MODULE *Protocols*

LOCAL INSTANCE *Naturals*

LOCAL INSTANCE *Sequences*

LOCAL INSTANCE *FiniteSets*

LOCAL INSTANCE *TLC*

MODULE *E2AP*

The *E2AP* module provides a formal specification of the *E2AP* protocol. The spec defines the client and server interfaces for *E2AP* and provides helpers for managing and operating on connections.

CONSTANT *Nil*

VARIABLE *servers*, *conns*

The *E2AP* protocol is implemented on *SCTP*

LOCAL $SCTP \triangleq$ INSTANCE *SCTP*

$vars \triangleq \langle servers, conns \rangle$

Message type constants

CONSTANTS
E2SetupRequestType,
E2SetupResponseType,
E2SetupFailureType

CONSTANTS
ResetRequestType,
ResetResponseType

CONSTANTS
RICSubscriptionRequestType,
RICSubscriptionResponseType,
RICSubscriptionFailureType

CONSTANTS
RICSubscriptionDeleteRequestType,
RICSubscriptionDeleteResponseType,
RICSubscriptionDeleteFailureType

CONSTANTS
RICControlRequestType,
RICControlResponseType,
RICControlFailureType,
RICServiceUpdateType

CONSTANTS
E2ConnectionUpdateType,
E2ConnectionUpdateAcknowledgeType,

$E2ConnectionUpdateFailureType$

CONSTANTS

$E2NodeConfigurationUpdateType$,
$E2NodeConfigurationUpdateAcknowledgeType$,
$E2NodeConfigurationUpdateFailureType$

LOCAL $messageTypes \triangleq$

$\{E2SetupRequestType$,
$E2SetupResponseType$,
$E2SetupFailureType$,
$ResetRequestType$,
$ResetResponseType$,
$RICSubscriptionRequestType$,
$RICSubscriptionResponseType$,
$RICSubscriptionFailureType$,
$RICSubscriptionDeleteRequestType$,
$RICSubscriptionDeleteResponseType$,
$RICSubscriptionDeleteFailureType$,
$RICControlRequestType$,
$RICControlResponseType$,
$RICControlFailureType$,
$RICServiceUpdateType$,
$E2ConnectionUpdateType$,
$E2ConnectionUpdateAcknowledgeType$,
$E2ConnectionUpdateFailureType$,
$E2NodeConfigurationUpdateType$,
$E2NodeConfigurationUpdateAcknowledgeType$,
$E2NodeConfigurationUpdateFailureType\}$

Message types should be defined as strings to simplify debugging

ASSUME $\forall\, m \in messageTypes : m \in$ STRING

Failure cause constants

CONSTANTS

$MiscFailureUnspecified$,
$MiscFailureControlProcessingOverload$,
$MiscFailureHardwareFailure$,
$MiscFailureOMIntervention$

CONSTANTS

$ProtocolFailureUnspecified$,
$ProtocolFailureTransferSyntaxError$,
$ProtocolFailureAbstractSyntaxErrorReject$,
$ProtocolFailureAbstractSyntaxErrorIgnoreAndNotify$,
$ProtocolFailureMessageNotCompatibleWithReceiverState$,
$ProtocolFailureSemanticError$,
$ProtocolFailureAbstractSyntaxErrorFalselyConstructedMessage$

CONSTANTS
RICFailureUnspecified,
RICFailureRANFunctionIDInvalid,
RICFailureActionNotSupported,
RICFailureExcessiveActions,
RICFailureDuplicateAction,
RICFailureDuplicateEvent,
RICFailureFunctionResourceLimit,
RICFailureRequestIDUnknown,
RICFailureInconsistentActionSubsequentActionSequence,
RICFailureControlMessageInvalid,
RICFailureCallProcessIDInvalid

CONSTANTS
RICServiceFailureUnspecified,
RICServiceFailureFunctionNotRequired,
RICServiceFailureExcessiveFunctions,
RICServiceFailureRICResourceLimit

CONSTANTS
TransportFailureUnspecified,
TransportFailureTransportResourceUnavailable

LOCAL $failureCauses \triangleq$
$\{$*MiscFailureUnspecified*,
MiscFailureControlProcessingOverload,
MiscFailureHardwareFailure,
MiscFailureOMIntervention,
ProtocolFailureUnspecified,
ProtocolFailureTransferSyntaxError,
ProtocolFailureAbstractSyntaxErrorReject,
ProtocolFailureAbstractSyntaxErrorIgnoreAndNotify,
ProtocolFailureMessageNotCompatibleWithReceiverState,
ProtocolFailureSemanticError,
ProtocolFailureAbstractSyntaxErrorFalselyConstructedMessage,
RICFailureUnspecified,
RICFailureRANFunctionIDInvalid,
RICFailureActionNotSupported,
RICFailureExcessiveActions,
RICFailureDuplicateAction,
RICFailureDuplicateEvent,
RICFailureFunctionResourceLimit,
RICFailureRequestIDUnknown,
RICFailureInconsistentActionSubsequentActionSequence,
RICFailureControlMessageInvalid,
RICFailureCallProcessIDInvalid,
RICServiceFailureUnspecified,

RICServiceFailureFunctionNotRequired,
RICServiceFailureExcessiveFunctions,
RICServiceFailureRICResourceLimit,
TransportFailureUnspecified,
TransportFailureTransportResourceUnavailable}

Failure causes should be defined as strings to simplify debugging
ASSUME $\forall\, c \in failureCauses : c \in \text{STRING}$

MODULE *Messages*

The *Messages* module defines predicates for receiving, sending, and verifying all the messages supported by *E2AP*.

This section defines predicates for identifying *E2AP* message types on the network.

$IsE2SetupRequest(m) \triangleq m.type = E2SetupRequestType$

$IsE2SetupResponse(m) \triangleq m.type = E2SetupResponseType$

$IsE2SetupFailure(m) \triangleq m.type = E2SetupFailureType$

$IsResetRequest(m) \triangleq m.type = ResetRequestType$

$IsResetResponse(m) \triangleq m.type = ResetResponseType$

$IsRICSubscriptionRequest(m) \triangleq m.type = RICSubscriptionRequestType$

$IsRICSubscriptionResponse(m) \triangleq m.type = RICSubscriptionResponseType$

$IsRICSubscriptionFailure(m) \triangleq m.type = RICSubscriptionFailureType$

$IsRICSubscriptionDeleteRequest(m) \triangleq m.type = RICSubscriptionDeleteRequestType$

$IsRICSubscriptionDeleteResponse(m) \triangleq m.type = RICSubscriptionDeleteResponseType$

$IsRICSubscriptionDeleteFailure(m) \triangleq m.type = RICSubscriptionDeleteFailureType$

$IsRICControlRequest(m) \triangleq m.type = RICControlRequestType$

$IsRICControlResponse(m) \triangleq m.type = RICControlResponseType$

$IsRICControlFailure(m) \triangleq m.type = RICControlFailureType$

$IsRICServiceUpdate(m) \triangleq m.type = RICServiceUpdateType$

$IsE2ConnectionUpdate(m) \triangleq m.type = E2ConnectionUpdateType$

$IsE2ConnectionUpdateAcknowledge(m) \triangleq m.type = E2ConnectionUpdateAcknowledgeType$

$IsE2ConnectionUpdateFailure(m) \triangleq m.type = E2ConnectionUpdateFailureType$

$IsE2NodeConfigurationUpdate(m) \triangleq m.type = E2NodeConfigurationUpdateType$

$IsE2NodeConfigurationUpdateAcknowledge(m) \triangleq m.type = E2NodeConfigurationUpdateAcknowledgeTy$

$IsE2NodeConfigurationUpdateFailure(m) \triangleq m.type = E2NodeConfigurationUpdateFailureType$

This section defines predicates for validating $E2AP$ message contents. The predicates provide precise documentation on the $E2AP$ message format and are used within the spec to verify that steps adhere to the $E2AP$ protocol specification.

LOCAL $ValidE2SetupRequest(m) \triangleq$ TRUE

LOCAL $ValidE2SetupResponse(m) \triangleq$ TRUE

LOCAL $ValidE2SetupFailure(m) \triangleq$ TRUE

LOCAL $ValidResetRequest(m) \triangleq$ TRUE

LOCAL $ValidResetResponse(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionRequest(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionResponse(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionFailure(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionDeleteRequest(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionDeleteResponse(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionDeleteFailure(m) \triangleq$ TRUE

LOCAL $ValidRICControlRequest(m) \triangleq$ TRUE

LOCAL $ValidRICControlResponse(m) \triangleq$ TRUE

LOCAL $ValidRICControlFailure(m) \triangleq$ TRUE

LOCAL $ValidRICServiceUpdate(m) \triangleq$ TRUE

LOCAL $ValidE2ConnectionUpdate(m) \triangleq$ TRUE

LOCAL $ValidE2ConnectionUpdateAcknowledge(m) \triangleq$ TRUE

LOCAL $ValidE2ConnectionUpdateFailure(m) \triangleq$ TRUE

LOCAL $ValidE2NodeConfigurationUpdate(m) \triangleq$ TRUE

LOCAL $ValidE2NodeConfigurationUpdateAcknowledge(m) \triangleq$ TRUE

LOCAL $ValidE2NodeConfigurationUpdateFailure(m) \triangleq$ TRUE

This section defines operators for constructing $E2AP$ messages.

LOCAL $SetType(m,\ t) \triangleq [m \text{ EXCEPT } !.type = t]$

$E2SetupRequest(m) \triangleq$
IF $Assert(ValidE2SetupRequest(m),$ "Invalid E2SetupRequest")
THEN $SetType(m,\ E2SetupRequestType)$
ELSE Nil

$E2SetupResponse(m) \triangleq$
IF $Assert(ValidE2SetupResponse(m),$ "Invalid E2SetupResponse")
THEN $SetType(m,\ E2SetupResponseType)$
ELSE Nil

$E2SetupFailure(m) \triangleq$
IF $Assert(ValidE2SetupFailure(m),$ "Invalid E2SetupFailure")
THEN $SetType(m,\ E2SetupFailureType)$
ELSE Nil

$ResetRequest(m) \triangleq$
IF $Assert(ValidResetRequest(m),$ "Invalid ResetRequest")
THEN $SetType(m,\ ResetRequestType)$
ELSE Nil

$ResetResponse(m) \triangleq$
IF $Assert(ValidResetResponse(m),$ "Invalid ResetResponse")
THEN $SetType(m,\ ResetResponseType)$
ELSE Nil

$RICSubscriptionRequest(m) \triangleq$
IF $Assert(ValidRICSubscriptionRequest(m),$ "Invalid RICSubscriptionRequest")
THEN $SetType(m,\ RICSubscriptionRequestType)$
ELSE Nil

$RICSubscriptionResponse(m) \triangleq$
IF $Assert(ValidRICSubscriptionResponse(m),$ "Invalid RICSubscriptionResponse")
THEN $SetType(m,\ RICSubscriptionResponseType)$
ELSE Nil

$RICSubscriptionFailure(m) \triangleq$
IF $Assert(ValidRICSubscriptionFailure(m),$ "Invalid RICSubscriptionFailure")
THEN $SetType(m,\ RICSubscriptionFailureType)$
ELSE Nil

$RICSubscriptionDeleteRequest(m) \triangleq$
IF $Assert(ValidRICSubscriptionDeleteRequest(m),$ "Invalid RICSubscriptionDeleteRequest")
THEN $SetType(m,\ RICSubscriptionDeleteRequestType)$

ELSE Nil

$RICSubscriptionDeleteResponse(m) \triangleq$
IF $Assert(ValidRICSubscriptionDeleteResponse(m),$ "Invalid RICSubscriptionDeleteResponse")
THEN $SetType(m,\ RICSubscriptionDeleteResponseType)$
ELSE Nil

$RICSubscriptionDeleteFailure(m) \triangleq$
IF $Assert(ValidRICSubscriptionDeleteFailure(m),$ "Invalid RICSubscriptionDeleteFailure")
THEN $SetType(m,\ RICSubscriptionDeleteFailureType)$
ELSE Nil

$RICControlRequest(m) \triangleq$
IF $Assert(ValidRICControlRequest(m),$ "Invalid RICControlRequest")
THEN $SetType(m,\ RICControlRequestType)$
ELSE Nil

$RICControlResponse(m) \triangleq$
IF $Assert(ValidRICControlResponse(m),$ "Invalid RICControlResponse")
THEN $SetType(m,\ RICControlResponseType)$
ELSE Nil

$RICControlFailure(m) \triangleq$
IF $Assert(ValidRICControlFailure(m),$ "Invalid RICControlFailure")
THEN $SetType(m,\ RICControlFailureType)$
ELSE Nil

$RICServiceUpdate(m) \triangleq$
IF $Assert(ValidRICServiceUpdate(m),$ "Invalid RICServiceUpdate")
THEN $SetType(m,\ RICServiceUpdateType)$
ELSE Nil

$E2ConnectionUpdate(m) \triangleq$
IF $Assert(ValidE2ConnectionUpdate(m),$ "Invalid E2ConnectionUpdate")
THEN $SetType(m,\ E2ConnectionUpdateType)$
ELSE Nil

$E2ConnectionUpdateAcknowledge(m) \triangleq$
IF $Assert(ValidE2ConnectionUpdateAcknowledge(m),$ "Invalid E2ConnectionUpdateAcknowledge")
THEN $SetType(m,\ E2ConnectionUpdateAcknowledgeType)$
ELSE Nil

$E2ConnectionUpdateFailure(m) \triangleq$
IF $Assert(ValidE2ConnectionUpdateFailure(m),$ "Invalid E2ConnectionUpdateFailure")
THEN $SetType(m,\ E2ConnectionUpdateFailureType)$
ELSE Nil

$E2NodeConfigurationUpdate(m) \triangleq$

IF $Assert(ValidE2NodeConfigurationUpdate(m),$ "Invalid E2NodeConfigurationUpdate")
THEN $SetType(m,\ E2NodeConfigurationUpdateType)$
ELSE Nil

$E2NodeConfigurationUpdateAcknowledge(m) \triangleq$
IF $Assert(ValidE2NodeConfigurationUpdateAcknowledge(m),$ "Invalid E2NodeConfigurationUpdateAck
THEN $SetType(m,\ E2NodeConfigurationUpdateAcknowledgeType)$
ELSE Nil

$E2NodeConfigurationUpdateFailure(m) \triangleq$
IF $Assert(ValidE2NodeConfigurationUpdateFailure(m),$ "Invalid E2NodeConfigurationUpdateFailure")
THEN $SetType(m,\ E2NodeConfigurationUpdateFailureType)$
ELSE Nil

The *Messages* module is instantiated locally to avoid access from outside the module.

LOCAL $Messages \triangleq$ INSTANCE $Messages$

MODULE *Client*

The *Client* module provides operators for managing and operating on $E2AP$ client connections and specifies the message types supported for the client.

MODULE *Send*

This module provides message type operators for the message types that can be send by the $E2AP$ client.

$E2SetupRequest(c,\ m) \triangleq$
$\land\ SCTP!Client!Send(c,\ Messages!E2SetupResponse(m))$

$ResetRequest(c,\ m) \triangleq$
$\land\ SCTP!Client!Send(c,\ Messages!ResetRequest(m))$

$ResetResponse(c,\ m) \triangleq$
$\land\ SCTP!Client!Reply(c,\ Messages!ResetResponse(m))$

$Send \triangleq$ INSTANCE $Send$

MODULE *Receive*

This module provides predicates for the types of messages that can be received by an $E2AP$ client.

$E2SetupResponse(c,\ h(\_,\ \_)) \triangleq$
$SCTP!Server!Handle(c,$ LAMBDA $x,\ m :$
$\land\ Messages!IsE2SetupResponse(m)$

$\land\ SCTP!Client!Receive(c)$
$\land\ h(c,\ m))$

$ResetRequest(c,\ h(\_,\ \_)) \triangleq$
$SCTP!Server!Handle(c,$ LAMBDA $x,\ m :$
$\land\ Messages!IsResetRequest(m)$
$\land\ SCTP!Client!Receive(c)$
$\land\ h(c,\ m))$

$ResetResponse(c,\ h(\_,\ \_)) \triangleq$
$SCTP!Server!Handle(c,$ LAMBDA $x,\ m :$
$\land\ Messages!IsResetResponse(m)$
$\land\ SCTP!Client!Receive(c)$
$\land\ h(c,\ m))$

Instantiate the $E2AP!Client!Receive$ module
$Receive \triangleq$ INSTANCE $Receive$

$Connect(s,\ d) \triangleq SCTP!Client!Connect(s,\ d)$

$Disconnect(c) \triangleq SCTP!Client!Disconnect(c)$

Provides operators for the $E2AP$ client
$Client \triangleq$ INSTANCE $Client$

MODULE $Server$

The $Server$ module provides operators for managing and operating on $E2AP$ servers and specifies the message types supported for the server.

MODULE $Send$

This module provides message type operators for the message types that can be send by the $E2AP$ server.

$E2SetupResponse(c,\ m) \triangleq$
$\land\ SCTP!Server!Reply(c,\ Messages!E2SetupResponse(m))$

$ResetRequest(c,\ m) \triangleq$
$\land\ SCTP!Server!Send(c,\ Messages!ResetRequest(m))$

$ResetResponse(c,\ m) \triangleq$
$\land\ SCTP!Server!Reply(c,\ Messages!ResetResponse(m))$

Instantiate the $E2AP!Server!Send$ module
$Send \triangleq$ INSTANCE $Send$

MODULE *Receive*

This module provides predicates for the types of messages that can be received by an $E2AP$ server.

$E2SetupRequest(c,\ h(\_,\ \_)) \triangleq$
  $SCTP!Server!Handle(c,$ LAMBDA $x,\ m :$
    $\land\ Messages!IsE2SetupRequest(m)$
    $\land\ SCTP!Server!Receive(c)$
    $\land\ h(c,\ m))$

$ResetRequest(c,\ h(\_,\ \_)) \triangleq$
  $SCTP!Server!Handle(c,$ LAMBDA $x,\ m :$
    $\land\ Messages!IsResetRequest(m)$
    $\land\ SCTP!Server!Receive(c)$
    $\land\ h(c,\ m))$

$ResetResponse(c,\ h(\_,\ \_)) \triangleq$
  $SCTP!Server!Handle(c,$ LAMBDA $x,\ m :$
    $\land\ Messages!IsResetResponse(m)$
    $\land\ SCTP!Server!Receive(c)$
    $\land\ h(c,\ m))$

Instantiate the $E2AP!Server!Receive$ module
$Receive \triangleq$ INSTANCE $Receive$

Starts a new $E2AP$ server
$Serve(s) \triangleq SCTP!Server!Start(s)$

Stops the given $E2AP$ server
$Stop(s) \triangleq SCTP!Server!Stop(s)$

Provides operators for the $E2AP$ server
$Server \triangleq$ INSTANCE $Server$

The set of all running $E2AP$ servers
$Servers \triangleq SCTP!Servers$

The set of all open $E2AP$ connections
$Connections \triangleq SCTP!Connections$

$Init \triangleq SCTP!Init$

$Next \triangleq SCTP!Next$

VARIABLES $e2apServers$, $e2apConns$

$E2AP \triangleq$ INSTANCE $E2AP$ WITH
$servers \leftarrow e2apServers$,
$conns \leftarrow e2apConns$,
$Nil \leftarrow [type \mapsto \text{“”}]$,
$E2SetupRequestType \leftarrow$ “E2SetupRequest”,
$E2SetupResponseType \leftarrow$ “E2SetupResponse”,
$E2SetupFailureType \leftarrow$ “E2SetupFailure”,
$ResetRequestType \leftarrow$ “ResetRequest”,
$ResetResponseType \leftarrow$ “ResetResponse”,
$RICSubscriptionRequestType \leftarrow$ “RICSubscriptionRequest”,
$RICSubscriptionResponseType \leftarrow$ “RICSubscriptionResponse”,
$RICSubscriptionFailureType \leftarrow$ “RICSubscriptionFailure”,
$RICSubscriptionDeleteRequestType \leftarrow$ “RICSubscriptionDeleteRequest”,
$RICSubscriptionDeleteResponseType \leftarrow$ “RICSubscriptionDeleteResponse”,
$RICSubscriptionDeleteFailureType \leftarrow$ “RICSubscriptionDeleteFailure”,
$RICControlRequestType \leftarrow$ “RICControlRequest”,
$RICControlResponseType \leftarrow$ “RICControlResponse”,
$RICControlFailureType \leftarrow$ “RICControlFailure”,
$RICServiceUpdateType \leftarrow$ “RICServiceUpdate”,
$E2ConnectionUpdateType \leftarrow$ “E2ConnectionUpdate”,
$E2ConnectionUpdateAcknowledgeType \leftarrow$ “E2ConnectionUpdateAcknowledge”,
$E2ConnectionUpdateFailureType \leftarrow$ “E2ConnectionUpdateFailure”,
$E2NodeConfigurationUpdateType \leftarrow$ “E2NodeConfigurationUpdate”,
$E2NodeConfigurationUpdateAcknowledgeType \leftarrow$ “E2NodeConfigurationUpdateAcknowledge”,
$E2NodeConfigurationUpdateFailureType \leftarrow$ “E2NodeConfigurationUpdateFailure”,
$MiscFailureUnspecified \leftarrow$ “MiscFailureUnspecified”,
$MiscFailureControlProcessingOverload \leftarrow$ “MiscFailureControlProcessingOverload”,
$MiscFailureHardwareFailure \leftarrow$ “MiscFailureHardwareFailure”,
$MiscFailureOMIntervention \leftarrow$ “MiscFailureOMIntervention”,
$ProtocolFailureUnspecified \leftarrow$ “ProtocolFailureUnspecified”,
$ProtocolFailureTransferSyntaxError \leftarrow$ “ProtocolFailureTransferSyntaxError”,
$ProtocolFailureAbstractSyntaxErrorReject \leftarrow$ “ProtocolFailureAbstractSyntaxErrorReject”,
$ProtocolFailureAbstractSyntaxErrorIgnoreAndNotify \leftarrow$ “ProtocolFailureAbstractSyntaxErrorIgnoreAndNotify”
$ProtocolFailureMessageNotCompatibleWithReceiverState \leftarrow$ “ProtocolFailureMessageNotCompatibleWithRece
$ProtocolFailureSemanticError \leftarrow$ “ProtocolFailureSemanticError”,
$ProtocolFailureAbstractSyntaxErrorFalselyConstructedMessage \leftarrow$ “ProtocolFailureAbstractSyntaxErrorFalsely
$RICFailureUnspecified \leftarrow$ “RICFailureUnspecified”,
$RICFailureRANFunctionIDInvalid \leftarrow$ “RICFailureRANFunctionIDInvalid”,
$RICFailureActionNotSupported \leftarrow$ “RICFailureActionNotSupported”,
$RICFailureExcessiveActions \leftarrow$ “RICFailureExcessiveActions”,
$RICFailureDuplicateAction \leftarrow$ “RICFailureDuplicateAction”,
$RICFailureDuplicateEvent \leftarrow$ “RICFailureDuplicateEvent”,
$RICFailureFunctionResourceLimit \leftarrow$ “RICFailureFunctionResourceLimit”,

RICFailureRequestIDUnknown ← "RICFailureRequestIDUnknown",
RICFailureInconsistentActionSubsequentActionSequence ← "RICFailureInconsistentActionSubsequentActionSe
RICFailureControlMessageInvalid ← "RICFailureControlMessageInvalid",
RICFailureCallProcessIDInvalid ← "RICFailureCallProcessIDInvalid",
RICServiceFailureUnspecified ← "RICServiceFailureUnspecified",
RICServiceFailureFunctionNotRequired ← "RICServiceFailureFunctionNotRequired",
RICServiceFailureExcessiveFunctions ← "RICServiceFailureExcessiveFunctions",
RICServiceFailureRICResourceLimit ← "RICServiceFailureRICResourceLimit",
TransportFailureUnspecified ← "TransportFailureUnspecified",
TransportFailureTransportResourceUnavailable ← "TransportFailureTransportResourceUnavailable"

MODULE *E2TService*

The *E2AP* module provides a formal specification of the *E2T* service. The spec defines the client and server interfaces for *E2T* and provides helpers for managing and operating on connections.

CONSTANT *Nil*

VARIABLE *servers*, *conns*

The *E2T API* is implemented as a *gRPC* service

LOCAL *GRPC* $\triangleq$ INSTANCE *GRPC*

$vars \triangleq \langle servers, conns \rangle$

Message type constants

CONSTANT
SubscribeRequestType,
SubscribeResponseType
CONSTANTS
UnsubscribeRequestType,
UnsubscribeResponseType
CONSTANTS
ControlRequestType,
ControlResponseType

LOCAL $messageTypes \triangleq$
{*SubscribeRequestType*,
SubscribeResponseType,
UnsubscribeRequestType,
UnsubscribeResponseType,
ControlRequestType,
ControlResponseType}

Message types should be defined as strings to simplify debugging

ASSUME $\forall m \in messageTypes : m \in$ STRING

MODULE *Messages*

The *Messages* module defines predicates for receiving, sending, and verifying all the messages supported by $E2T$.

This section defines predicates for identifying $E2T$ message types on the network.

$IsSubscribeRequest(m) \triangleq m.type = SubscribeRequestType$

$IsSubscribeResponse(m) \triangleq m.type = SubscribeResponseType$

$IsUnsubscribeRequest(m) \triangleq m.type = UnsubscribeRequestType$

$IsUnsubscribeResponse(m) \triangleq m.type = UnsubscribeResponseType$

$IsControlRequest(m) \triangleq m.type = ControlRequestType$

$IsControlResponse(m) \triangleq m.type = ControlResponseType$

This section defines predicates for validating $E2T$ message contents. The predicates provide precise documentation on the $E2T$ message format and are used within the spec to verify that steps adhere to the $E2T$ protocol specification.

LOCAL $ValidSubscribeRequest(m) \triangleq$ TRUE

LOCAL $ValidSubscribeResponse(m) \triangleq$ TRUE

LOCAL $ValidUnsubscribeRequest(m) \triangleq$ TRUE

LOCAL $ValidUnsubscribeResponse(m) \triangleq$ TRUE

LOCAL $ValidControlRequest(m) \triangleq$ TRUE

LOCAL $ValidControlResponse(m) \triangleq$ TRUE

This section defines operators for constructing $E2T$ messages.

LOCAL $SetType(m,\ t) \triangleq [m$ EXCEPT $!.type = t]$

$SubscribeRequest(m) \triangleq$
  IF $Assert(ValidSubscribeRequest(m),$ "Invalid SubscribeRequest")
  THEN $SetType(m,\ SubscribeRequestType)$
  ELSE Nil

$SubscribeResponse(m) \triangleq$
  IF $Assert(ValidSubscribeResponse(m),$ "Invalid SubscribeResponse")
  THEN $SetType(m,\ SubscribeResponseType)$
  ELSE Nil

$UnsubscribeRequest(m) \triangleq$
  IF $Assert(ValidUnsubscribeRequest(m),$ "Invalid UnsubscribeRequest")

```
        THEN SetType(m, UnsubscribeRequestType)
        ELSE Nil

    UnsubscribeResponse(m) ≜
      IF Assert(ValidUnsubscribeResponse(m), "Invalid UnsubscribeResponse")
        THEN SetType(m, UnsubscribeResponseType)
        ELSE Nil

    ControlRequest(m) ≜
      IF Assert(ValidControlRequest(m), "Invalid ControlRequest")
        THEN SetType(m, ControlRequestType)
        ELSE Nil

    ControlResponse(m) ≜
      IF Assert(ValidControlResponse(m), "Invalid ControlResponse")
        THEN SetType(m, ControlResponseType)
        ELSE Nil
```

The *Messages* module is instantiated locally to avoid access from outside the module.

```
LOCAL Messages ≜ INSTANCE Messages
```

MODULE *Client*

The *Client* module provides operators for managing and operating on $E2T$ client connections and specifies the message types supported for the client.

MODULE *Send*

This module provides message type operators for the message types that can be send by the $E2T$ client.

```
    SubscribeRequest(c, m) ≜
      ∧ GRPC!Client!Send(c, Messages!SubscribeRequest(m))

    UnsubscribeRequest(c, m) ≜
      ∧ GRPC!Client!Send(c, Messages!UnsubscribeRequest(m))

    ControlRequest(c, m) ≜
      ∧ GRPC!Client!Send(c, Messages!ControlRequest(m))
```

Instantiate the $E2T!Client!Send$ module

```
Send ≜ INSTANCE Send
```

MODULE *Receive*

This module provides predicates for the types of messages that can be received by an $E2T$ client.

$SubscribeResponse(c,\ h(\_,\ \_)) \triangleq$
$GRPC\,!\,Client\,!\,Handle(c,\ \text{LAMBDA}\ x,\ m :$
$\land\ Messages\,!\,IsSubscribeResponse(m)$
$\land\ GRPC\,!\,Client\,!\,Receive(c)$
$\land\ h(c,\ m))$

$UnsubscribeResponse(c,\ h(\_,\ \_)) \triangleq$
$GRPC\,!\,Client\,!\,Handle(c,\ \text{LAMBDA}\ x,\ m :$
$\land\ Messages\,!\,IsUnsubscribeResponse(m)$
$\land\ GRPC\,!\,Client\,!\,Receive(c)$
$\land\ h(c,\ m))$

$ControlResponse(c,\ h(\_,\ \_)) \triangleq$
$GRPC\,!\,Client\,!\,Handle(c,\ \text{LAMBDA}\ x,\ m :$
$\land\ Messages\,!\,IsControlResponse(m)$
$\land\ GRPC\,!\,Client\,!\,Receive(c)$
$\land\ h(c,\ m))$

Instantiate the $E2T\,!\,Client\,!\,Receive$ module
$Receive \triangleq \text{INSTANCE}\ Receive$

$Connect(s,\ d) \triangleq GRPC\,!\,Client\,!\,Connect(s,\ d)$

$Disconnect(c) \triangleq GRPC\,!\,Client\,!\,Disconnect(c)$

Provides operators for the $E2T$ client
$Client \triangleq \text{INSTANCE}\ Client$

MODULE *Server*

The *Server* module provides operators for managing and operating on $E2T$ servers and specifies the message types supported for the server.

MODULE *Send*

This module provides message type operators for the message types that can be send by the $E2T$ server.

$SubscribeResponse(c,\ m) \triangleq$
$\land\ GRPC\,!\,Server\,!\,Reply(c,\ Messages\,!\,SubscribeResponse(m))$

$UnsubscribeResponse(c,\ m) \triangleq$
$\land\ GRPC\,!\,Server\,!\,Reply(c,\ Messages\,!\,UnsubscribeResponse(m))$

$ControlResponse(c,\ m) \triangleq$
$\land\ GRPC\,!\,Server\,!\,Reply(c,\ Messages\,!\,ControlResponse(m))$

Instantiate the $E2T!Server!Send$ module
$Send \triangleq$ INSTANCE $Send$

MODULE $Receive$

This module provides predicates for the types of messages that can be received by an $E2T$ server.

$SubscribeRequest(c, h(\_, \_)) \triangleq$
  $GRPC!Server!Handle(c,$ LAMBDA $x, m :$
    $\land Messages!IsSubscribeRequest(m)$
    $\land GRPC!Server!Receive(c)$
    $\land h(c, m))$

$UnsubscribeRequest(c, h(\_, \_)) \triangleq$
  $GRPC!Server!Handle(c,$ LAMBDA $x, m :$
    $\land Messages!IsUnsubscribeRequest(m)$
    $\land GRPC!Server!Receive(c)$
    $\land h(c, m))$

$ControlRequest(c, h(\_, \_)) \triangleq$
  $GRPC!Server!Handle(c,$ LAMBDA $x, m :$
    $\land Messages!IsControlRequest(m)$
    $\land GRPC!Server!Receive(c)$
    $\land h(c, m))$

Instantiate the $E2T!Server!Receive$ module
$Receive \triangleq$ INSTANCE $Receive$

Starts a new $E2T$ server
$Serve(s) \triangleq GRPC!Server!Start(s)$

Stops the given $E2T$ server
$Stop(s) \triangleq GRPC!Server!Stop(s)$

Provides operators for the $E2T$ server
$Server \triangleq$ INSTANCE $Server$

The set of all running $E2T$ servers
$Servers \triangleq GRPC!Servers$

The set of all open $E2T$ connections
$Connections \triangleq GRPC!Connections$

$Init \triangleq GRPC!Init$

$Next \triangleq GRPC!Next$

VARIABLES $e2tServers$, $e2tConns$

$E2T \triangleq$ INSTANCE $E2TService$ WITH
$servers \leftarrow e2tServers$,
$conns \leftarrow e2tConns$,
$Nil \leftarrow [type \mapsto$ ""$]$,
$SubscribeRequestType \leftarrow$ "SubscribeRequest",
$SubscribeResponseType \leftarrow$ "SubscribeResponse",
$UnsubscribeRequestType \leftarrow$ "UnsubscribeRequest",
$UnsubscribeResponseType \leftarrow$ "UnsubscribeResponse",
$ControlRequestType \leftarrow$ "ControlRequest",
$ControlResponseType \leftarrow$ "ControlResponse"

MODULE *TopoService*

The *Topo* module provides a formal specification of the *ONOS* topology service. The spec defines the client and server interfaces for *ONOS Topo* and provides helpers for managing and operating on connections.

CONSTANT Nil

VARIABLES $servers$, $conns$

The *Topo API* is implemented as a *gRPC* service
LOCAL $GRPC \triangleq$ INSTANCE $GRPC$

$vars \triangleq \langle servers, conns \rangle$

Message type constants
CONSTANT
$CreateRequestType$,
$CreateResponseType$
CONSTANTS
$UpdateRequestType$,
$UpdateResponseType$
CONSTANTS
$DeleteRequestType$,
$DeleteResponseType$
CONSTANT
$GetRequestType$,
$GetResponseType$
CONSTANT
$ListRequestType$,
$ListResponseType$
CONSTANT

$WatchRequestType$,
$WatchResponseType$

LOCAL $messageTypes \triangleq$
$\{CreateRequestType$,
$CreateResponseType$,
$UpdateRequestType$,
$UpdateResponseType$,
$DeleteRequestType$,
$DeleteResponseType$,
$GetRequestType$,
$GetResponseType$,
$ListRequestType$,
$ListResponseType$,
$WatchRequestType$,
$WatchResponseType\}$

Message types should be defined as strings to simplify debugging
ASSUME $\forall\, m \in messageTypes : m \in$ STRING

MODULE *Messages*

The *Messages* module defines predicates for receiving, sending, and verifying all the messages supported by *ONOS Topo*.

This section defines predicates for identifying *ONOS Topo* message types on the network.

$IsCreateRequest(m) \triangleq m.type = CreateRequestType$

$IsCreateResponse(m) \triangleq m.type = CreateResponseType$

$IsUpdateRequest(m) \triangleq m.type = UpdateRequestType$

$IsUpdateResponse(m) \triangleq m.type = UpdateResponseType$

$IsDeleteRequest(m) \triangleq m.type = DeleteRequestType$

$IsDeleteResponse(m) \triangleq m.type = DeleteResponseType$

$IsGetRequest(m) \triangleq m.type = GetRequestType$

$IsGetResponse(m) \triangleq m.type = GetResponseType$

$IsListRequest(m) \triangleq m.type = ListRequestType$

$IsListResponse(m) \triangleq m.type = ListResponseType$

$IsWatchRequest(m) \triangleq m.type = WatchRequestType$

$IsWatchResponse(m) \triangleq m.type = WatchResponseType$

This section defines predicates for validating *ONOS Topo* message contents. The predicates provide precise documentation on the *E2AP* message format and are used within the spec to verify that steps adhere to the *E2AP* protocol specification.

LOCAL $ValidCreateRequest(m) \triangleq$ TRUE

LOCAL $ValidCreateResponse(m) \triangleq$ TRUE

LOCAL $ValidUpdateRequest(m) \triangleq$ TRUE

LOCAL $ValidUpdateResponse(m) \triangleq$ TRUE

LOCAL $ValidDeleteRequest(m) \triangleq$ TRUE

LOCAL $ValidDeleteResponse(m) \triangleq$ TRUE

LOCAL $ValidGetRequest(m) \triangleq$ TRUE

LOCAL $ValidGetResponse(m) \triangleq$ TRUE

LOCAL $ValidListRequest(m) \triangleq$ TRUE

LOCAL $ValidListResponse(m) \triangleq$ TRUE

LOCAL $ValidWatchRequest(m) \triangleq$ TRUE

LOCAL $ValidWatchResponse(m) \triangleq$ TRUE

This section defines operators for constructing *ONOS Topo* messages.

LOCAL $SetType(m,\ t) \triangleq [m \text{ EXCEPT } !.type = t]$

$CreateRequest(m) \triangleq$
  IF $Assert(ValidCreateRequest(m),$ "Invalid CreateRequest")
  THEN $SetType(m,\ CreateRequestType)$
  ELSE Nil

$CreateResponse(m) \triangleq$
  IF $Assert(ValidCreateResponse(m),$ "Invalid CreateResponse")
  THEN $SetType(m,\ CreateResponseType)$
  ELSE Nil

$UpdateRequest(m) \triangleq$
  IF $Assert(ValidUpdateRequest(m),$ "Invalid UpdateRequest")
  THEN $SetType(m,\ UpdateRequestType)$
  ELSE Nil

$UpdateResponse(m) \triangleq$
  IF $Assert(ValidUpdateResponse(m),$ "Invalid UpdateResponse")

```
    THEN SetType(m, UpdateResponseType)
    ELSE Nil

DeleteRequest(m) ≜
  IF Assert(ValidDeleteRequest(m), "Invalid DeleteRequest")
   THEN SetType(m, DeleteRequestType)
   ELSE Nil

DeleteResponse(m) ≜
  IF Assert(ValidDeleteResponse(m), "Invalid DeleteResponse")
   THEN SetType(m, DeleteResponseType)
   ELSE Nil

GetRequest(m) ≜
  IF Assert(ValidGetRequest(m), "Invalid GetRequest")
   THEN SetType(m, GetRequestType)
   ELSE Nil

GetResponse(m) ≜
  IF Assert(ValidGetResponse(m), "Invalid GetResponse")
   THEN SetType(m, GetResponseType)
   ELSE Nil

ListRequest(m) ≜
  IF Assert(ValidListRequest(m), "Invalid ListRequest")
   THEN SetType(m, ListRequestType)
   ELSE Nil

ListResponse(m) ≜
  IF Assert(ValidListResponse(m), "Invalid ListResponse")
   THEN SetType(m, ListResponseType)
   ELSE Nil

WatchRequest(m) ≜
  IF Assert(ValidWatchRequest(m), "Invalid WatchRequest")
   THEN SetType(m, WatchRequestType)
   ELSE Nil

WatchResponse(m) ≜
  IF Assert(ValidWatchResponse(m), "Invalid WatchResponse")
   THEN SetType(m, WatchResponseType)
   ELSE Nil
```

The *Messages* module is instantiated locally to avoid access from outside the module.

```
LOCAL Messages ≜ INSTANCE Messages
```

MODULE *Client*

The *Client* module provides operators for managing and operating on *Topo* client connections and specifies the message types supported for the client.

MODULE *Send*

This module provides message type operators for the message types that can be send by the *Topo* client.

$$CreateRequest(c,\ m) \triangleq \\ \quad \land\ GRPC!Client!Send(c,\ Messages!CreateRequest(m))$$

$$UpdateRequest(c,\ m) \triangleq \\ \quad \land\ GRPC!Client!Send(c,\ Messages!UpdateRequest(m))$$

$$DeleteRequest(c,\ m) \triangleq \\ \quad \land\ GRPC!Client!Send(c,\ Messages!DeleteRequest(m))$$

$$GetRequest(c,\ m) \triangleq \\ \quad \land\ GRPC!Client!Send(c,\ Messages!GetRequest(m))$$

$$ListRequest(c,\ m) \triangleq \\ \quad \land\ GRPC!Client!Send(c,\ Messages!ListRequest(m))$$

$$WatchRequest(c,\ m) \triangleq \\ \quad \land\ GRPC!Client!Send(c,\ Messages!WatchRequest(m))$$

Instantiate the *Topo*!*Client*!*Send* module

$$Send \triangleq \text{INSTANCE } Send$$

MODULE *Receive*

This module provides predicates for the types of messages that can be received by an *Topo* client.

$$CreateResponse(c,\ h(\_,\ \_)) \triangleq \\ \quad GRPC!Client!Handle(c,\ \text{LAMBDA } x,\ m : \\ \quad\quad \land\ Messages!IsCreateResponse(m) \\ \quad\quad \land\ GRPC!Client!Receive(c) \\ \quad\quad \land\ h(c,\ m))$$

$$UpdateResponse(c,\ h(\_,\ \_)) \triangleq \\ \quad GRPC!Client!Handle(c,\ \text{LAMBDA } x,\ m : \\ \quad\quad \land\ Messages!IsUpdateResponse(m) \\ \quad\quad \land\ GRPC!Client!Receive(c) \\ \quad\quad \land\ h(c,\ m))$$

$$DeleteResponse(c,\ h(\_,\ \_)) \triangleq \\ \quad GRPC!Client!Handle(c,\ \text{LAMBDA } x,\ m :$$

$\land\ Messages!IsDeleteResponse(m)$
$\land\ GRPC!Client!Receive(c)$
$\land\ h(c,\ m))$

$GetResponse(c,\ h(\_,\ \_)) \triangleq$
$GRPC!Client!Handle(c,$ LAMBDA $x,\ m :$
$\land\ Messages!IsGetResponse(m)$
$\land\ GRPC!Client!Receive(c)$
$\land\ h(c,\ m))$

$ListResponse(c,\ h(\_,\ \_)) \triangleq$
$GRPC!Client!Handle(c,$ LAMBDA $x,\ m :$
$\land\ Messages!IsListResponse(m)$
$\land\ GRPC!Client!Receive(c)$
$\land\ h(c,\ m))$

$WatchResponse(c,\ h(\_,\ \_)) \triangleq$
$GRPC!Client!Handle(c,$ LAMBDA $x,\ m :$
$\land\ Messages!IsWatchResponse(m)$
$\land\ GRPC!Client!Receive(c)$
$\land\ h(c,\ m))$

Instantiate the $Topo!Client!Receive$ module
$Receive \triangleq$ INSTANCE $Receive$

$Connect(s,\ d) \triangleq GRPC!Client!Connect(s,\ d)$

$Disconnect(c) \triangleq GRPC!Client!Disconnect(c)$

Provides operators for the *Topo* client
$Client \triangleq$ INSTANCE $Client$

MODULE *Server*

The *Server* module provides operators for managing and operating on *Topo* servers and specifies the message types supported for the server.

MODULE *Send*

This module provides message type operators for the message types that can be send by the *Topo* server.

$CreateResponse(c,\ m) \triangleq$
$\land\ GRPC!Server!Reply(c,\ Messages!CreateResponse(m))$

$UpdateResponse(c,\ m) \triangleq$
$\land\ GRPC!Server!Reply(c,\ Messages!UpdateResponse(m))$

$DeleteResponse(c,\ m) \triangleq$
$\quad \land\ GRPC!Server!Reply(c,\ Messages!DeleteResponse(m))$

$GetResponse(c,\ m) \triangleq$
$\quad \land\ GRPC!Server!Reply(c,\ Messages!GetResponse(m))$

$ListResponse(c,\ m) \triangleq$
$\quad \land\ GRPC!Server!Reply(c,\ Messages!ListResponse(m))$

$WatchResponse(c,\ m) \triangleq$
$\quad \land\ GRPC!Server!Reply(c,\ Messages!WatchResponse(m))$

Instantiate the $Topo!Server!Send$ module
$Send \triangleq$ INSTANCE $Send$

MODULE *Receive*

This module provides predicates for the types of messages that can be received by an *Topo* server.

$CreateRequest(c,\ h(\_,\ \_)) \triangleq$
$\quad GRPC!Server!Handle(c,$ LAMBDA $x,\ m :$
$\quad\quad \land\ Messages!IsCreateRequest(m)$
$\quad\quad \land\ GRPC!Server!Receive(c)$
$\quad\quad \land\ h(c,\ m))$

$UpdateRequest(c,\ h(\_,\ \_)) \triangleq$
$\quad GRPC!Server!Handle(c,$ LAMBDA $x,\ m :$
$\quad\quad \land\ Messages!IsUpdateRequest(m)$
$\quad\quad \land\ GRPC!Server!Receive(c)$
$\quad\quad \land\ h(c,\ m))$

$DeleteRequest(c,\ h(\_,\ \_)) \triangleq$
$\quad GRPC!Server!Handle(c,$ LAMBDA $x,\ m :$
$\quad\quad \land\ Messages!IsDeleteRequest(m)$
$\quad\quad \land\ GRPC!Server!Receive(c)$
$\quad\quad \land\ h(c,\ m))$

$GetRequest(c,\ h(\_,\ \_)) \triangleq$
$\quad GRPC!Server!Handle(c,$ LAMBDA $x,\ m :$
$\quad\quad \land\ Messages!IsGetRequest(m)$
$\quad\quad \land\ GRPC!Server!Receive(c)$
$\quad\quad \land\ h(c,\ m))$

$ListRequest(c,\ h(\_,\ \_)) \triangleq$
$\quad GRPC!Server!Handle(c,$ LAMBDA $x,\ m :$
$\quad\quad \land\ Messages!IsListRequest(m)$

```
        ∧ GRPC!Server!Receive(c)
        ∧ h(c, m))

    WatchRequest(c, h(_, _)) ≜
      GRPC!Server!Handle(c, LAMBDA x, m :
        ∧ Messages!IsWatchRequest(m)
        ∧ GRPC!Server!Receive(c)
        ∧ h(c, m))
```

```
  Instantiate the Topo!Server!Receive module
Receive ≜ INSTANCE Receive

  Starts a new Topo server
Serve(s) ≜ GRPC!Server!Start(s)

  Stops the given Topo server
Stop(s) ≜ GRPC!Server!Stop(s)
```

```
  Provides operators for the Topo server
Server ≜ INSTANCE Server

  The set of all running Topo servers
Servers ≜ GRPC!Servers

  The set of all open Topo connections
Connections ≜ GRPC!Connections

Init ≜ GRPC!Init

Next ≜ GRPC!Next
```

```
VARIABLE topoServers, topoConns

Topo ≜ INSTANCE TopoService WITH
  servers ← topoServers,
  conns ← topoConns,
  Nil ← [type ↦ ""],
  CreateRequestType ← "CreateRequest",
  CreateResponseType ← "CreateResponse",
  UpdateRequestType ← "UpdateRequest",
  UpdateResponseType ← "UpdateResponse",
  DeleteRequestType ← "DeleteRequest",
  DeleteResponseType ← "DeleteResponse",
```

$GetRequestType \leftarrow$ "GetRequest",
$GetResponseType \leftarrow$ "GetResponse",
$ListRequestType \leftarrow$ "ListRequest",
$ListResponseType \leftarrow$ "ListResponse",
$WatchRequestType \leftarrow$ "WatchRequest",
$WatchResponseType \leftarrow$ "WatchResponse"

* Modification History
* Last modified *Fri Aug* 13 18:57:44 *PDT* 2021 by *jordanhalterman*
* Created *Fri Aug* 13 15:34:11 *PDT* 2021 by *jordanhalterman*