Building Correct SDN Components from a Global Event-B Model

Christian Attiogbé

University of Nantes, France

FACS'2018 / Pohang - June 12 2018

안녕 모두

LS2N LABORATOIRE DES SCIENCES DU NUMÉRIQUE DE NANTES

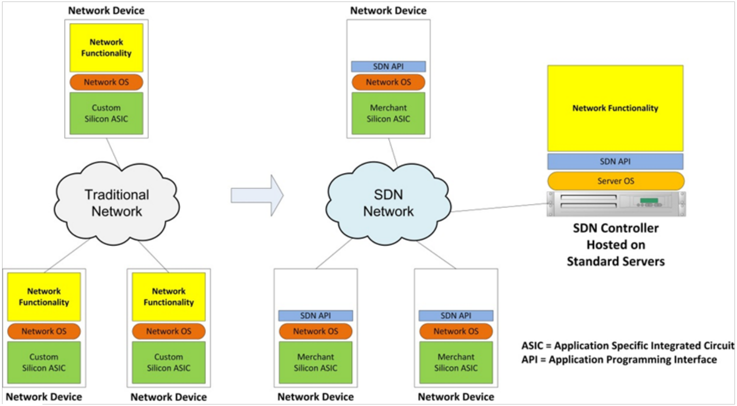
christian.attiogbe@ls2n.fr

Agenda

Agenda

Context : SDN-based systems

Continuous reconfigurations of network supports in offices, companies, institutions ; mobility of users ; modifications of roles, IoT environment, etc



(www.commscope.com)

Context : Building reliable SDN-based systems

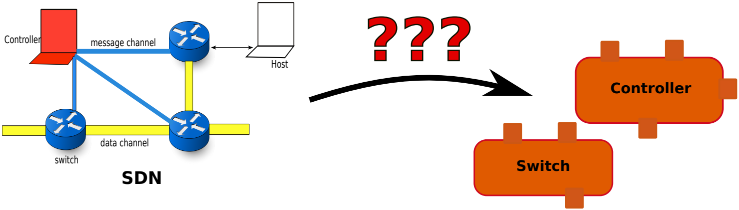
Numerous application domains impacted

Highly reactive systems, distributed applications, IoT, smart manufacturing, etc

SDN as support of critical applications

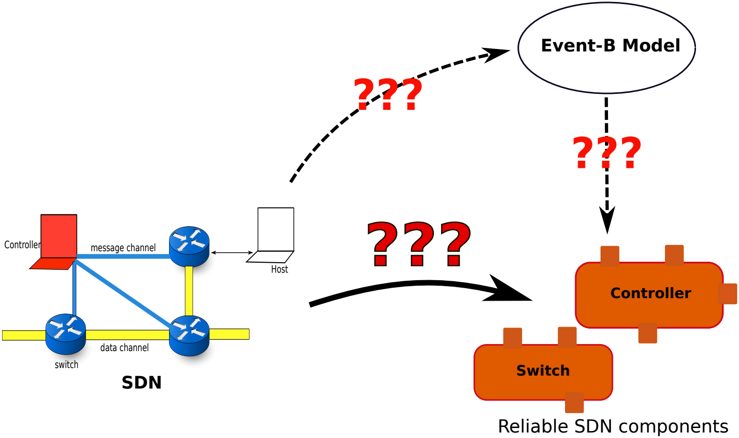
- Correction-by-construction : refinement of abstract models into systems
- Verification of required properties from abstract models

Problem statement

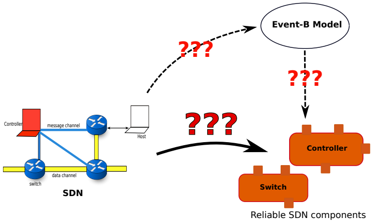


Reliable SDN components

Problem statement



Motivations Correct reusable SDN models



Scientific challenges

- Modelling, tackling the complexity
- Composition of heterogeneous entities
- Refinement method

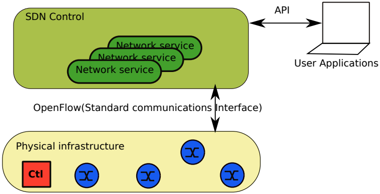
Goals

- Generic formal models
- refinable into dedicated environments

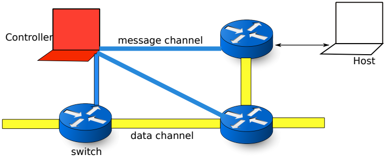
☞ Mastering the implementation of SDN components (switches, controllers)

SDN concepts and components

The layered architecture of a SDN



Interactions in a SDN



Agenda

A global model of an SDN

A mathematical model of an SDN is a model comprising

- its state space and, where
- all its components are interacting simultaneously.

The challenge is to build such a model :

- parallel composition of processes interacting on channels ?
- but a refinement-based approach helps to master the complexity.

In Event-B an abstract model is used to capture the general behaviour of an SDN, through the interrelated behaviours of its components.

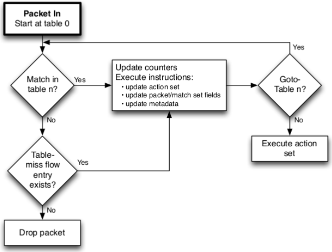
Abstraction of the components

A switch component

Data part

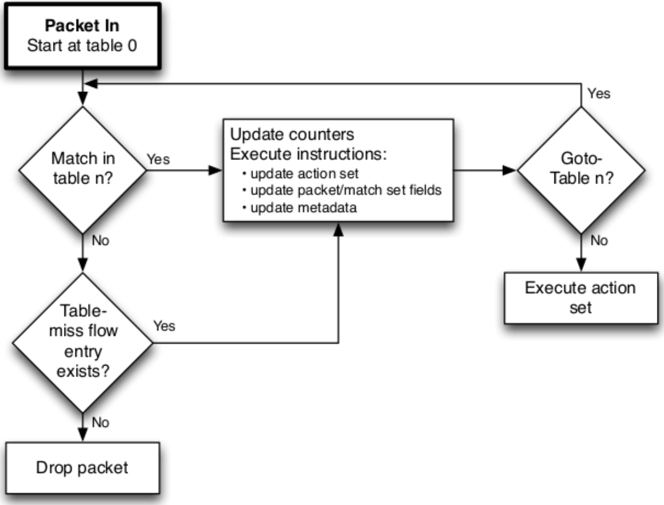
- Packets
- Messages
- Buffers
- Channels
- an entry table set by the controller

Behaviour



(source : ONF TS-006 - OpenFlow switch specification)

A switch reacts to action queries from the controller/user


Packet In
Start at table 0
Match in table n?
Yes
Update counters
Execute instructions:
• update action set
• update packet/match set fields
• update metadata
Goto-Table n?
Yes
No
Execute action set
No
Table-miss flow entry exists?
Yes
No
Drop packet

Abstraction of the controller component

Data part

- Packets
- Messages
- Channels
- Buffers

Behaviour

Interacts with switches to process all packets
Manages packet flow
Sets rules in the switches flow tables
Maintains entry flow tables in the switches

A controller administrates the switches with control messages

Modelling in Event-B

CONTEXT EnvCtx0

SETS

- PACKET set of packets (exchanged between switches, controllers, hosts)
- MESG set of messages (exchanged between switches and controller)
- MESGTYPE message types
- ENTRY set of entries of the flow table
- HEADER header+Actions : set of actions applied by switch to match packets
- SW_ID switch ID
- SW_STATE Openflow switch state

CONSTANTS

PKIn PKOut BarrierQ BarrierR FlowMd askStatus Status AddE DelE ModE

AXIOMS

$$MESGTYPE = \{PKIn, PKOut, BarrierQ, BarrierR, FlowMd, askStatus, Status, AddE, DelE, ModE\}$$

Modelling the the switches

Each switch has :

- a **flow table :** $flowTable \in ENTRY \rightarrowtail switches$
  Each entry has several headers : $eHeader_i \in ENTRY \rightarrowtail HEADER$
  $\mathrm{dom}(eHeader_i) = \mathrm{dom}(flowTable)$
- a **status :**
  $swStatus \in SW\_ID \rightarrowtail SW\_STATE \wedge \mathrm{dom}(swStatus) = switches$
- a **buffer of all received messages :**
  $swIncomingMsg \subseteq MESG \times switches$
- a **buffer of all received packets**, before treatment :
  $swIPk \in PACKET \leftrightarrow switches$
  $swIncomingPk \subseteq PACKET$ and $swIncomingPk = \mathrm{dom}(swIPk)$.
  Each packet has a header : $pHeader_i \in PACKET \rightarrowtail HEADER$

Modelling the the switches

- ...
- a **buffer** *swOMsg* that contains messages to be sent to the controller : $swOMsg \in MESG \leftrightarrow switches$; *swOutgoingMsg* is a set of messages such that $swOutgoingMsg \subseteq MESG \wedge swOutgoingMsg = \mathrm{dom}(swOMsg)$
- a **buffer of packets to be sent** to other switches or to the controller : $swOPk \in PACKET \leftrightarrow switches$ $swOutgoingPk \subseteq PACKET \wedge swOutgoingPk = \mathrm{dom}(swOPk)$.

Modelling the behaviour of the switches

Behaviour of the switch : how it is involved in the interaction with its environment.

sw_rcv_matchingPkt	on the reception of a matching packet
sw_rcv_unmatchingPkt	reception of an unmatching packet
sw_sndPk2ctrl	sending a packet to the controller
sw_sendPckt2sw	sending a packet to another switch
sw_newFTentry	handling a new entry (from the controller)

In Event-B an **event** is modelled with a **guard** and a generalized **substitution**

Switch behaviour : an event

event sw_rcv_matchingPkt // a switch receives a packet matching a flow table entry
ANY sw pkt ahd
WHERE /* the guard */

$sw \in switches \;\wedge\; pkt \in PACKET \;\wedge\; (pkt \mapsto sw) \in dataChan$

$ahd \in HEADER \;\wedge\; pkt \in \mathrm{dom}(pHeader1)$

$ahd = pHeader1(pkt)$

$\exists ee \cdot (((ee \in ENTRY) \wedge (ee \in \mathrm{dom}(flowTable)) \wedge (eHeader1(ee) = ahd))$

$sw \in \mathrm{dom}(swIPk) \;\wedge\; sw \mapsto pkt \notin swIPk$

THEN /* the substitution */

$swIncomingPk := swIncomingPk \cup \{pkt\}$ // input buffer updated ;

$dataChan := dataChan \setminus \{pkt \mapsto sw\}$

$swIPk := swIPk \cup \{sw \mapsto pkt\}$ // packet will be forwarded

END

LS2N LABORATOIRE DES SCIENCES DU NUMÉRIQUE DE NANTES

Modelling the controller in Event-B

A controller has :

- buffers which contain sent/received messages or packets
- a buffer for incoming packets ($ctlIncomingPk \subseteq PACKET$)
- a buffer for outgoing packets ($ctlOutgoingPk \subseteq PACKET$)

A controller administrates the switches with control messages.

Controller behaviour

The events result from the splitting of the components interactions

ctl_emitPkt	when the controller emits a packet
ctl_rcvPacketIn	when the controller receives a packet
ctl_askBarrier	when it asks a barrier
...	

The channels for interactions are modelled with sets.

$$secureChan \subseteq MESG \times switches$$
$$dataChan \subseteq PACKET \times switches$$

LS2N LABORATOIRE DES SCIENCES DU NUMÉRIQUE DE NANTES

Controller behaviour : an event

event ctl_emitPkt // the controller emits a mesg conveying a packet
ANY sw pkt msg
WHERE /* the guard */
$sw \in switches$ // in destination to one of the switches
$pkt \in PACKET$
$pkt \in ctlOutgoingPk$ // one of the packet to be sent on the sw
$msg \in MESG$ // a given message to convey the packet
$(msg \mapsto PKOut) \in mesgType$ // a packet of type OUT
$(msg \mapsto pkt) \in mesgPk$ // the message contains the packet
THEN /* the substitution */
$secureChan := secureChan \cup \{msg \mapsto sw\}$ //emission on the channel
$ctlOutgoingPk := ctlOutgoingPk \setminus \{pkt\}$
END

LS2N LABORATOIRE DES SCIENCES DU NUMÉRIQUE DE NANTES

The global abstract model

```
MACHINE GblModel0
SEES EnvCtx0
VARIABLES
...
INVARIANTS
...
```

```
EVENTS
initialisation = ...
sw_rcv_matchingPkt = ...
sw_rcv_unmatchingPkt = ...
sw_sndPk2ctrl = ...
sw_sendPckt2sw = ...
sw_newFTentry = ...
ctl_emitPkt = ...
ctl_rcvPacketIn = ...
ctl_askBarrier = ...
...
END
```

The invariants include the correctness properties

Correctness conditions of the global model

The properties are built from the understanding and structuring of the SDN

Outgoing packets are sent by one of the switches or by the controller

$$swOutgoingPk \subseteq swSentPkts \cup ctlSentPkts$$

Packets in data channel should be sent by the controller or the switches

$$\mathrm{dom}(dataChan) \subseteq swSentPkts \cup ctlSentPkts$$

The contents of the switches buffers should come from the controller/switches

$$swIncomingPk \subseteq ctlSentPkts \cup swSentPkts$$

Agenda

LABORATOIRE DES SCIENCES DU NUMÉRIQUE DE NANTES

christian.attiogbe@ls2n.fr

How to refine the global model ?

Refinement challenge

A well-known challenge is to determine the refinement steps according to the problem at hand.

We deal with the components and their related characteristics

- what are the main characteristics of the switches and how they impact their environments ?
- what are the main characteristics of the controller behaviour and how they impact its environment ?

Switches use **various ports** and they **receive/emit messages on ports**. The abstract model of channels is impacted and refined accordingly.

Refinement policy

After the identification of the characteristics of components

Abstact model GblModel0	All the events are specified at a high level
Refinement GblModel0_1	Ports and headers are introduced.
Refinement GblModel0_2	Priorities are introduced in state space ; messages are sent from the controller with a priority
Refinement GblModel0_3	The events guard are refined according to priority rules

Refinement 1 : ports, actions, packets

The set of ports (PORTID) is introduced as data refinement

$actionsQueues \in PORTID \nrightarrow \mathbb{P}(PACKET)$	// packets targeting a port
$actions \in ENTRY \nrightarrow \mathbb{P}(ACTION)$	// ports concerned by an entry
$\mathrm{dom}(actions) = \mathrm{dom}(flowTable)$	// all entries have target ports

Packets are data-refined ; their fields are modelled with functions :
macSrc, *macDst*, *IpSrc*, *IpDst*, *IpProto*, *TpSrc*, *TpDst*, *TpSrcPt*, *TpDstPt*

$macSrc \in PACKET \nrightarrow MACADR$

Refinement 2 : Introducing explicit priorities

Explicit priorities are introduced as information contained in the messages.

MSG_PRIORITY : the set of priorities (a subset of naturals).
$msgPriority \in MESG \pfun MSG\_PRIORITY$

Related events are refined accordingly (example : ctl_emitPkt)

Refinement 2 : Introducing implicit priorities

There are implicit (consistency) priorities between events

- Identify the priorities
- Refine the event accordingly

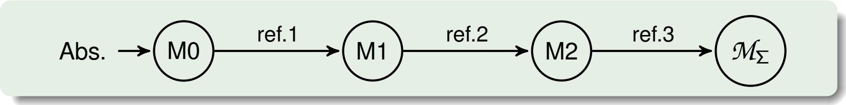
Example : *messages modifications should have lower priority compared with the forwarding messages*.

Priority rule : the add control messages are processed after the forwarding of all data packets

$$sw\_newFTentry \prec sw\_sendPckt2sw$$
$$sw\_sendPk2ctrl \prec sw\_sendPckt2sw$$
$$sw\_sendPk2ctrl \prec sw\_newFTentry$$

The global Event-B machine

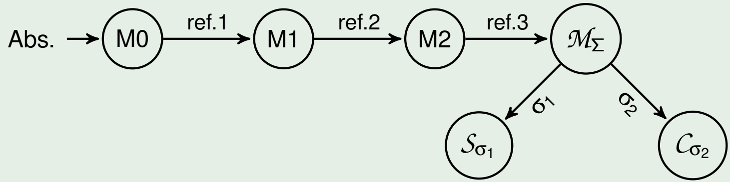
Result of several refinements.



Agenda

Deriving the components by decomposition

$\mathcal{M}_\Sigma$ is composed of the components $\mathcal{S}_{\sigma_1}$ and $\mathcal{C}_{\sigma_2}$
$\Sigma = \sigma_1 \cup \sigma_2$ used to decompose $\mathcal{M}_\Sigma$



Each component is correct wrt to global SDN properties, and can now be refined into specific code

Experimentation with the decomposition plugin of the **Rodin** toolkit

Agenda

Consistency and safety properties

Consistency PO : each event e = (P_e, S_e) preserves the invariant $I(gcv)$.

$$I(gcv) \wedge P_e(bv, gcv) \wedge \text{prd}_{bv}(S_e) \Rightarrow [S_e]I(gcv)$$

Safety properties :

$\mathbf{SP}_a$	Any packet in the data channel was sent by the controller or the switches $swOutgoingPk \subseteq swSentPkts \cup ctlSentPkts$
$\mathbf{SP}_b$	Any packet in the switches buffers was sent by the controller or the switches
$\mathbf{SP}_c$	The packets sent via the message channel are contained in *ctl_sentPkts*

TABLE – A part of the considered safety properties

LS2N LABORATOIRE DES SCIENCES DU NUMÉRIQUE DE NANTES

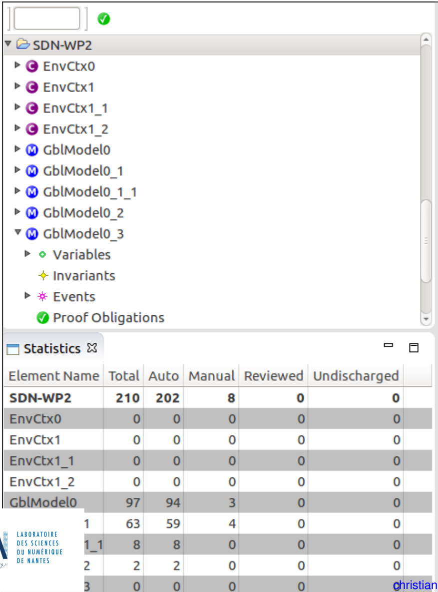
Liveness properties

Event-B provides the facilities to state and prove liveness properties, via **ProB** the model-checker integrated in Rodin.

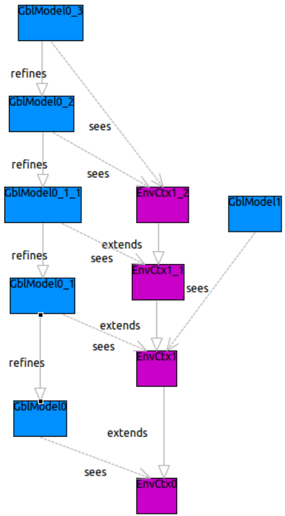
Example : *After the occurrence of the event* `ctl_havePacket` *we will finally (F) observe the occurrence of* `ctl_emitPkt`.

$\mathbf{LP}_{deliv}$	e(ctl_havePacket) ⇒ F(e(ctl_emitPkt))
$\mathbf{LP}_{OKstatus}$	e(ctl_askStatusMsg) ⇒ F(e(ctl_rcvStatus))
$\mathbf{LP}_{OKMach}$	e(ctl_emitPkt) ⇒ X(e(sw_rcv_machingPkt))

Experimentation with Rodin



Element Name	Total	Auto	Manual	Reviewed	Undischarged
SDN-WP2	210	202	8	0	0
EnvCtx0	0	0	0	0	0
EnvCtx1	0	0	0	0	0
EnvCtx1_1	0	0	0	0	0
EnvCtx1_2	0	0	0	0	0
GblModel0	97	94	3	0	0
1	63	59	4	0	0
1_1	8	8	0	0	0
2	2	2	0	0	0
3	0	0	0	0	



Conclusion

Stepwise construction of SDN components

Systematic construction of the global model using refinements
The recipe : interaction between components ; mutual impact on the environment

Event-B decomposition technique to build components
Simulation of their interactions

Tool-assistance : Rodin+ProB

Modelling challenge : fine splitting of the components interactions

Correctness : properties captured from the SDN systems requirements

christian.attiogbe@ls2n.fr

Next step

Parametric derivation of controllers

Deployment of IoT-based systems requires the easy development of dedicated controllers.

- Parametric derivation of specific components
- Capture the features of the target component/environment
- Define associated refinement policy

Thanks

Thank for your attention !

감사

Any questions ?

LABORATOIRE DES SCIENCES DU NUMÉRIQUE DE NANTES