1 Application

3.3.3.2 Agent

4. BlAckadder Monitoring wraPpER clasS (BAMPERS)

Primitives

ADD_LINK

Field	Type	Description
epoch	`uint32_t`	The time since 1 January 1970
sourceNodeId	`NODE_ID`	The node identifier of a source network element

ADD_NODE

Field	Type	Description
epoch	`uint32_t`	The time since 1 January 1970
nameLength	`uint32_t`	The length of the node name given in the field “name”
name	`NAME`	User friendly name of the node

CMC_GROUP_SIZE

Field	Type	Description
epoch	`uint32_t`	The time since 1 January 1970
groupSize	`uint32_t`	The coincidental multicast group size

HTTP_REQUESTS_PER_FQDN

Field	Type	Description
epoch	`uint32_t`	The time since 1 January 1970
fqdnLength	`uint32_t`	Length of the next field “fqdn”
fqdn	`string`	The FQDN for which the number of HTTP

		requests traversed the node
numberOfRequests	`uint32_t`	The number of requests that traversed the node

LINK_STATE

When generated: The monitoring agent publishes the new scope path into the domain local RV (if not done so) and publishes the information towards the monitoring server under the corresponding monitoring namespace:

/monitoring/topology/links/linkId/destinationNodeId/linkType/state

Field	Type	Description
epoch	`uint32_t`	The time since 1 January 1970
state	`uin8_t`	The new state of the reported link

NETWORK_LATENCY_PER_FQDN

When generated: MONA publishes the scope path to the domain local RV (if not done so) and publishes the data under CID

/monitoring/topology/nodes/nap/nodeId/networkLatencyPerFqdn

or

/monitoring/topology/nodes/gw/nodeId/networkLatencyPerFqdn

depending on the node type using BAMPERS.

Field	Type	Description
epoch	`uint32_t`	The time since 1 January 1970
hashedFqdn	`uint32_t`	The FQDN (hashed) for which the average network latency is being reported
networkLatency	`uint16_t`	The ICN network latency measured by the NAP