A framework to evaluate 5G networks for smart and fail-safe communications in ERTMS/ETCS

Roberto Canonico(*), Stefano Marrone(**),
Roberto Nardone(*), and Valeria Vittorini(*)

(*) Università degli Studi di Napoli "Federico II", Italy
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione DIETI

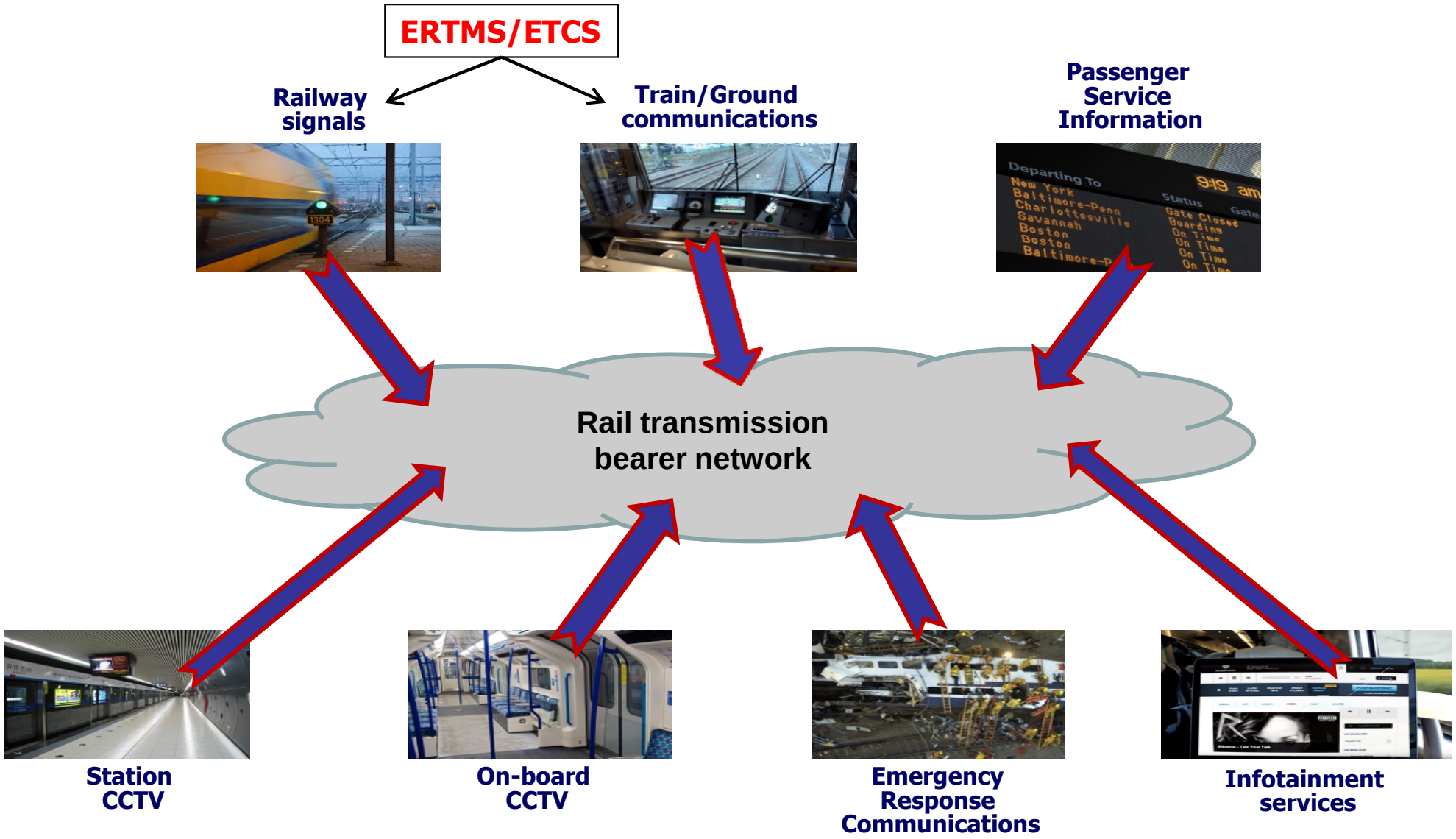
(**) Università della Campania "Luigi Vanvitelli", Dip. di Matematica e Fisica (Italy)

International Conference Reliability, Safety and Security of Railway Systems: Modelling, Analysis, Verification and Certification
November 14-16, 2017 - Pistoia, Italy

Work motivations and rationale

- Context: evolution of wireless communication technologies and need to upgrade existing communication infrastructures for railways
- General goal: investigate opportunities and challenges deriving from the adoption of the SDN paradigm (characterizing future 5G networks) in the specific railway context
- Specific research objectives: automatic control-plane logic generation derived from network topology, knowledge of train trajectories and QoS communication requirements of different applications (in particular, *signaling*)
- Methodology: design a framework that combines and integrates formal modeling and analysis tools and techniques into a network emulator, to evaluate the impact on safety and security deriving from the adoption of an SDN centralized control plane in a railway communication infrastructure
- Contributions:
  - a first architectural frame design, aimed at evaluating the role and interplay of different tools and methodologies
  - a prototypal implementation of the framework

Convergence of services in future Rail communications infrastructure

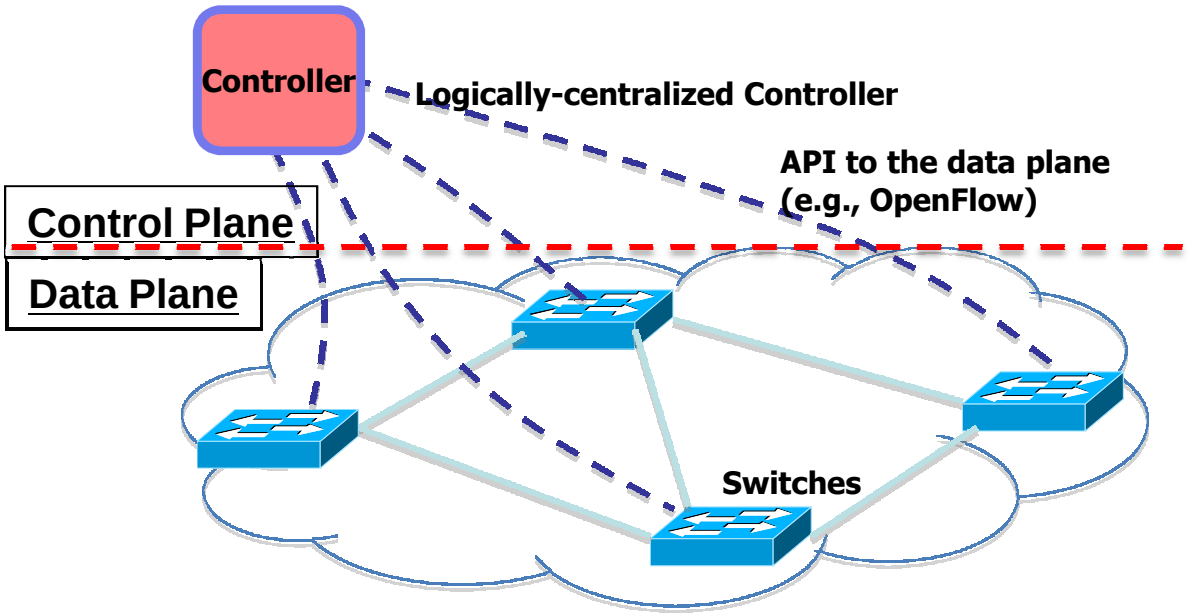


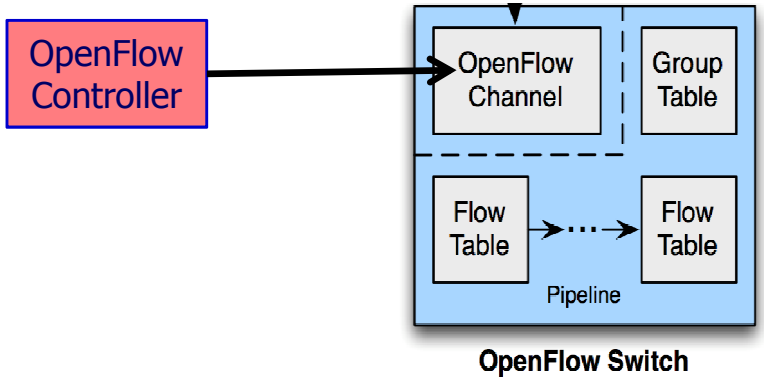
Network softwarization and 5G (1/2)

- Networking relies on separation of concerns:
  - ***data plane*** is responsible of packet processing and forwarding
  - ***control plane*** is responsible of establishing state in network devices to make the whole network behave as expected under varying conditions
    - E.g. routing
  - ***management plane***, by configuring devices, determines how the control plane should be configured
- Traditional networking embeds both data and control plane functions in network devices
  - The two functions work at different timescales and hence are differently implemented
- The emerging *Software Defined Networking* (SDN) paradigm relies on physical decoupling of data and control plane functions
  - Control plane in SDN is logically centralized and implemented as a program running in *controller* entities → ***softwarization***

Network softwarization and 5G (2/2)

- The SDN paradigm is central to the design of future 5G networks
- SDN may be implemented in different ways
  - We consider the OpenFlow model
- In OpenFlow a network device (*switch*) behavior is controlled by a number of *flow-tables* (state) describing, by means of a set of rules, how packets should be treated by the switch





MAC src	MAC dst	IP src	IP dst	TCP sport	TCP dport	Action

Flow entry rule

Opportunities for SDN-based rail networks

- **Differentiated end-to-end QoS for vital and non-vital flows:** in a convergent communication infrastructure, traffic flows have different communication requirements
- **Flexible resource management:** application-aware resource allocation in the presence of different services (signaling, infotainment, …) and tenants (e.g., network slicing and partitioning)
- **Reactivity:** adopt topology- and application-aware reconfiguration strategies after network failures (e.g., loss of nodes) and/or anomalous application-level events (e.g., loss of MA)
- **Proactivity:** ability to pre-configure network for efficiently managing large number of traffic flows (e.g., network handover)
  - E.g. to collectively and proactively manage the handover events of thousands of passengers' mobile devices when they travel together on the same train
- **Advanced network services:** traffic logs, infotainment content delivery

Vision and approach

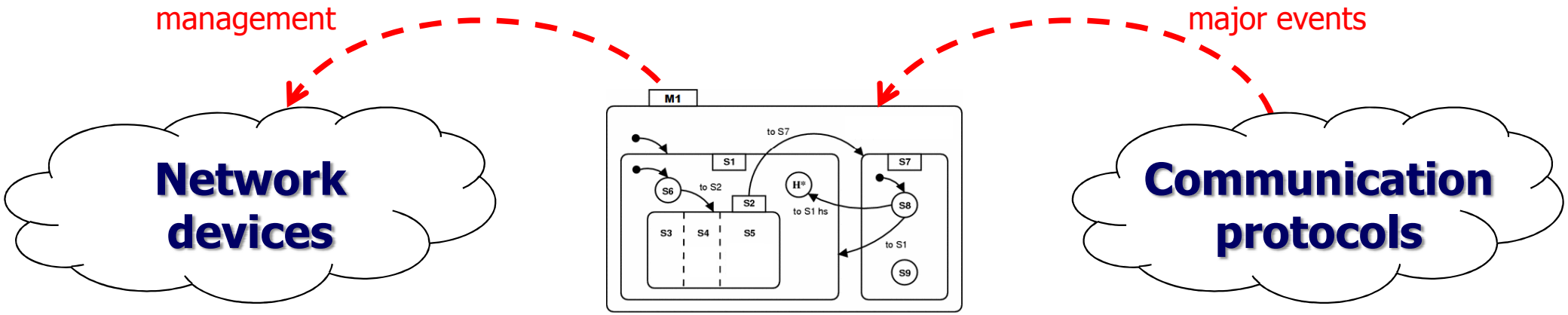
Vision:

implement application-aware network controller of a convergent communication infrastructure, able to automatically configure network devices considering specific communication requirements of different traffic flows

Approach:

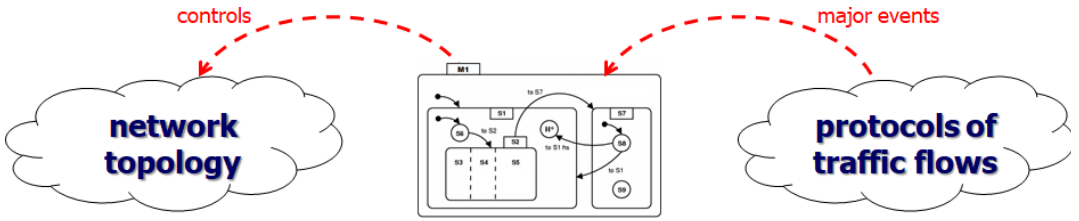
*model the network controller as a **"dynamic" state machine (DSTM)***

and subsequently derive a runnable controller directly from the model

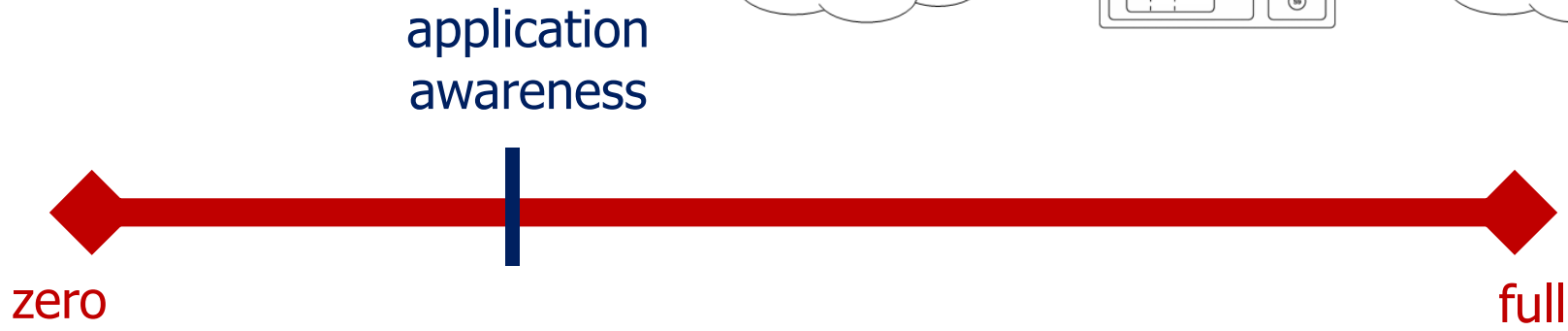


Network controller model

A model-based design of network controller



Problem:



Idea:

A formal state-based model of the network controller can include all the events to consider:

- nominal conditions (for proactive strategies)
- network failures
- emergency conditions
- …

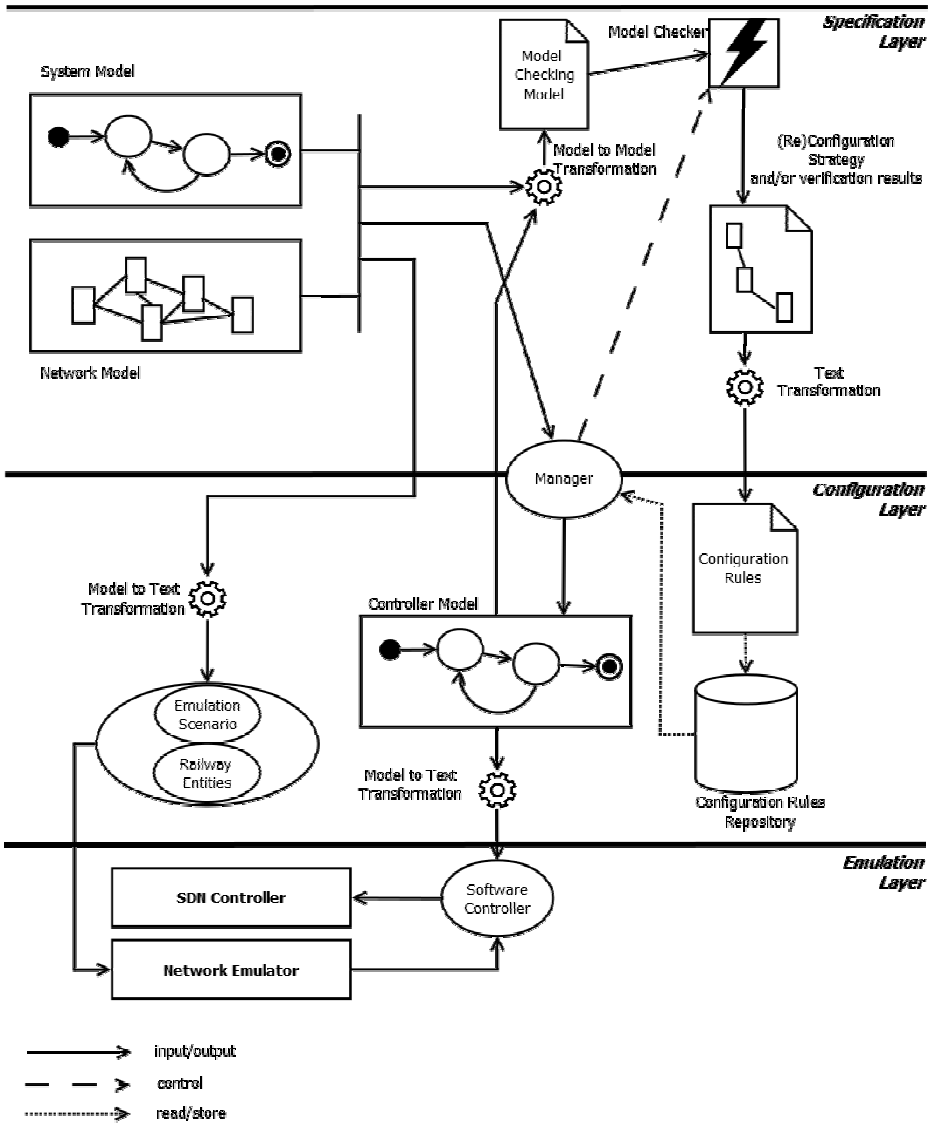
Network controller as a DSTM

- The DSTM (*Dynamic STate Machines*) formalism is an extension of hierarchical state machines (*)
- Basically, a DSTM machine can be parametric and dynamically instantiated
- A **DSTM machine** models how the network is configured to support communication between a centralized control centre (RBC) and a single train
- A **DSTM state** models a specific network configuration at a specific time
- A **DSTM transition** models the occurrence of a network reconfiguration, triggered by events such as train movement, network device failures, application-level alarms, emergency conditions, …
- A DSTM model can be automatically transformed into a runnable Python script that configures a "traditional" SDN controller

(*) M.Benerecetti, R.DeGuglielmo, U.Gentile, S.Marrone, N.Mazzocca, R.Nardone, A.Peron, L.Velardi, V.Vittorini. *Dynamic state machines for modelling railway control systems*. Science of Computer Programming, 133:116-153, 2017.

The proposed framework architecture

- Three logical layers:
  - Specification
  - Configuration
  - Emulation



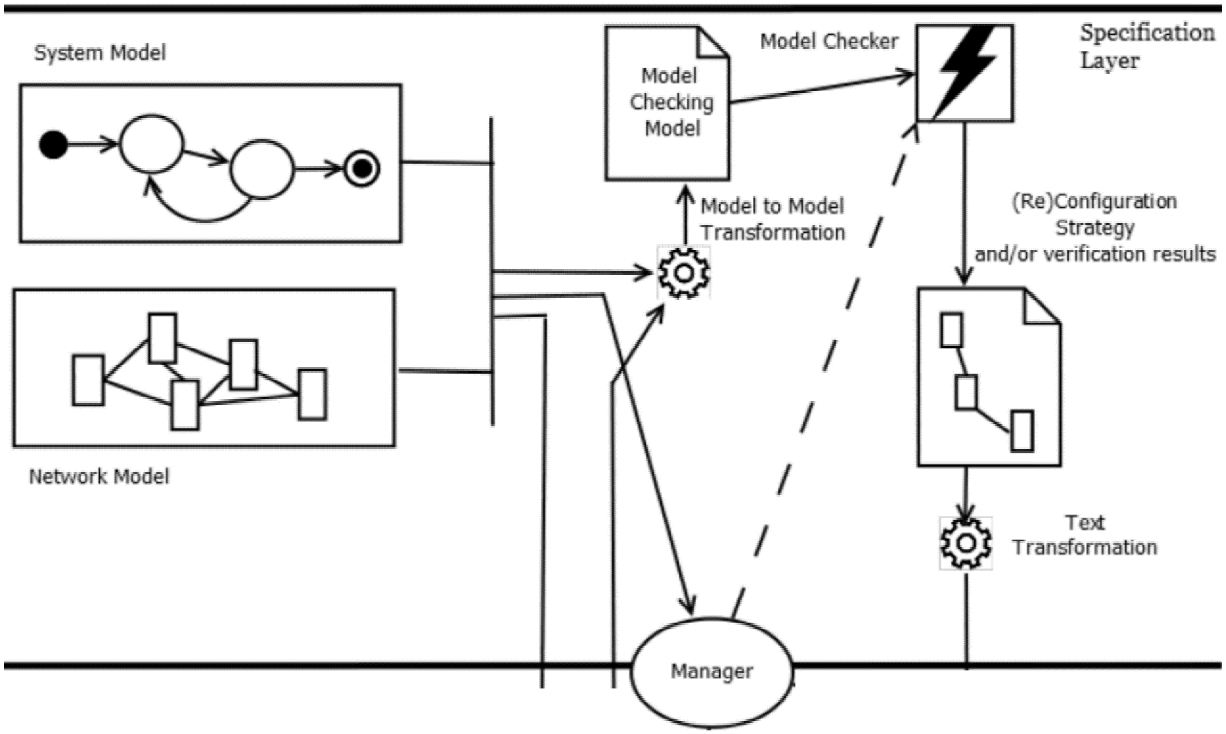
Specification Layer

Purpose:

generate and verify the rules to configure the network

Outputs

Network configuration rules to reach a train from the control centre and vice versa in a whatever position along the track



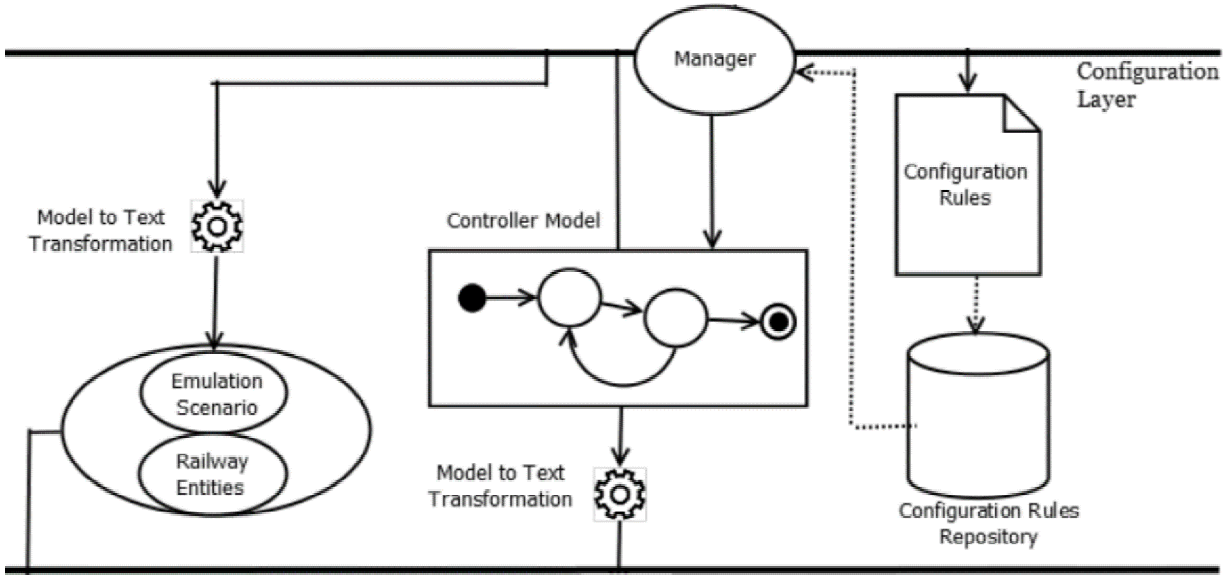
Methodology and tools

We use model checking (SPIN) to generate configuration rules for both the nominal and degraded network conditions. Through this approach we generates all the configuration rules to be executed in the network to allow the communications between the end-points.

Configuration Layer

Purpose:

generate the artefacts needed by the emulation layer, in form of configuration files



Inputs

Configuration rules from the specification layer

Outputs

Controller model

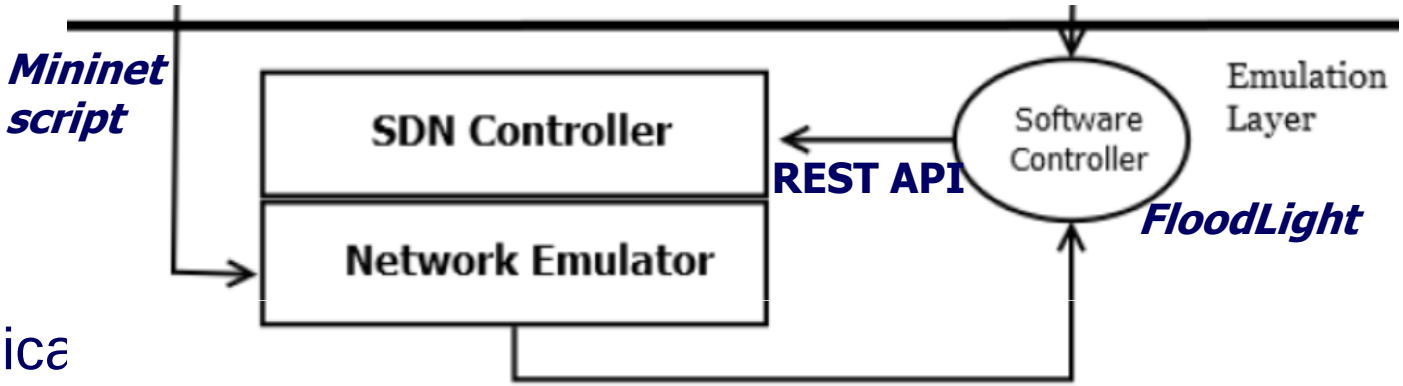
Description files of both scenario and railway entities for the emulator

Methodology and tools

We use exploit model-to-model transformations

Emulation Layer

- **Purpose**:
  Reproduce
  in a virtualized
  environment the
  same traffic flows
  generated by real applica
  and observe how these traffic flows are managed by network devices assembled in a realistic (but small scale) topology



- **Methodology and tools used in current prototype**: we combine (in a VM)
  - an emulator of a SDN network (***Mininet-WiFi***) that is able to instantiate a network made of OpenFlow software switches (Open vSwitch, OVS) and of virtualized end systems
  - a full-fledged open-source OpenFlow Controller, FloodLight, written in Java
- **Inputs**:
  - a Python script that can be directly run in the Mininet emulator to reproduce data plane
  - a Python script that instructs the FloodLight controller through its REST API
- **Outputs**: the execution of the script reproduce the network behavior
  Traffic-level or application-level metrics can be gathered at runtime

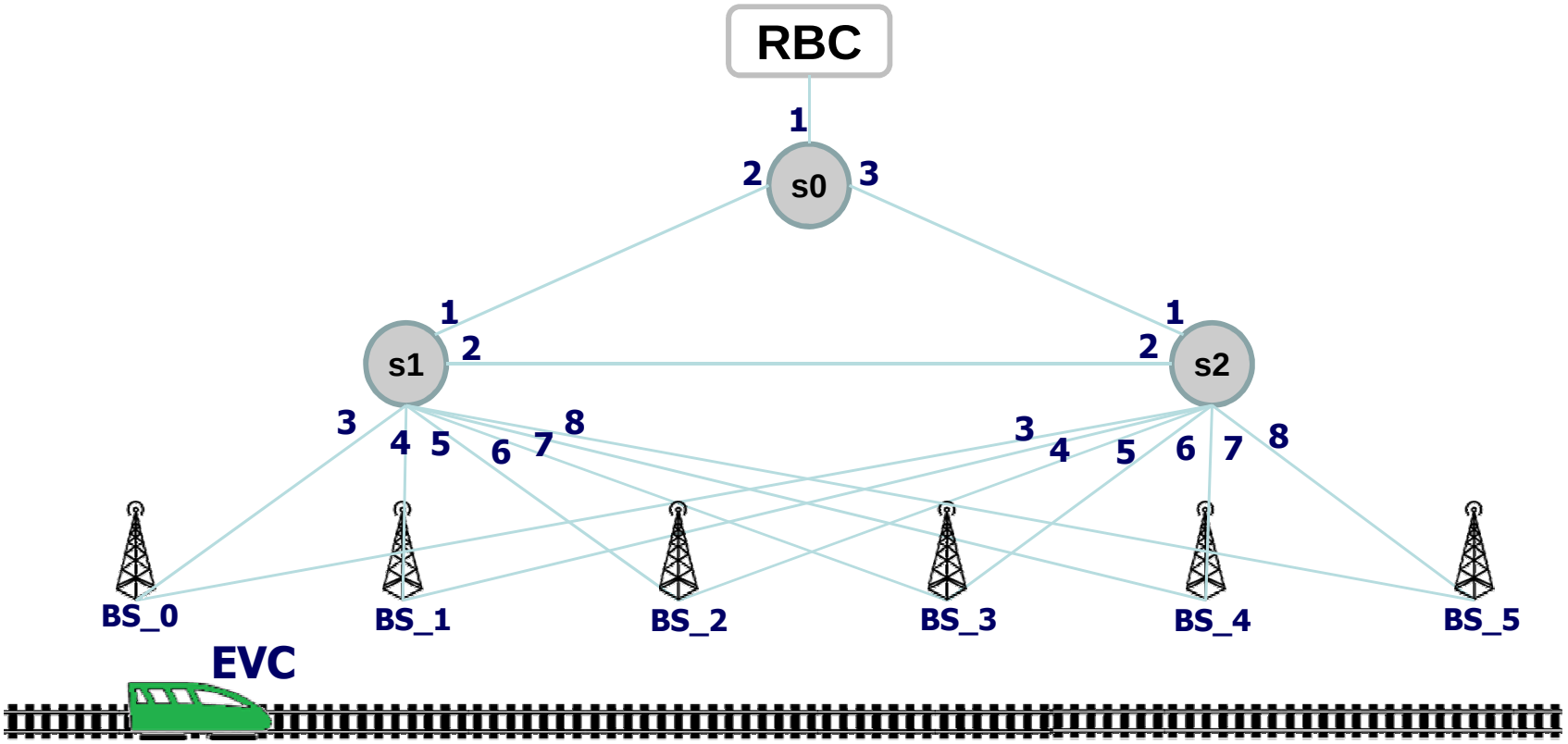
On the faithfulness of the Emulation Layer

- The Emulation Layer is meant to test the effectiveness of the control strategy by reproducing as faithfully as possible the real operational conditions
- We need a network emulator able to reproduce a virtualized environment made of OpenFlow-enabled switches and mobile hosts whose position can be controlled in a virtual space
- The use of an emulator allows us to include in the experiments real applications and directly collect from them the most relevant performance indicators
  - In our prototype we implemented a simplified version of the train signaling protocol
- We chose Mininet-WiFi even though it is only able to reproduce the WiFi lower layers of communication

Role of model checking in our framework

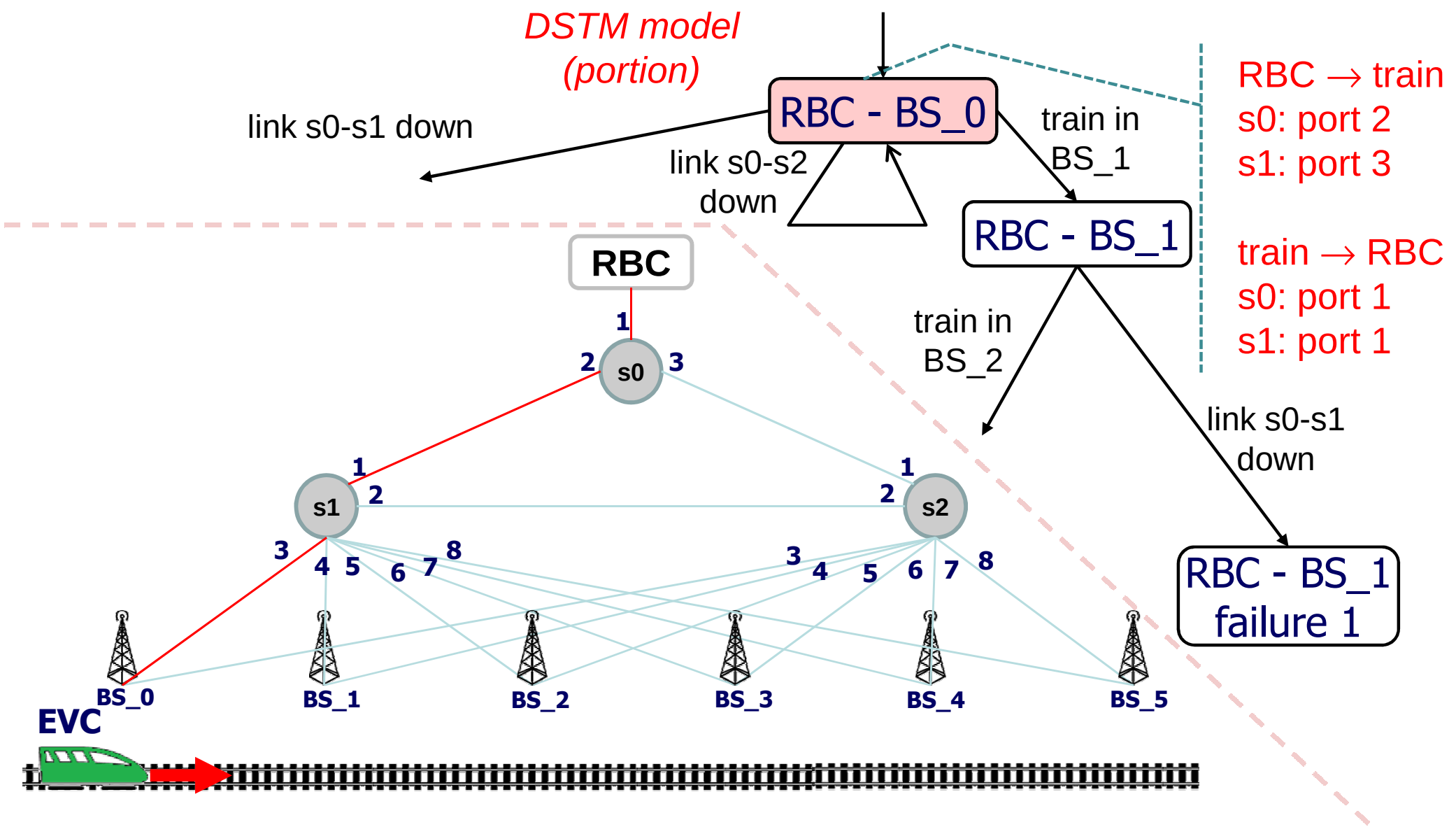
- In our general framework we use model checking to generate and verify a set of rules to configure the network
- In our current prototype we use the Spin model checker
- We use the Promela language to describe both the network topology and the essential events typically generated by the communication procedures between RBC and EVC
- The model checker output is a counterexample related to a violated property
- We define ad hoc properties (*network unreachability*) to obtain network routes that we translate into network device configurations
- The SDN controller behaves as a finite state machine

A first reference scenario

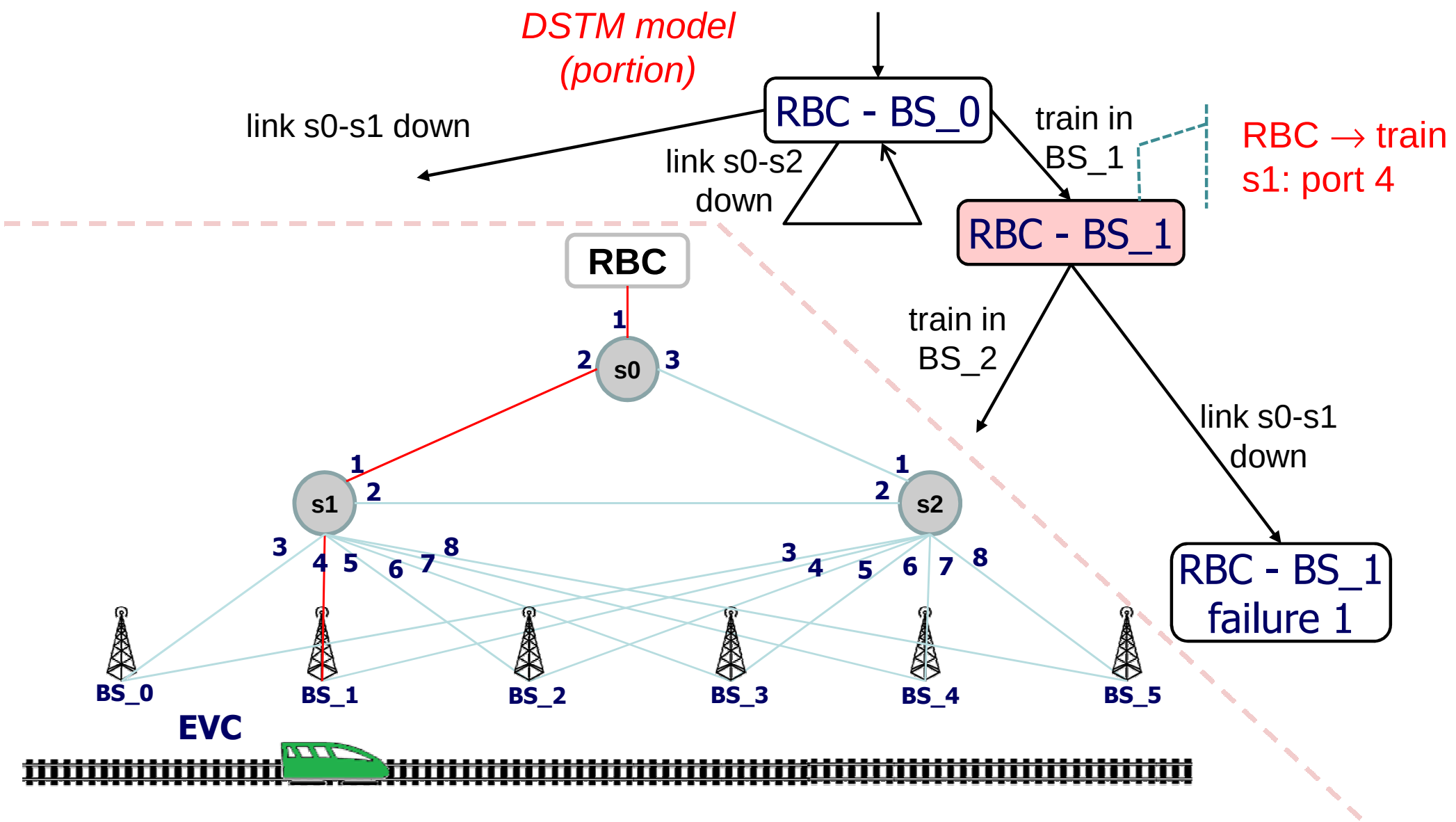


- A linear piece of railway with a single train
- Network infrastructure: two levels hierarchy of switches + base stations
- Communicating hosts: RBC (fixed, at the root) and EVC (on board)

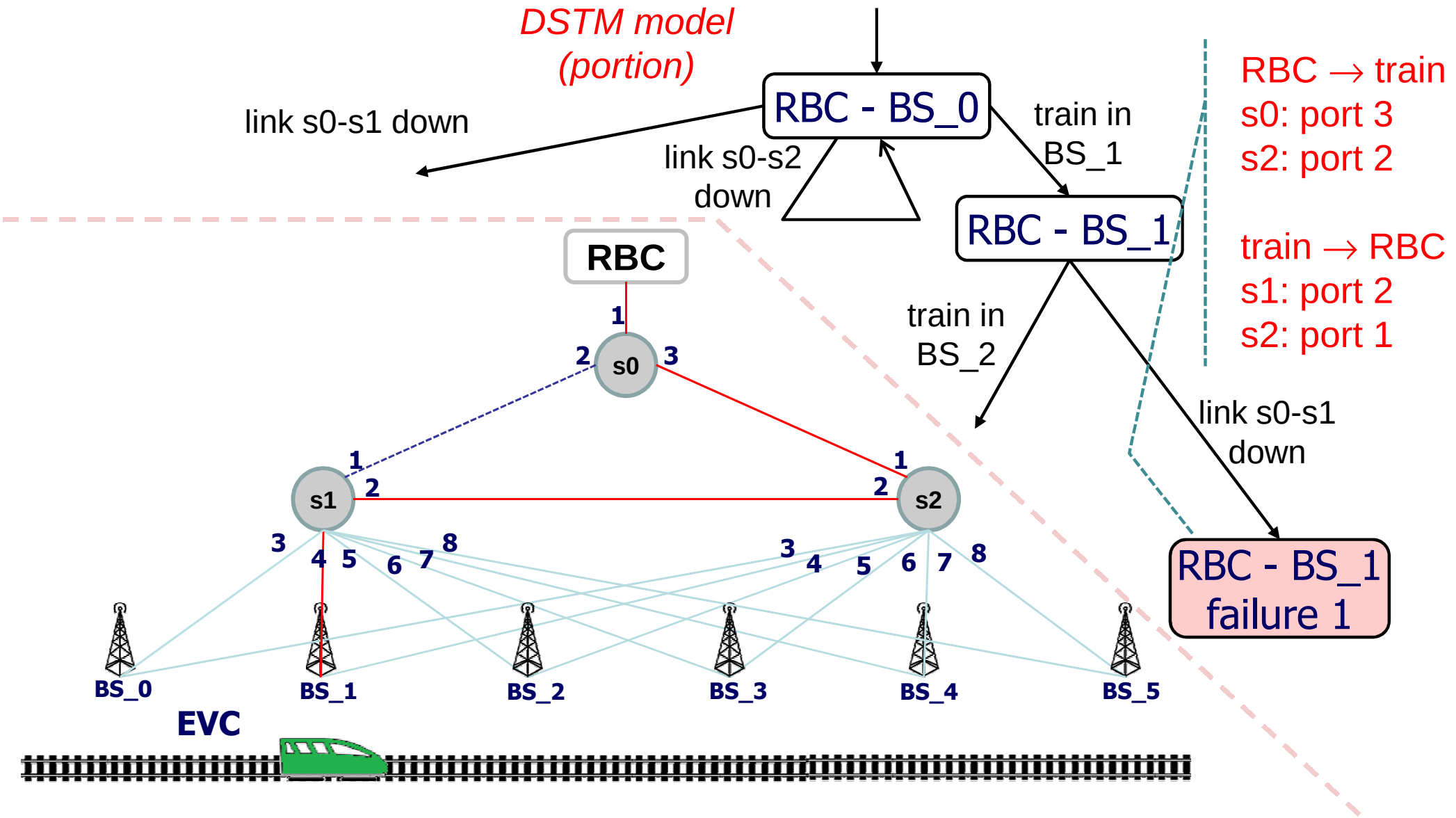
Controller model (in DSTM)



Controller model (in DSTM)



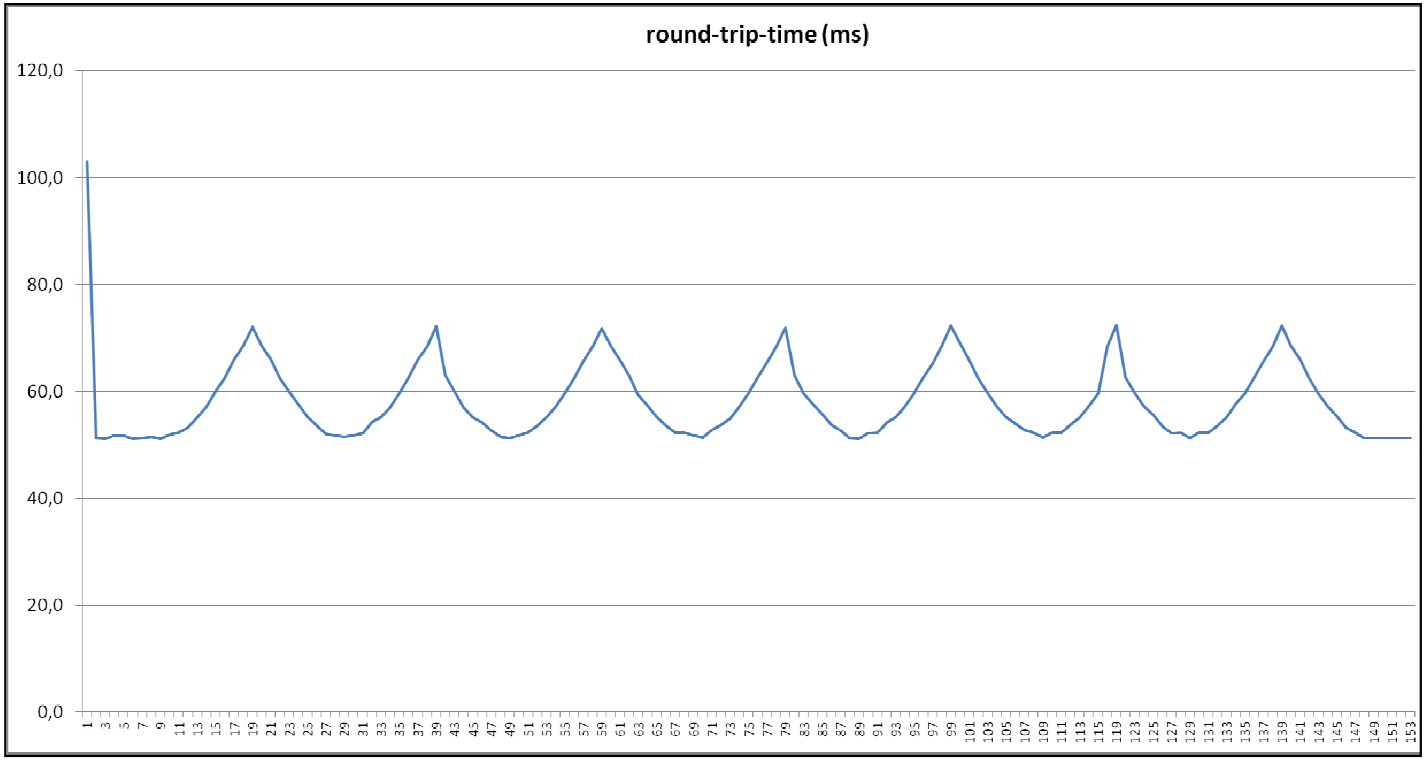
Controller model (in DSTM)



Automatic network configuration and its enforcement

- The control logic is implemented as a set of rules that are passed to the SDN controller, which, in turn, configures the switch flow tables
  - We adopt a hybrid reactive/proactive control strategy
  - It is *reactive*, in the sense that it must be able to change after relevant events (e.g. a train handover, a link or base station failure, …)
  - It is *proactive*, in the sense that it anticipates most of the network level events according to a precomputed strategy to achieve a higher degree of scalability in the control plane
- The SDN controller logic may be changed by instructing it through a Northbound API
  - In our prototype, we adopt the FloodLight OpenFlow controller and its proprietary REST API

Example of metrics collected at runtime



- By running an emulation script in which a single train moves in the sample reference scenario from left to right at a constant speed we collect ping round-trip times (RTT) from train to the RBC
- RTTs are variable over time, depending on the distance of the train from the closest base station

Conclusions and Future Work

- The proposed framework is still under refinement
- Some of the adopted tools have been changed based on the experience we gained from the prototype
- We are still investigating the limits of the approach and the maximum size of the system that can be reasonably dealt
- A more faithful emulation layer implementation is also been investigated
  - Integration of Mininet and ns-3 is considered to reproduce more faithfully the behaviour of cellular networks

Q & A