Towards Zero Touch Configuration of 5G Non-Public Networks for Time Sensitive Networking

SOFTWARE
UNIVERSIDAD DE MÁLAGA

MORSE

F. Luque-Schempp, L. Panizo,
M-d-M. Gallardo, P. Merino and J. Rivas
PROLE 2022

OVERVIEW

01. INTRODUCTION

02. ZERO TOUCH CONFIGURATION OF THE 5G NPN

03. EXPERIMENTAL TESTBED

04. AUTOMATA LEARNING APPROACH

05. EXPERIMENTS

06. CONCLUSIONS & FUTURE WORK

Introduction

Industrial environments are adopting **Time Sensitive Networking** (TSN)

- Evolution of Ethernet standards (wired)
- Guarantee deterministic traffic (jitter, latency, etc.)

Factories of the Future will require mobility and remove cables → **TSN over 5G**

Challenges of TSN over 5G

- New procedures and entities to relate TSN and 5G domains
  - Time synchronization, traffic prioritization/mapping
  - DS-TT (device side), NW-TT (network side), TSN Application Function
- New traffic patterns with very demanding Key Performance Indicators (KPIs)
- Dynamic configuration of a complete 5G Non-Public Network (5G-NPN)

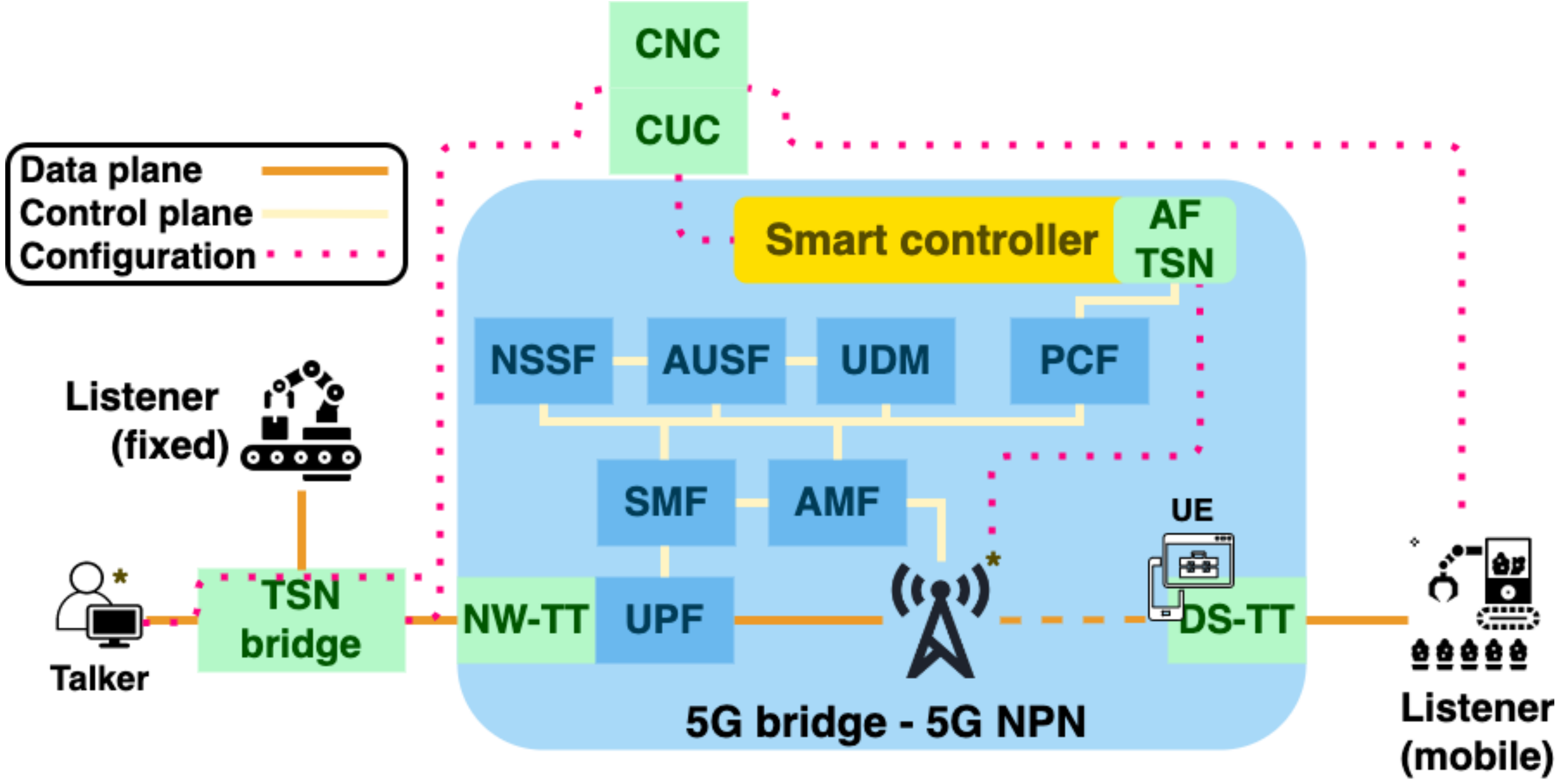
Introduction

Novel approach to automate the network re-configuration process to support TSN over 5G

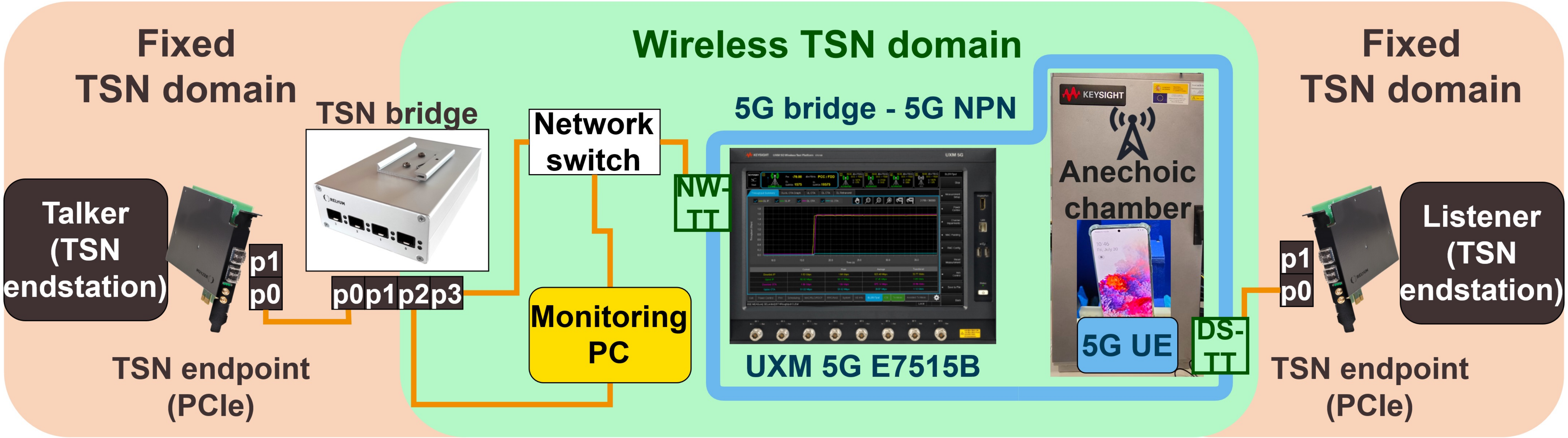
- Based on a real 5G NPN network
- Use learning techniques to produce an automata-based model of the network
- The model is used to predict the network behavior and decide the future configuration to meet the traffic requirements

Zero Touch Configuration of the 5G NPN

- The TSN endpoints (talker) *communicate in advance* their intended traffic pattern and requirements to the CUC/CNC
- The Smart controller beside CUC/CNC and TSN AF entities will be in charge of the 5G NPN configuration (RAN & core network)
- Zero touch approach: without human interaction



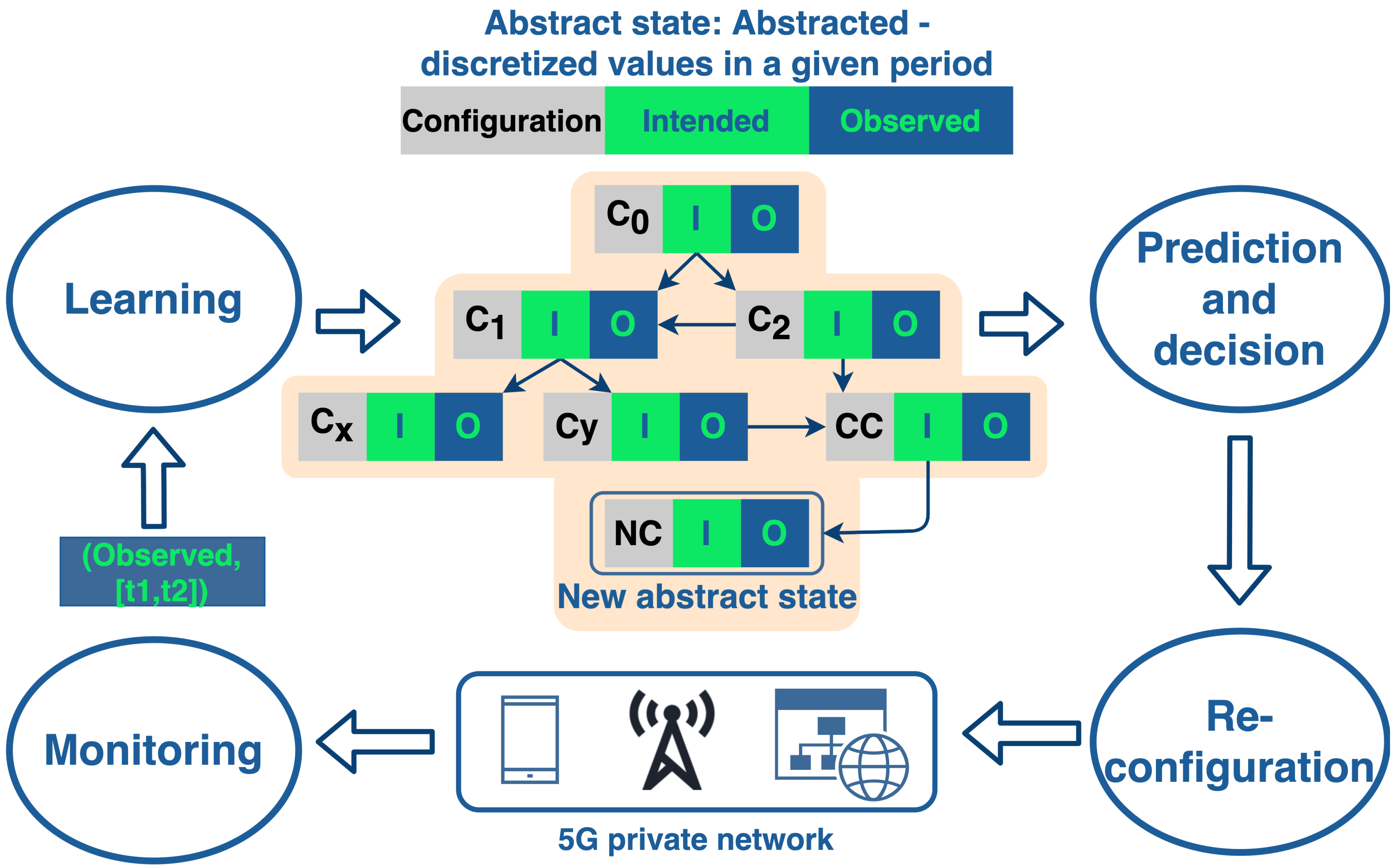
Experimental Testbed



Automata Learning Approach

Close-loop approach

- Monitoring and learning from set of traces
- Automaton construction (abstract states)
  - Configuration
  - Intended traffic
  - Observed traffic
- Prediction and decision
  - Deviation detection
  - Next configuration
- Re-configuration
  - Apply the new values of the parameters



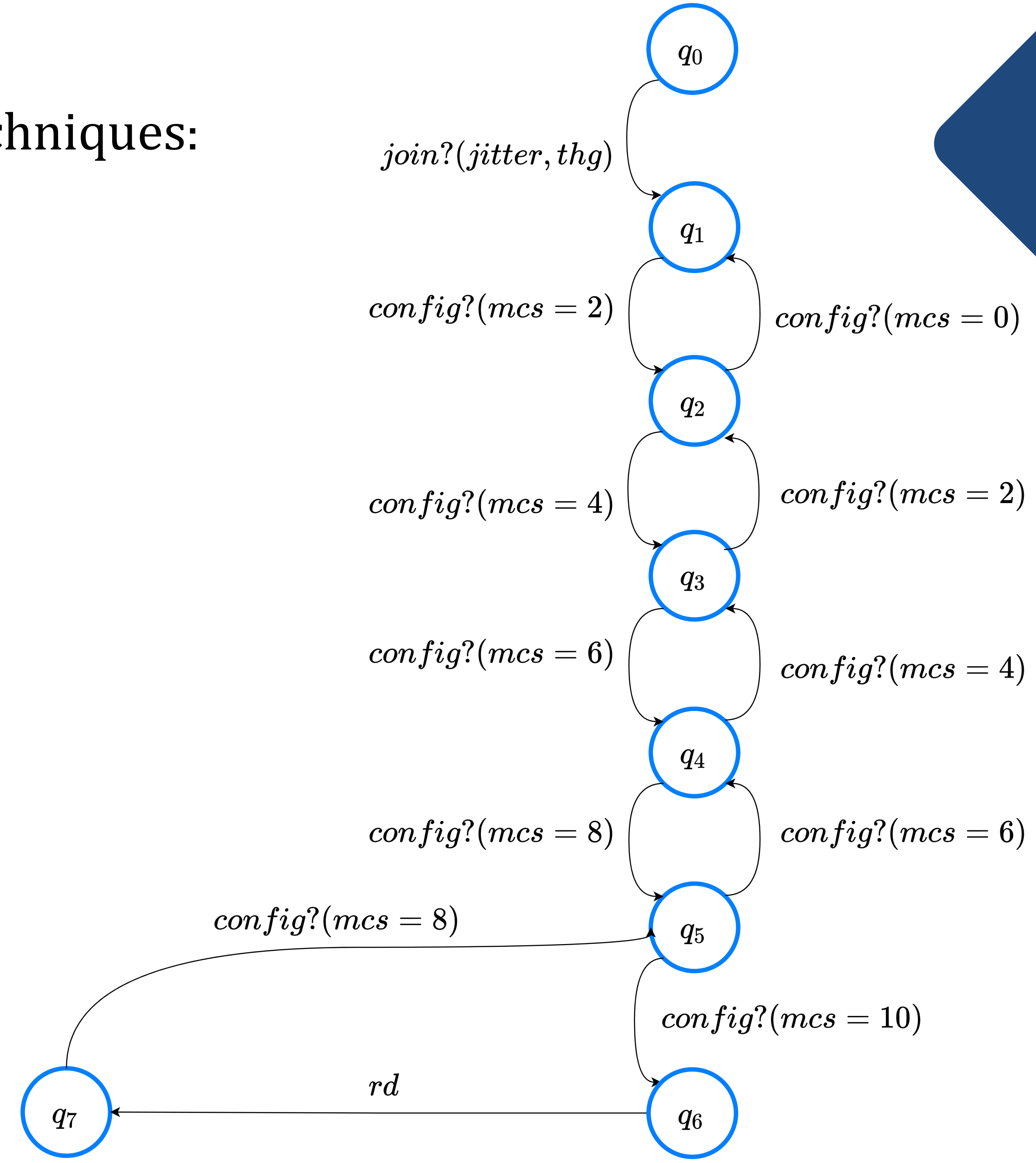
Automata Learning Approach

Advantages of automata learning compared to ML techniques:

- Behavioural model instead of a black box model

  → **decisions can be explained**

- Analysis of the automata:
  - Equivalence checking
  - Model composition and refinement



Automata Learning Approach

The Learn algorithm composes the automaton resulting from learning a finite set of k traces

Each trace $\pi = s_0 \cdot s_1 \cdot \cdots \cdot s_{n-1} \in \mathcal{T}$
is a sequence of observed states s_i

Each observed state is a tuple $\langle ts, conf, intd, obsd \rangle$

Algorithm Learn $(\downarrow \mathcal{T} = \{\pi^1, \cdots, \pi^k\}, \uparrow \mathcal{A}_k)$:
- $\mathcal{A}_0 := \langle \{q_0\}, q_0, \emptyset, \{q_0 \to (conf_d, \bar{\iota}, \bar{0})\} \rangle$;
- **for** $i := 1 \dots k$ **do**
  - $\mathcal{A}_i := Compose(\mathcal{A}_{i-1}, \pi^i)$;
- **return** $\mathcal{A}_k$;

Observed states are produced:

- periodically during network execution *rd*
- when events occurs:
  - *config*(*conf*)*?* → change of configuration requested
  - *join*(*intd*)? → new TSN session with *intd* KPIs
  - *leave* ? → a existing TSN session is closed

Automata Learning Approach

For each state of the trace, the function Compose considers 4 different cases:

1. The observed state is produced by a change in the network configuration

case $config(conf_n)?$ **do**
  if $conf_n \neq conf_c$ **then**
    $q' := \mathcal{A}.get(conf_n, it, \bar{0});$
    if $q' = None$ **then**
      $q' := \mathcal{A}.newState(conf_n, it, \bar{0});$
    $\mathcal{A}.addTran(q_c, config(conf_n)?, q');$
    $q_c := q';$

Function Compose($\updownarrow \mathcal{A}, \downarrow \pi = s_0 \cdot \dots \cdot s_{n-1}$):
  $q_c := q_0;$
  for $i := 0 \dots n-1$ **do**
    $(conf_c, it, ob^\alpha) := f_{\mathcal{A}}(q_c);$
    switch s_i **do**
      case $config(conf_n)?$ **do**
        if $conf_n \neq conf_c$ **then**
          $q' := \mathcal{A}.get(conf_n, it, \bar{0});$
          if $q' = None$ **then**
            $q' := \mathcal{A}.newState(conf_n, it, \bar{0});$
          $\mathcal{A}.addTran(q_c, config(conf_n)?, q');$
          $q_c := q';$
      case $join(it')?$ **do**
        $q' := \mathcal{A}.get(conf_c, it', \bar{0});$
        if $q' = None$ **then**
          $q' = \mathcal{A}.newState(conf_c, it', \bar{0});$
        if $q' \neq q_c$ **then**
          $\mathcal{A}.addTran(q_c, join(it')?, q');$
          $q_c = q';$
      case $leave?$ **do**
        $\mathcal{A}.addTran(q_c, leave?, q_0);$
        $q_c := q_0;$
      case $\langle ts, conf_c, it', ob' \rangle$ **do**
        $o^\alpha = deviation(it', ob');$
        $q' := \mathcal{A}.get(conf_c, it', o^\alpha);$
        if $q' = None$ **then**
          $q' := \mathcal{A}.newState(conf_c, it', o^\alpha);$
        if $q' \neq q_c$ **then**
          $\mathcal{A}.addTran(q_c, rd, q');$
          $q_c := q';$
  return $\mathcal{A}$;

Automata Learning Approach

For each state of the trace, the function Compose considers 4 different cases:

1. The observed state is produced by a change in the network configuration
2. The observed state is produced by a new session

case $join(it')?$ **do**
  $q' := \mathcal{A}.get(conf_c, it', \bar{0});$
  if $q' = None$ **then**
    $q' = \mathcal{A}.newState(conf_c, it', \bar{0});$
  if $q' \neq q_c$ **then**
    $\mathcal{A}.addTran(q_c, join(it')?, q');$
    $q_c = q';$

Function Compose($\updownarrow \mathcal{A}, \downarrow \pi = s_0 \cdot \dots \cdot s_{n-1}$):
  $q_c := q_0;$
  for $i := 0 \dots n-1$ **do**
    $(conf_c, it, ob^\alpha) := f_{\mathcal{A}}(q_c);$
    switch s_i **do**
      case $config(conf_n)?$ **do**
        if $conf_n \neq conf_c$ **then**
          $q' := \mathcal{A}.get(conf_n, it, \bar{0});$
          if $q' = None$ **then**
            $q' := \mathcal{A}.newState(conf_n, it, \bar{0});$
          $\mathcal{A}.addTran(q_c, config(conf_n)?, q');$
          $q_c := q';$
      case $join(it')?$ **do**
        $q' := \mathcal{A}.get(conf_c, it', \bar{0});$
        if $q' = None$ **then**
          $q' = \mathcal{A}.newState(conf_c, it', \bar{0});$
        if $q' \neq q_c$ **then**
          $\mathcal{A}.addTran(q_c, join(it')?, q');$
          $q_c = q';$
      case $leave?$ **do**
        $\mathcal{A}.addTran(q_c, leave?, q_0);$
        $q_c := q_0;$
      case $\langle ts, conf_c, it', ob' \rangle$ **do**
        $o^\alpha = deviation(it', ob');$
        $q' := \mathcal{A}.get(conf_c, it', o^\alpha);$
        if $q' = None$ **then**
          $q' := \mathcal{A}.newState(conf_c, it', o^\alpha);$
        if $q' \neq q_c$ **then**
          $\mathcal{A}.addTran(q_c, rd, q');$
          $q_c := q';$
  return $\mathcal{A};$

Automata Learning Approach

For each state of the trace, the function Compose considers 4 different cases:

1. The observed state is produced by a change in the network configuration
2. The observed state is produced by a new session
3. The observed state is produced by the end of a session

case $leave?$ **do**
    $\mathcal{A}.addTran(q_c, leave?, q_0);$
    $q_c := q_0;$

Function Compose($\updownarrow \mathcal{A}, \downarrow \pi = s_0 \cdot \dots \cdot s_{n-1}$):
  $q_c := q_0;$
  for $i := 0 \dots n-1$ **do**
    $(conf_c, it, ob^\alpha) := f_{\mathcal{A}}(q_c);$
    switch s_i **do**
      case $config(conf_n)?$ **do**
        if $conf_n \neq conf_c$ **then**
          $q' := \mathcal{A}.get(conf_n, it, \bar{0});$
          if $q' = None$ **then**
            $q' := \mathcal{A}.newState(conf_n, it, \bar{0});$
          $\mathcal{A}.addTran(q_c, config(conf_n)?, q');$
          $q_c := q';$
      case $join(it')?$ **do**
        $q' := \mathcal{A}.get(conf_c, it', \bar{0});$
        if $q' = None$ **then**
          $q' = \mathcal{A}.newState(conf_c, it', \bar{0});$
        if $q' \neq q_c$ **then**
          $\mathcal{A}.addTran(q_c, join(it')?, q');$
          $q_c = q';$
      case $leave?$ **do**
        $\mathcal{A}.addTran(q_c, leave?, q_0);$
        $q_c := q_0;$
      case $\langle ts, conf_c, it', ob' \rangle$ **do**
        $o^\alpha = deviation(it', ob');$
        $q' := \mathcal{A}.get(conf_c, it', o^\alpha);$
        if $q' = None$ **then**
          $q' := \mathcal{A}.newState(conf_c, it', o^\alpha);$
        if $q' \neq q_c$ **then**
          $\mathcal{A}.addTran(q_c, rd, q');$
          $q_c := q';$
  return $\mathcal{A}$;

Automata Learning Approach

For each state of the trace, the function Compose considers 4 different cases:

1. The observed state is produced by a change in the network configuration
2. The observed state is produced by a new session
3. The observed state is produced by the end of a session
4. The observed state is produced by time pass

case $\langle ts, conf_c, it', ob' \rangle$ **do**
  $o^\alpha = deviation(it', ob')$;
  $q' := \mathcal{A}.get(conf_c, it', o^\alpha)$;
  if $q' = None$ **then**
    $q' := \mathcal{A}.newState(conf_c, it', o^\alpha)$;
  if $q' \neq q_c$ **then**
    $\mathcal{A}.addTran(q_c, rd, q')$;
    $q_c := q'$;

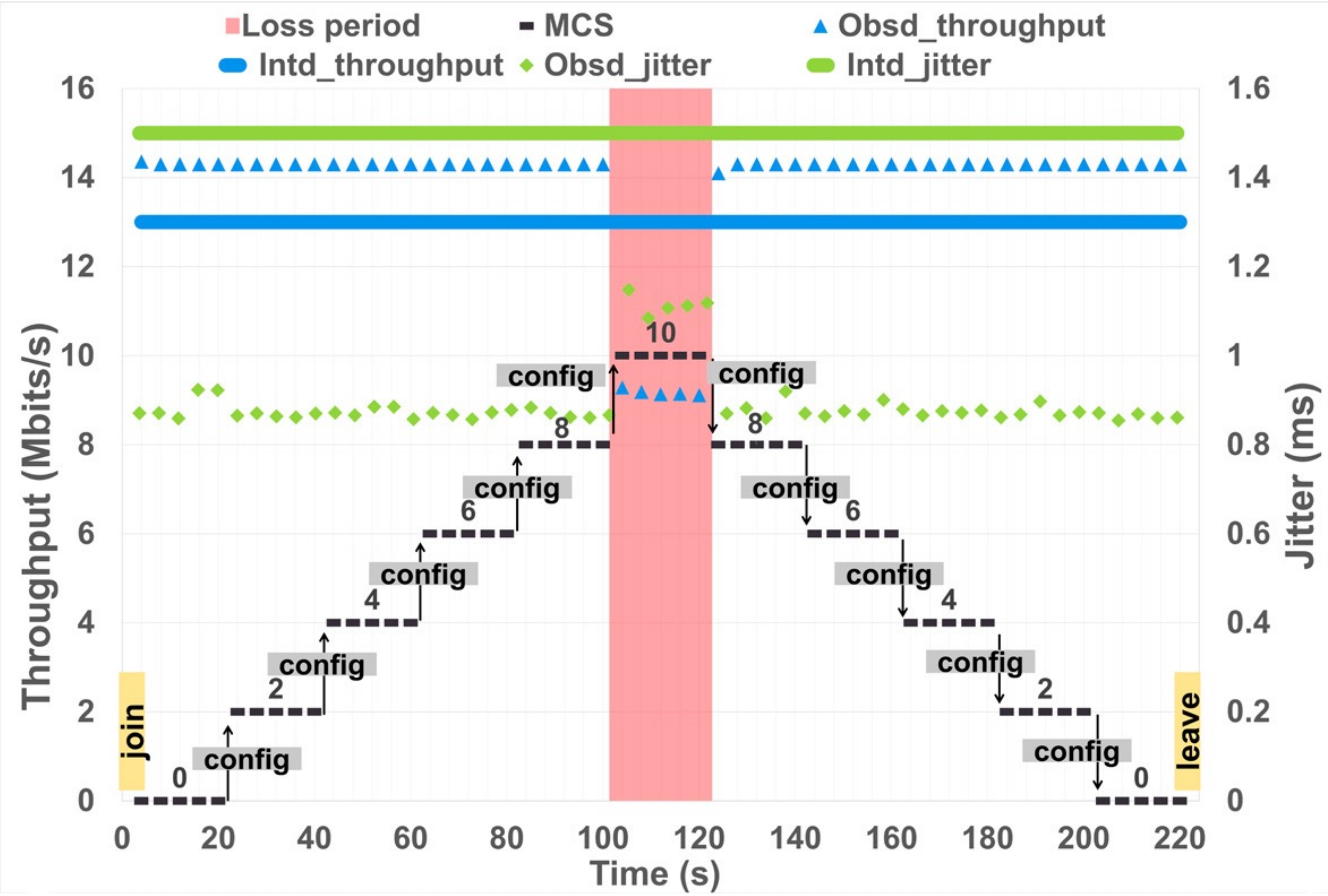
Function Compose($\updownarrow \mathcal{A}, \downarrow \pi = s_0 \cdot \dots \cdot s_{n-1}$):
  $q_c := q_0$;
  for $i := 0 \dots n-1$ **do**
    $(conf_c, it, ob^\alpha) := f_{\mathcal{A}}(q_c)$;
    switch s_i **do**
      case $config(conf_n)?$ **do**
        if $conf_n \neq conf_c$ **then**
          $q' := \mathcal{A}.get(conf_n, it, \bar{0})$;
          if $q' = None$ **then**
            $q' := \mathcal{A}.newState(conf_n, it, \bar{0})$;
          $\mathcal{A}.addTran(q_c, config(conf_n)?, q')$;
          $q_c := q'$;
      case $join(it')?$ **do**
        $q' := \mathcal{A}.get(conf_c, it', \bar{0})$;
        if $q' = None$ **then**
          $q' = \mathcal{A}.newState(conf_c, it', \bar{0})$;
        if $q' \neq q_c$ **then**
          $\mathcal{A}.addTran(q_c, join(it')?, q')$;
          $q_c = q'$;
      case $leave?$ **do**
        $\mathcal{A}.addTran(q_c, leave?, q_0)$;
        $q_c := q_0$;
      case $\langle ts, conf_c, it', ob' \rangle$ **do**
        $o^\alpha = deviation(it', ob')$;
        $q' := \mathcal{A}.get(conf_c, it', o^\alpha)$;
        if $q' = None$ **then**
          $q' := \mathcal{A}.newState(conf_c, it', o^\alpha)$;
        if $q' \neq q_c$ **then**
          $\mathcal{A}.addTran(q_c, rd, q')$;
          $q_c := q'$;
  return $\mathcal{A}$;

Construction of the learned automaton

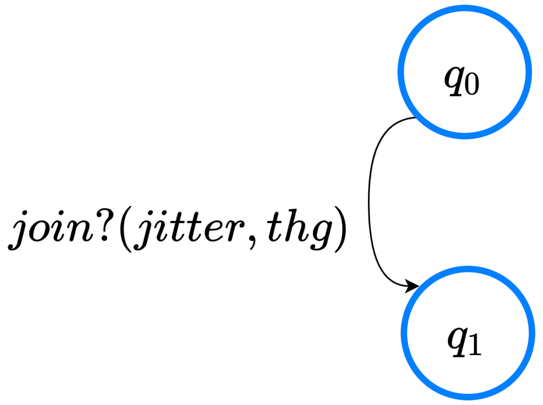
Sample trace:

- At time 0 sec. a new session starts (join) → Request max jitter and min throughput
- At time 220 sec. the session ends (leave)
- Each 20 sec. there is a configuration change
  - MCS in [0,10]
- **Each 4 sec. a new state is observed**

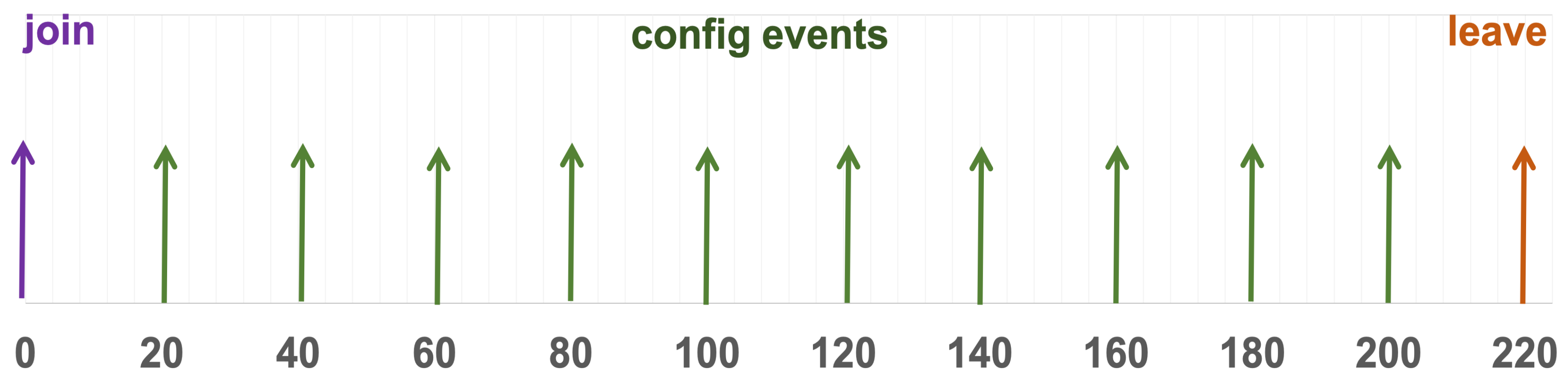
Given this sequence of events,
how the Learn Algorithm works ?

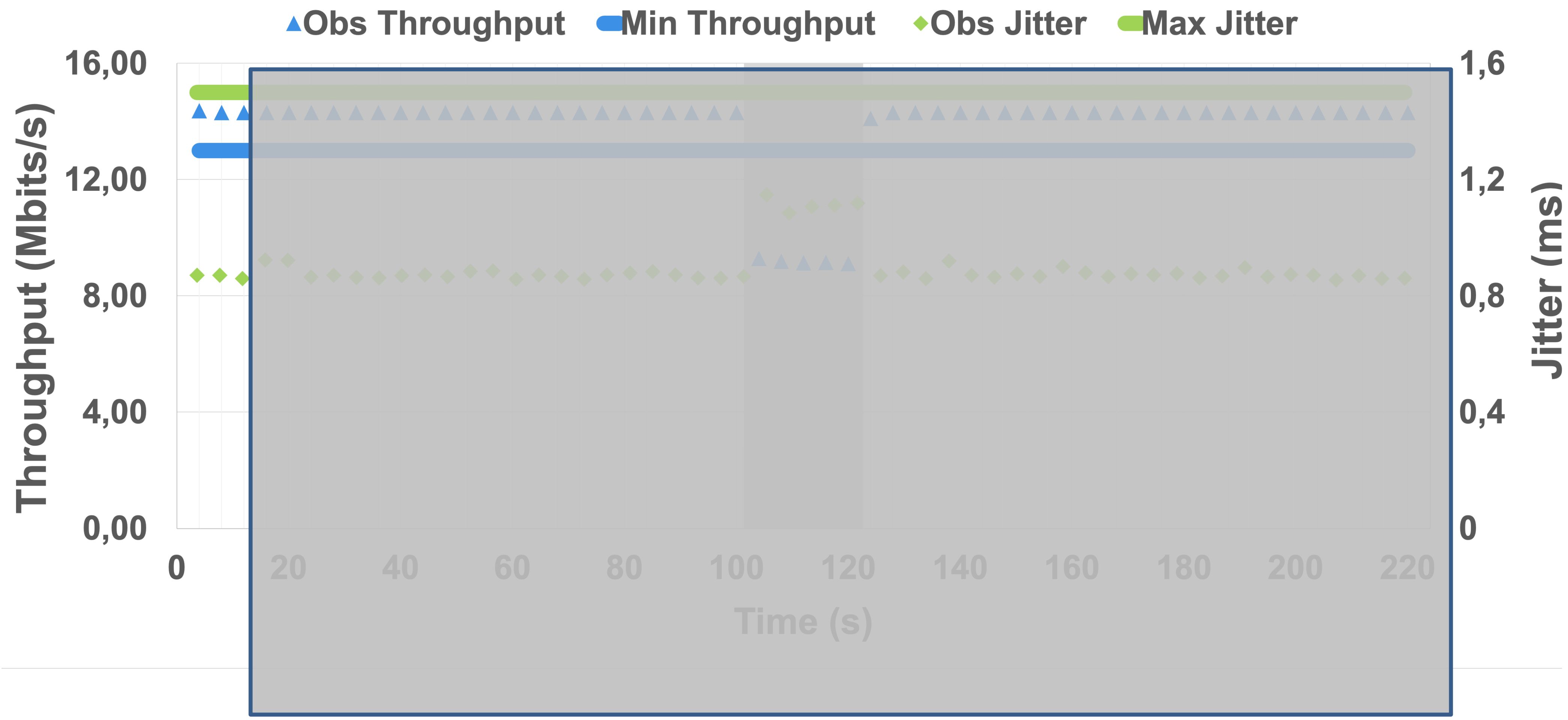


Construction of the learned automaton



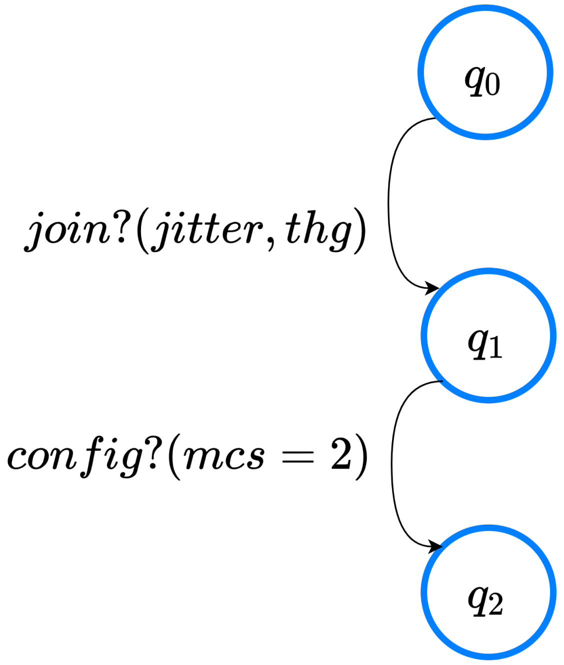
Initial state mcs = 0

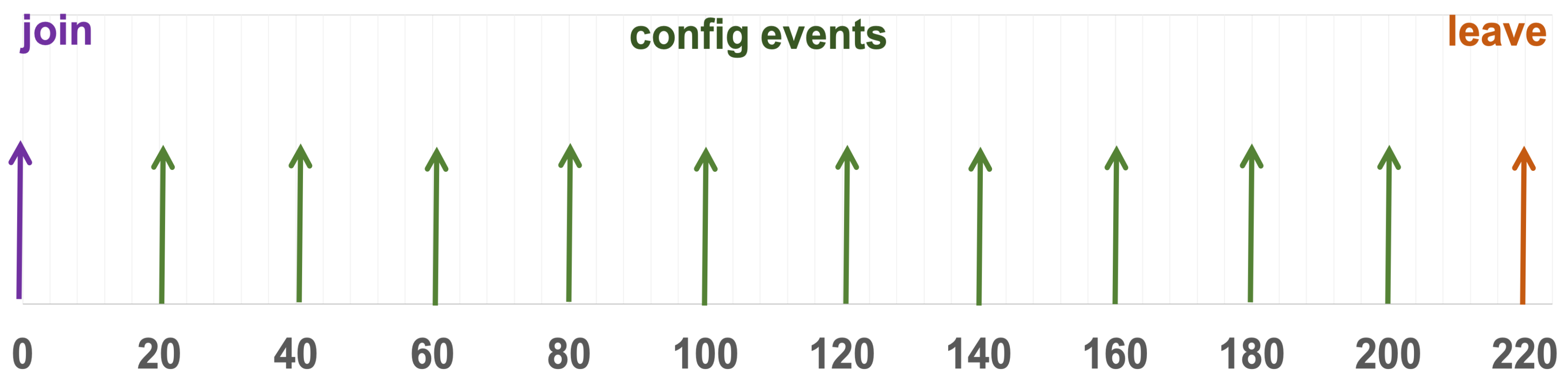


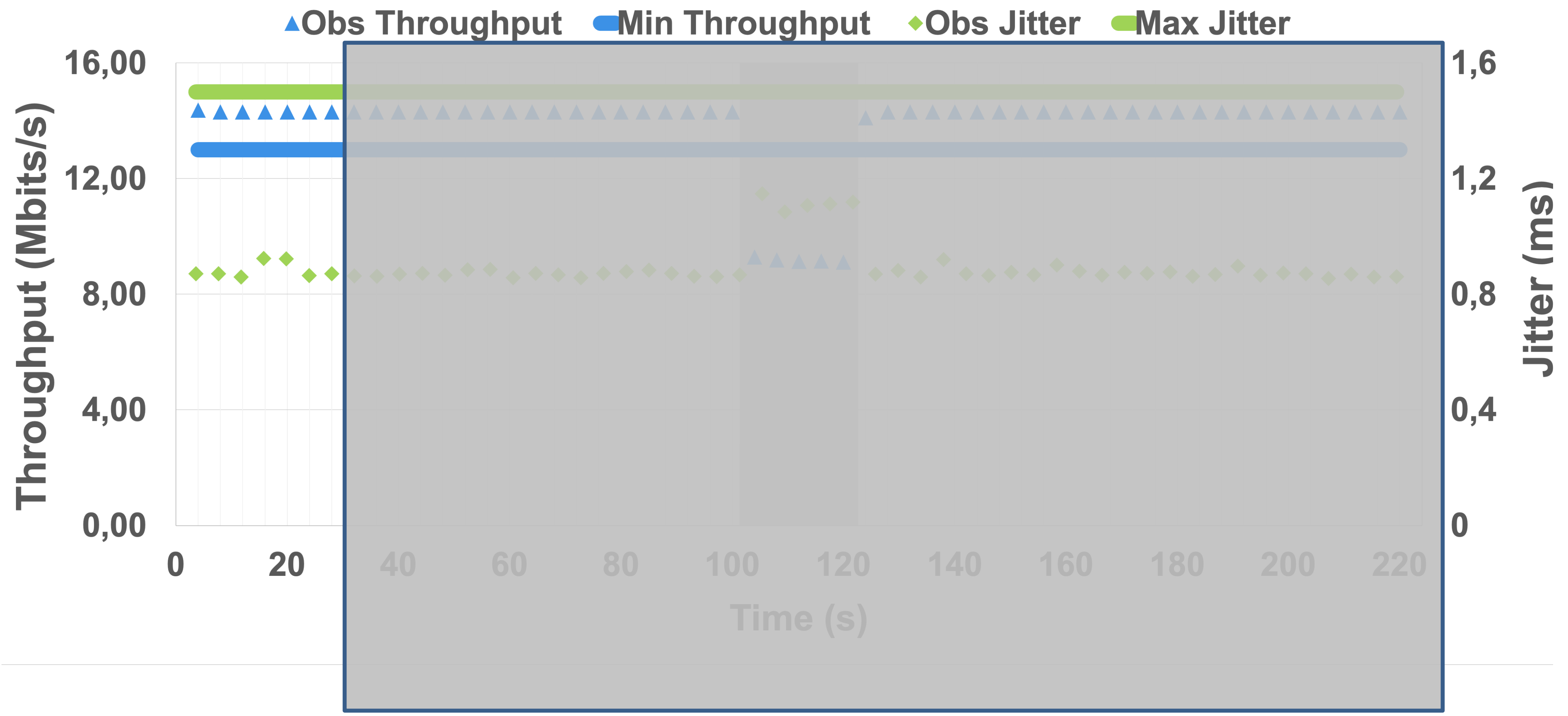


Construction of the learned automaton

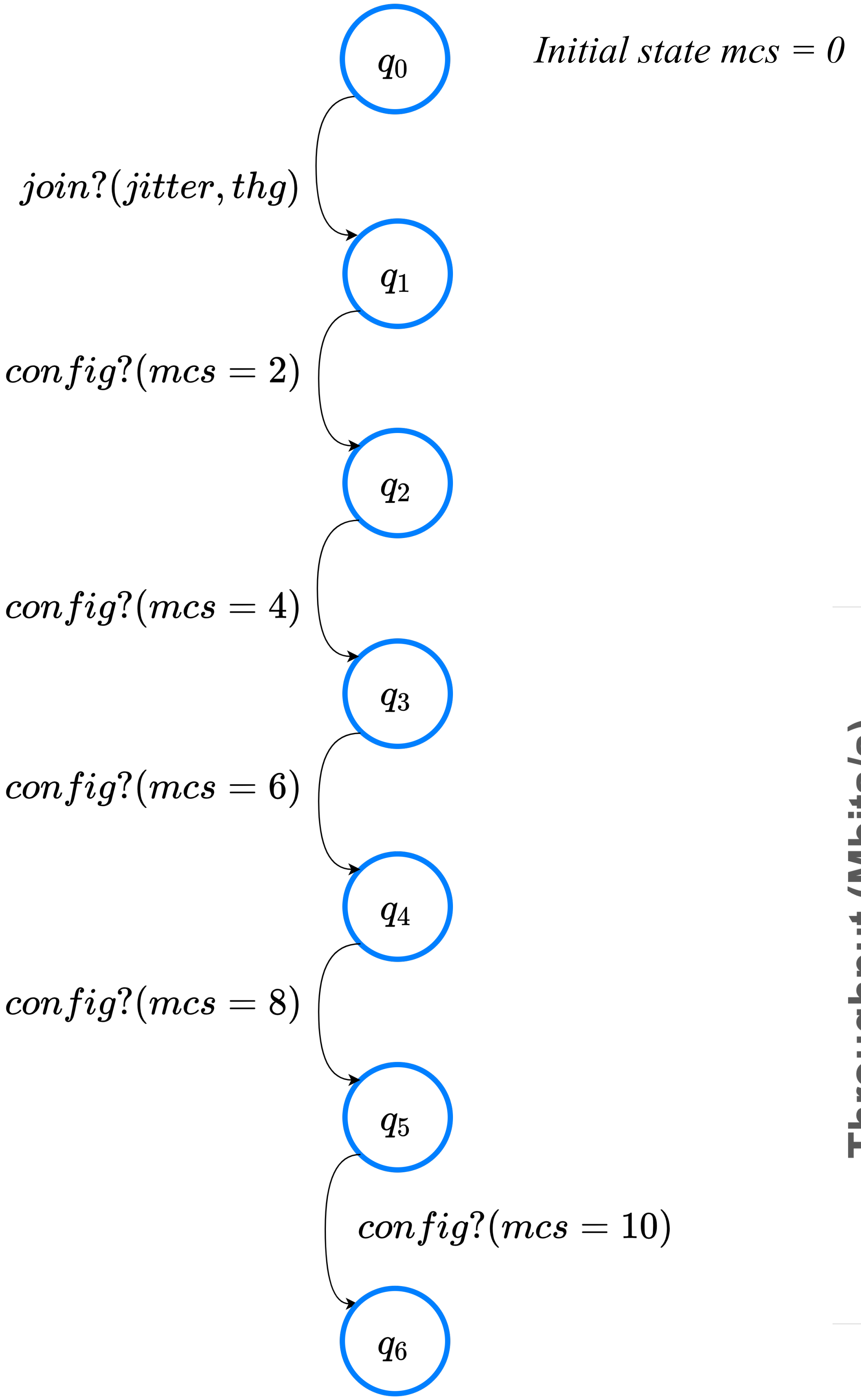
Initial state mcs = 0

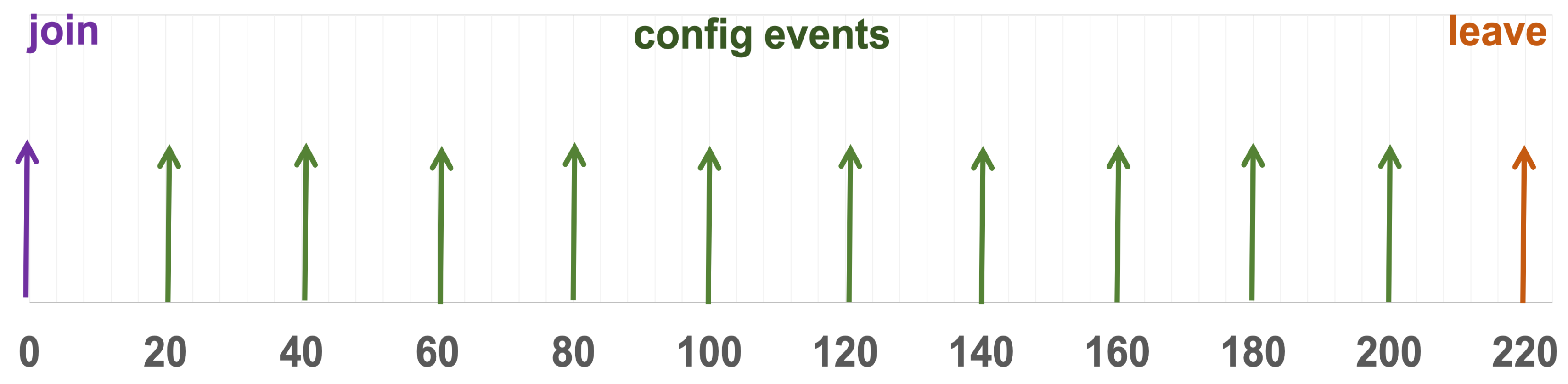


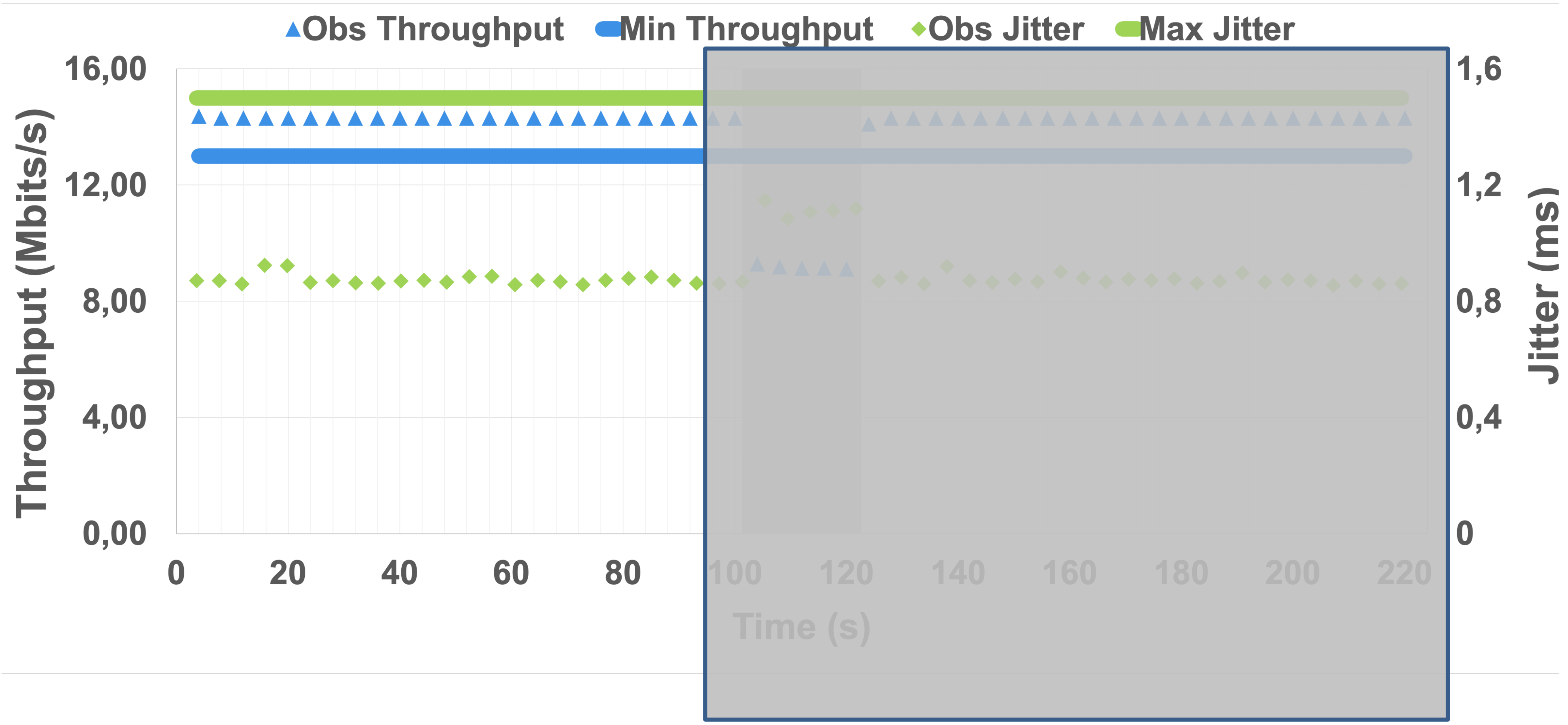




Construction of the learned automaton

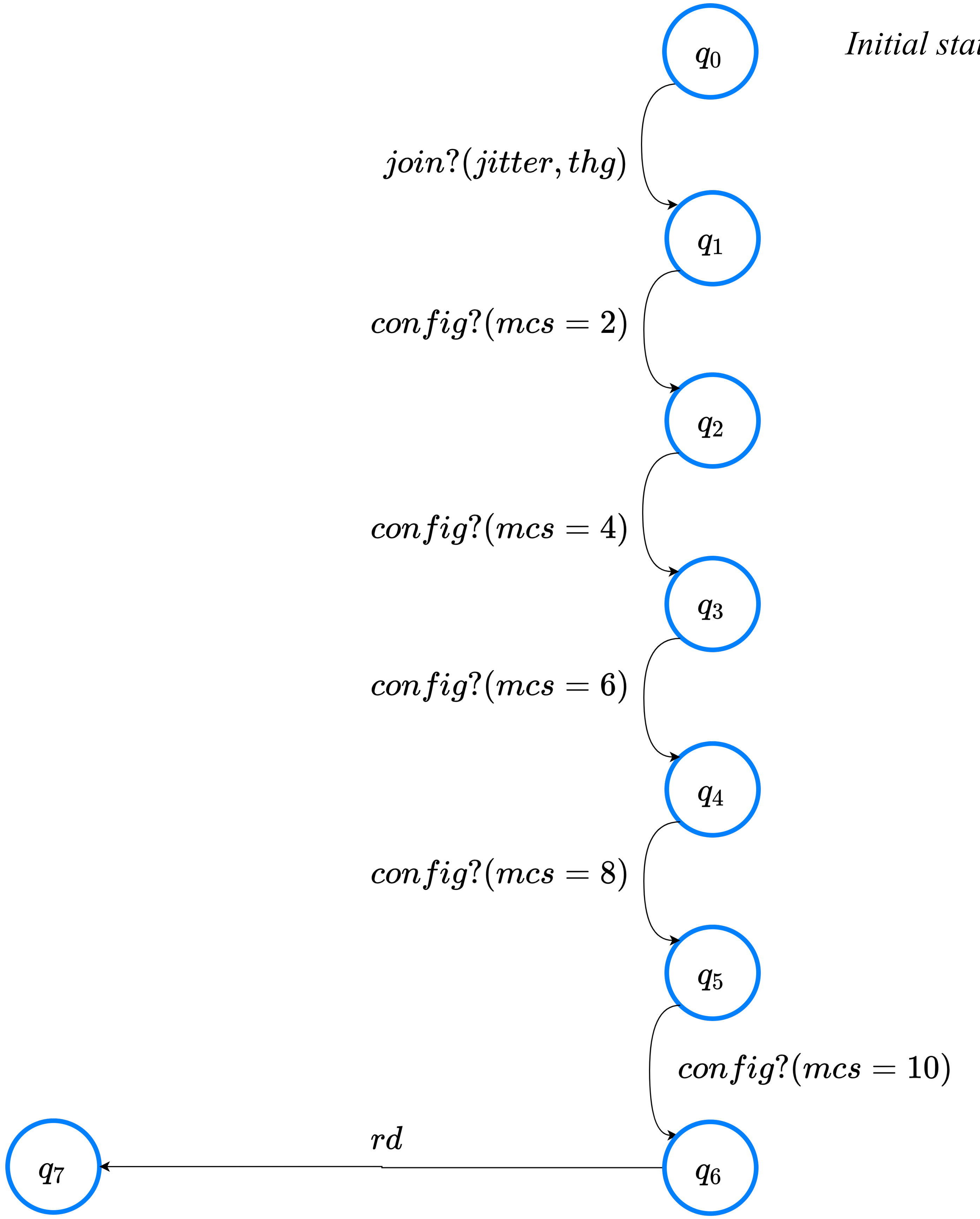


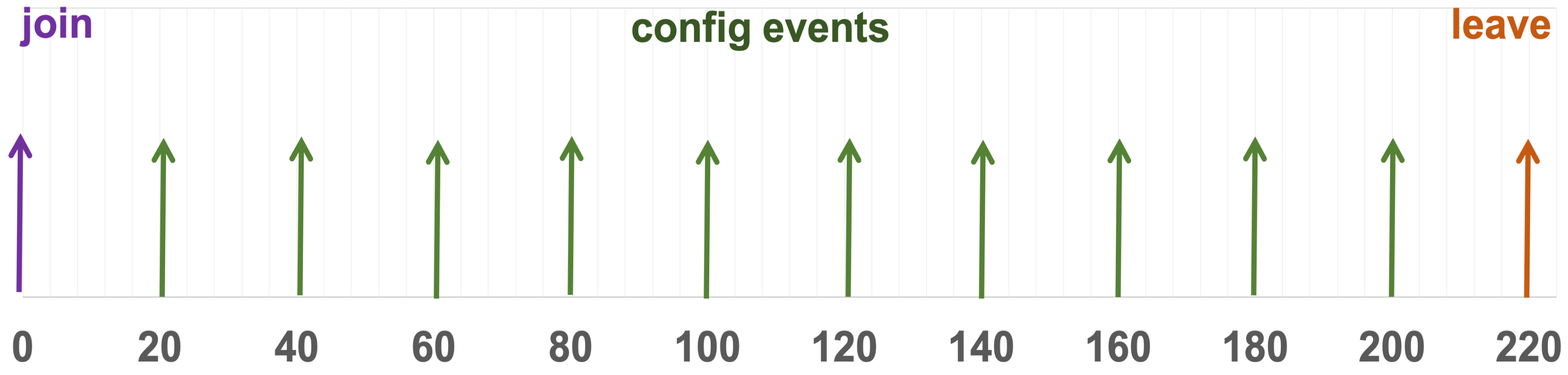


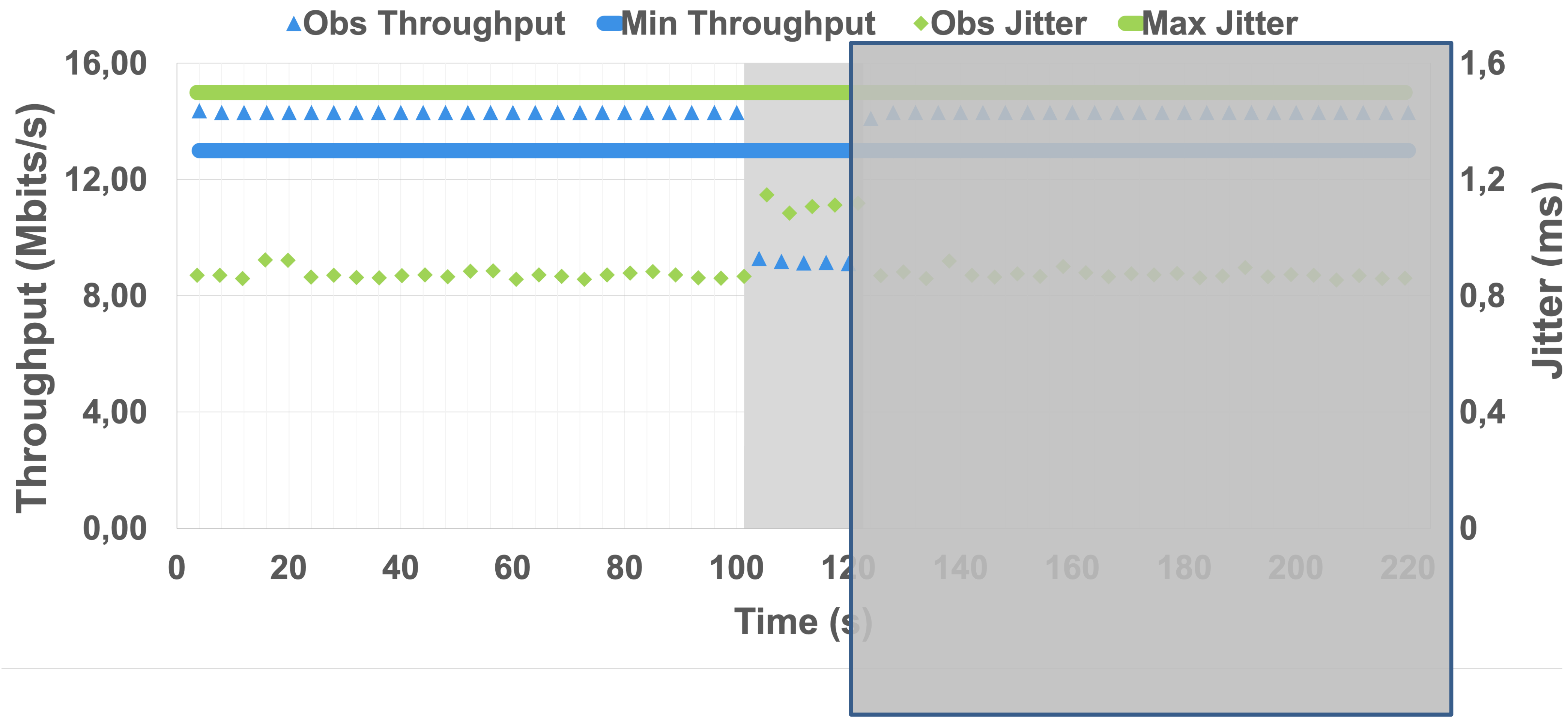


Construction of the learned automaton

Initial state mcs = 0

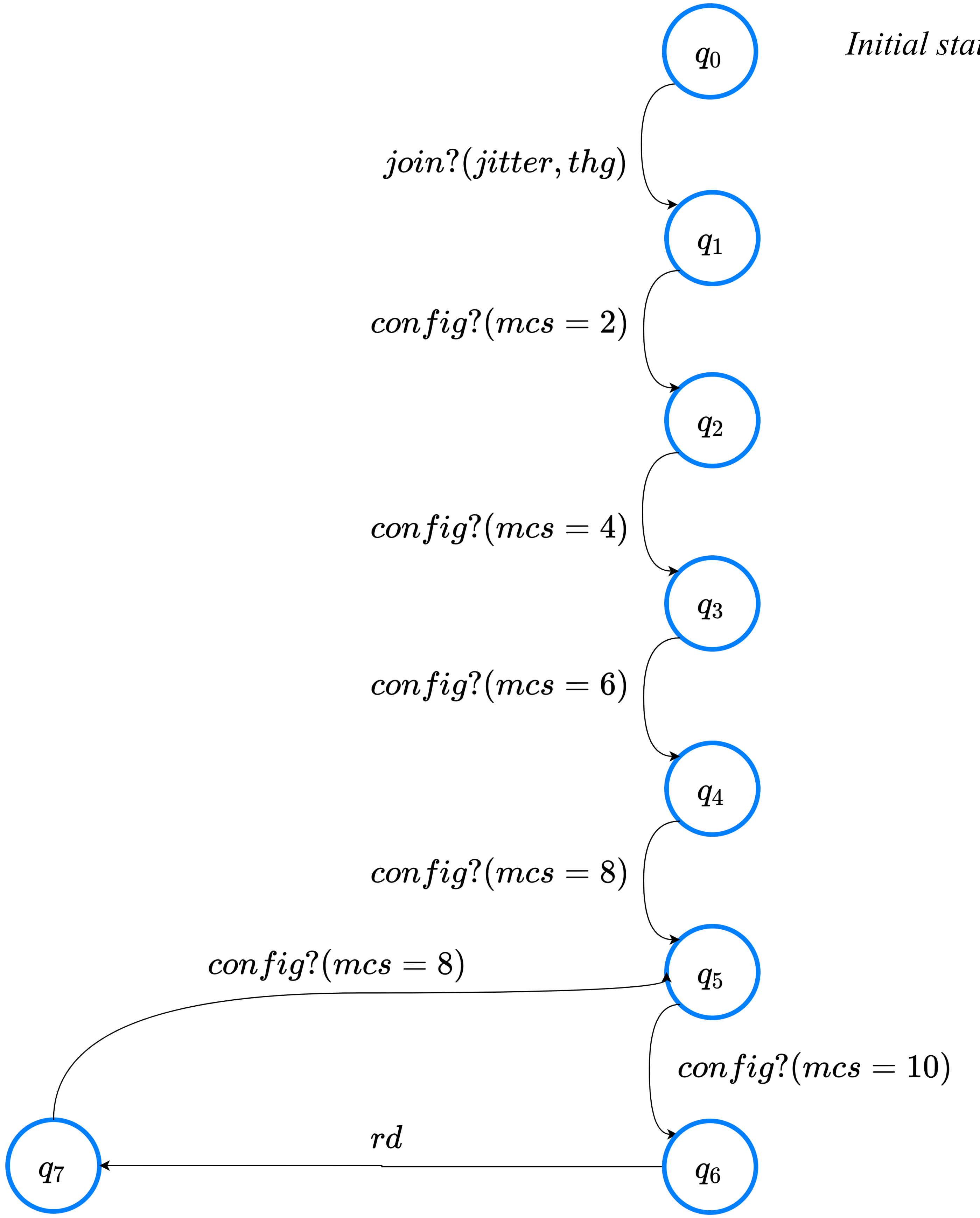


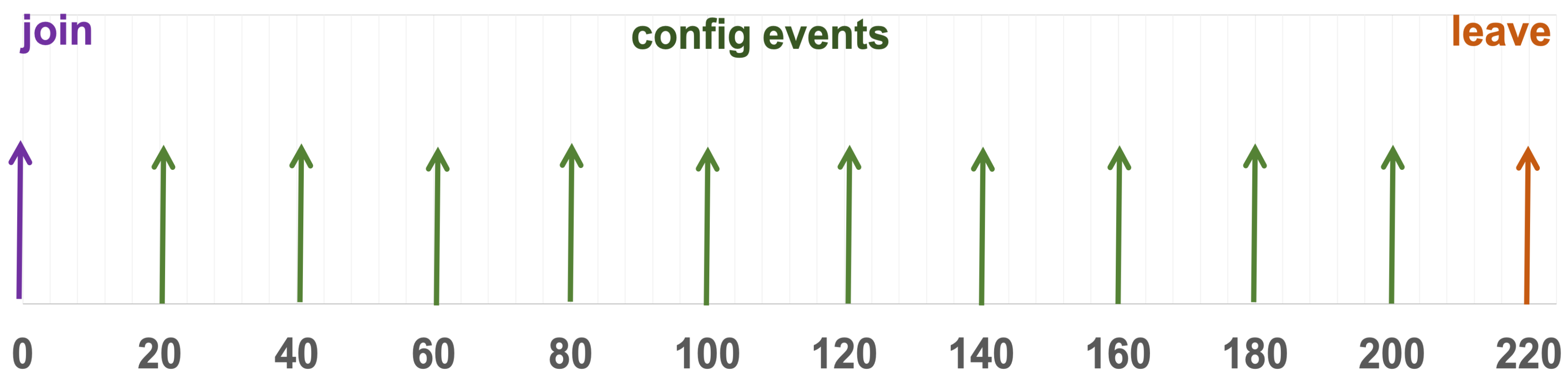


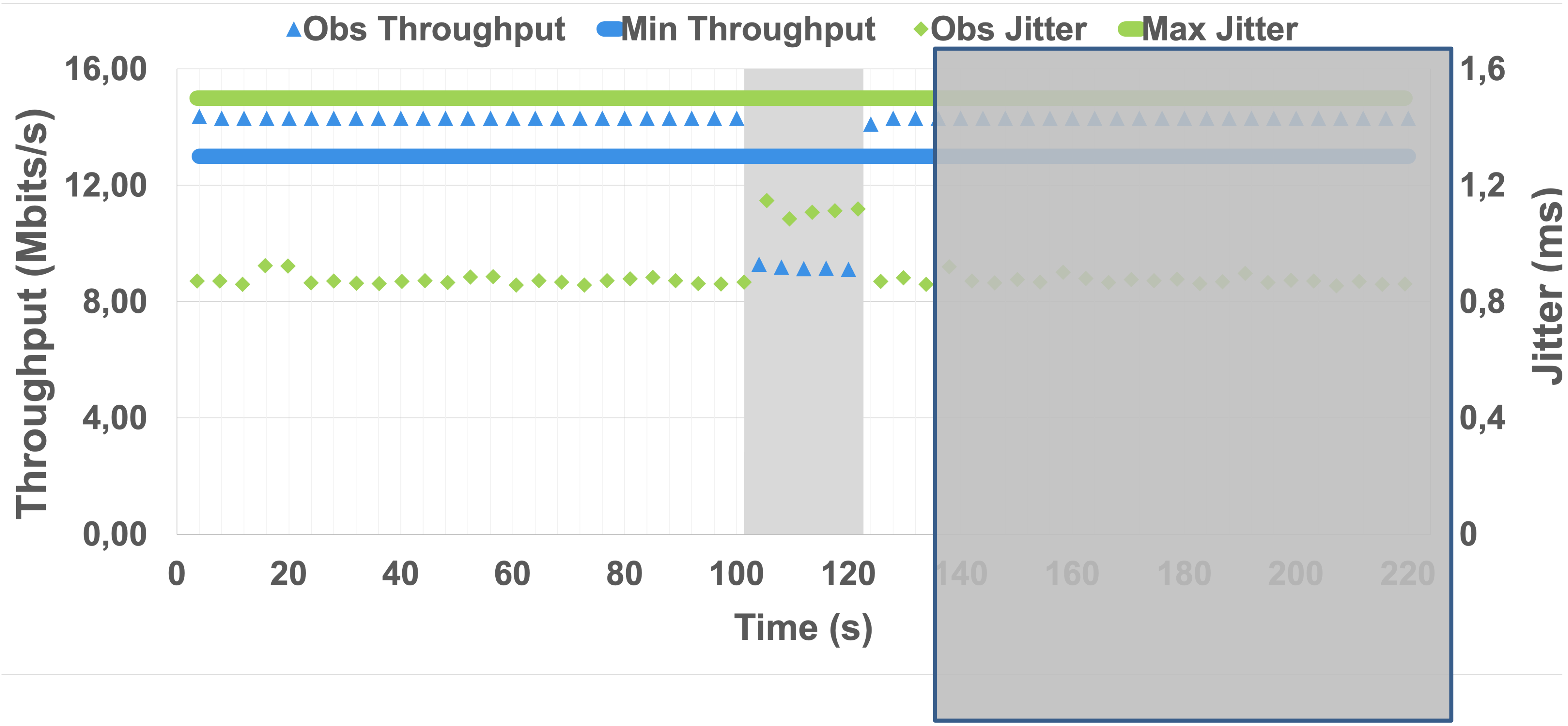


Construction of the learned automaton

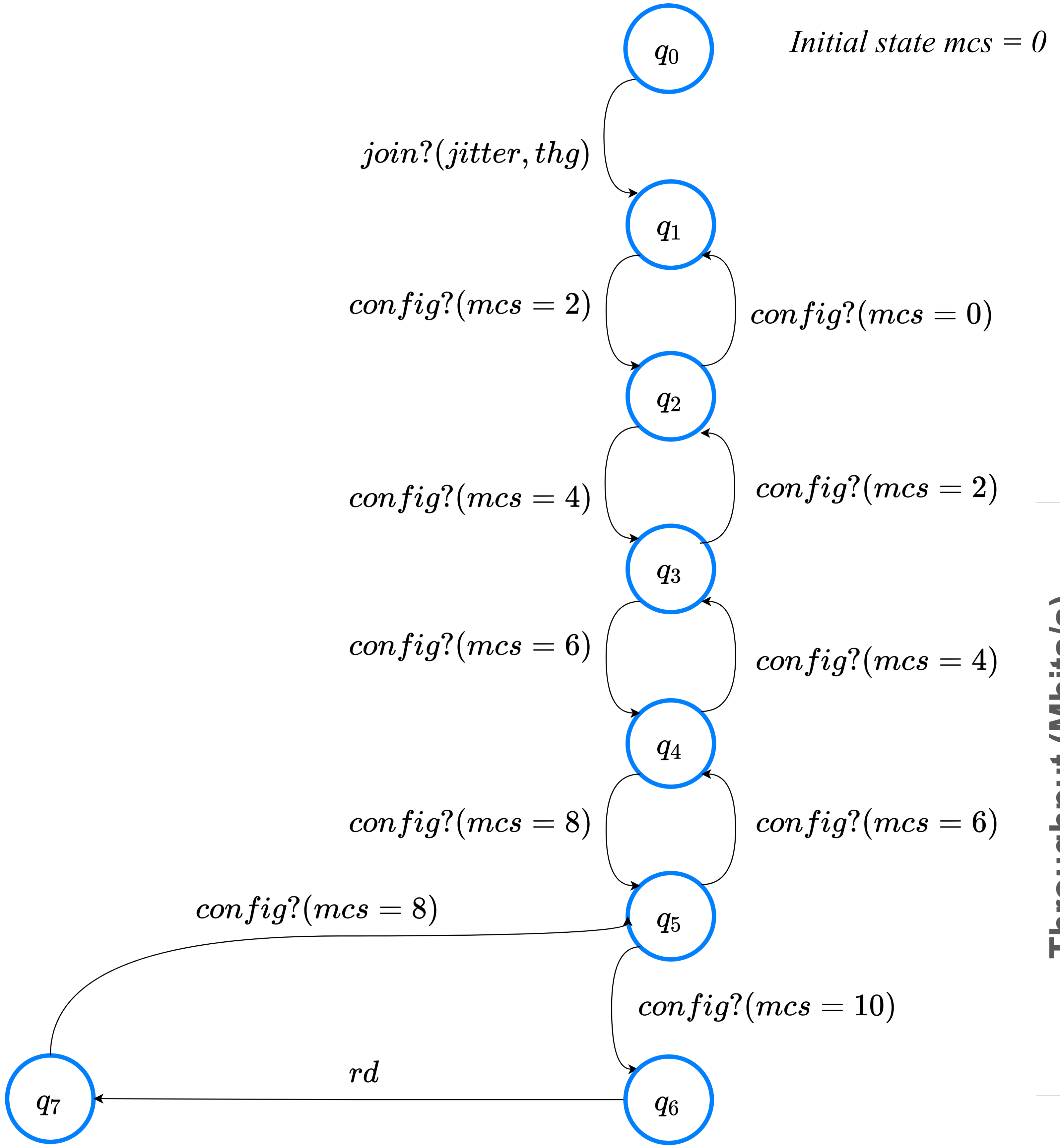
Initial state mcs = 0

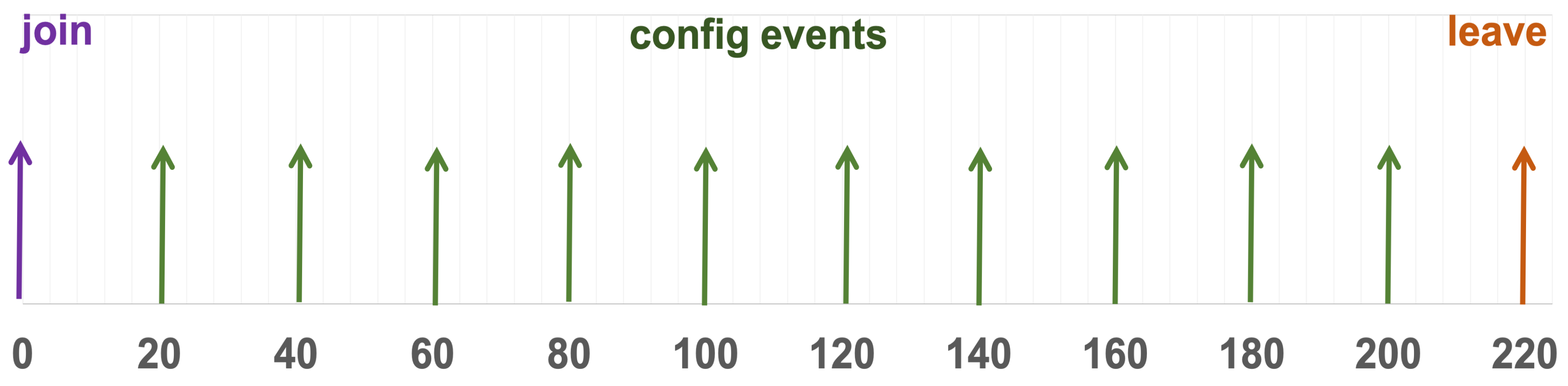


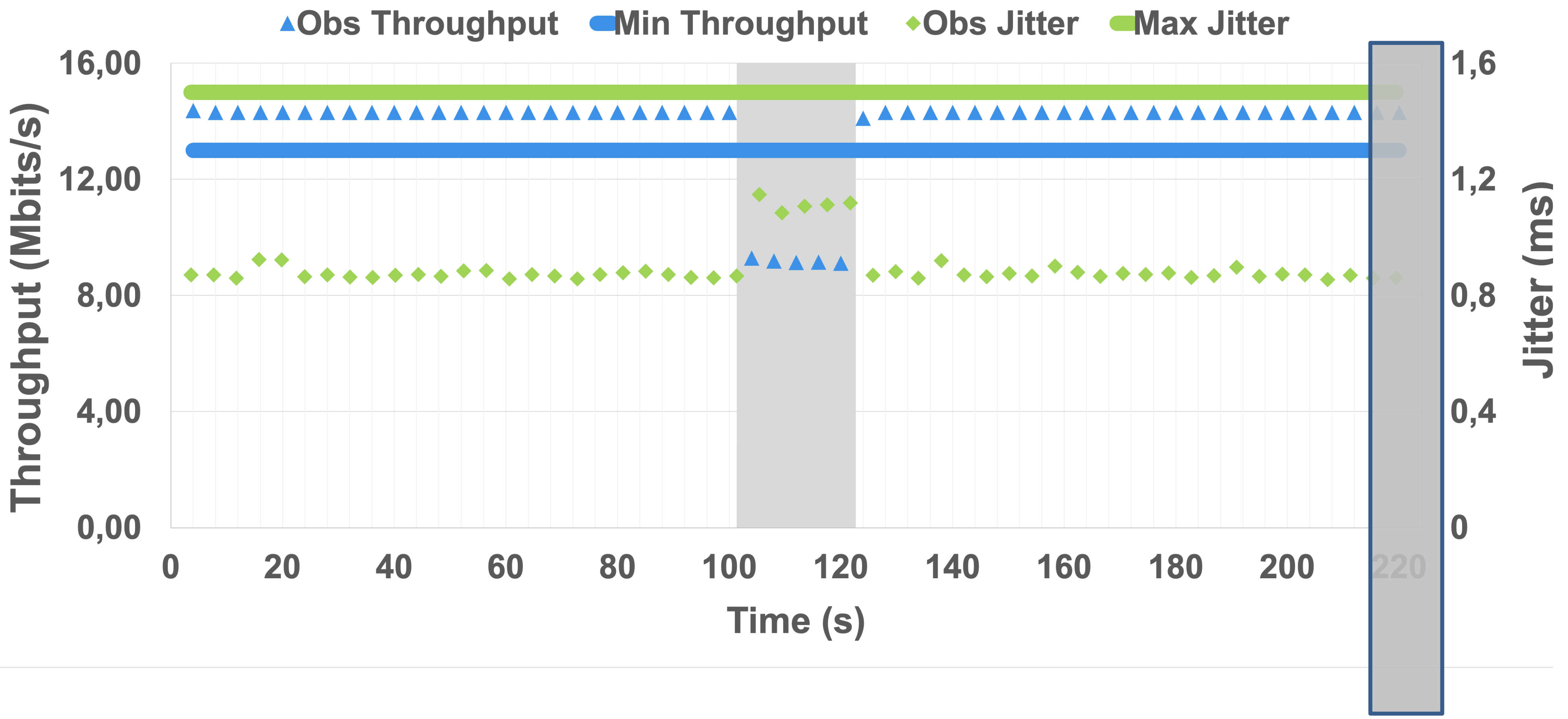




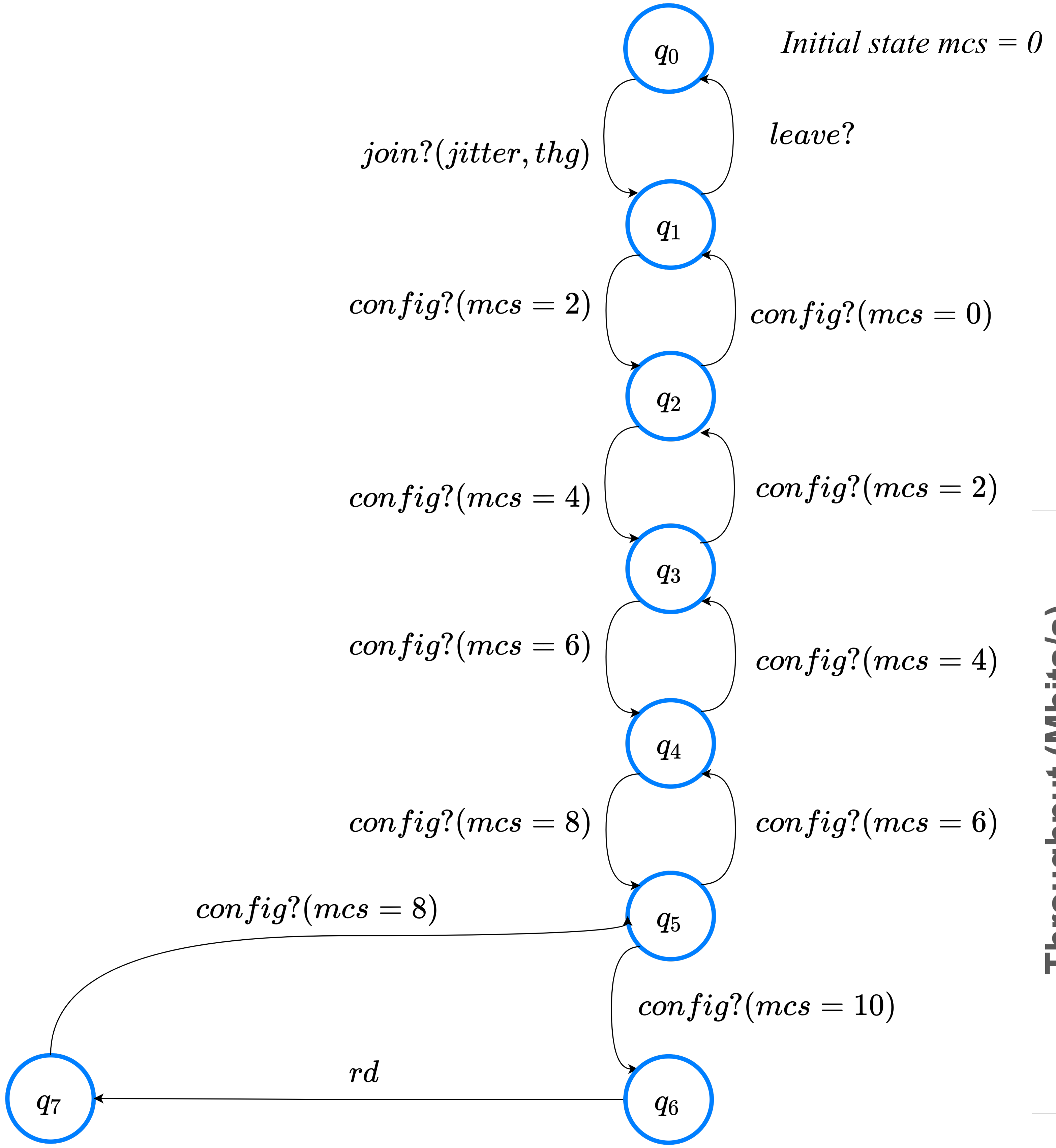
Construction of the learned automaton

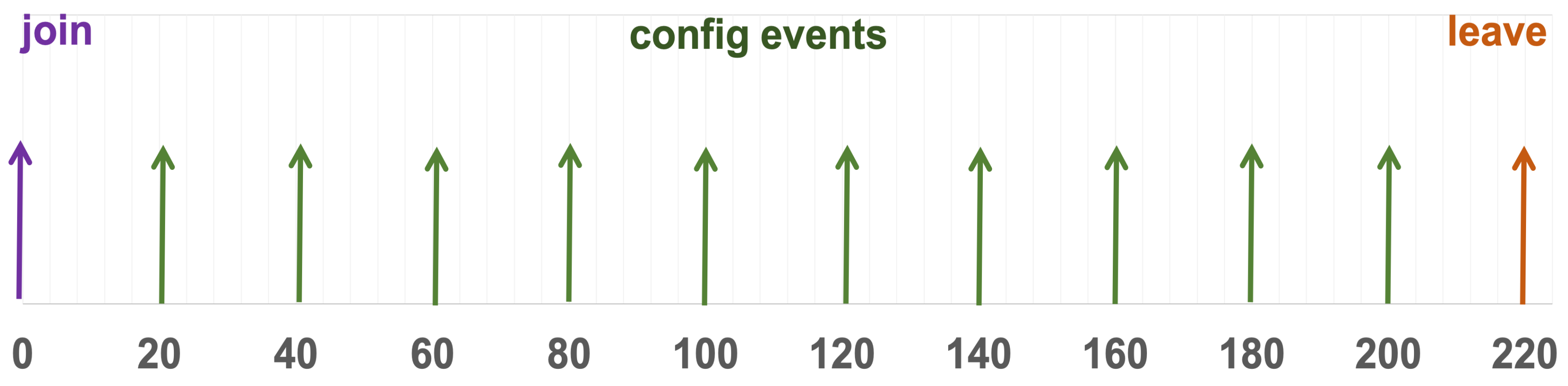


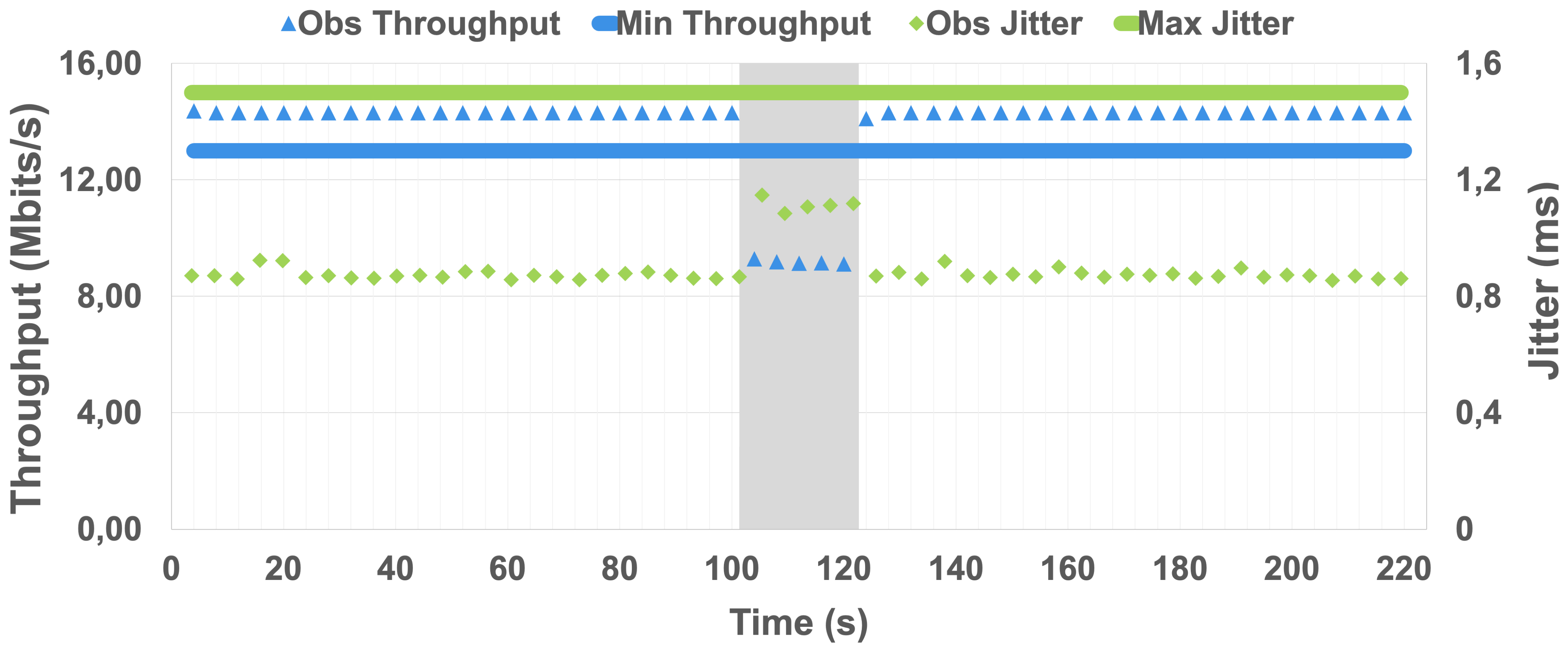




Construction of the learned automaton

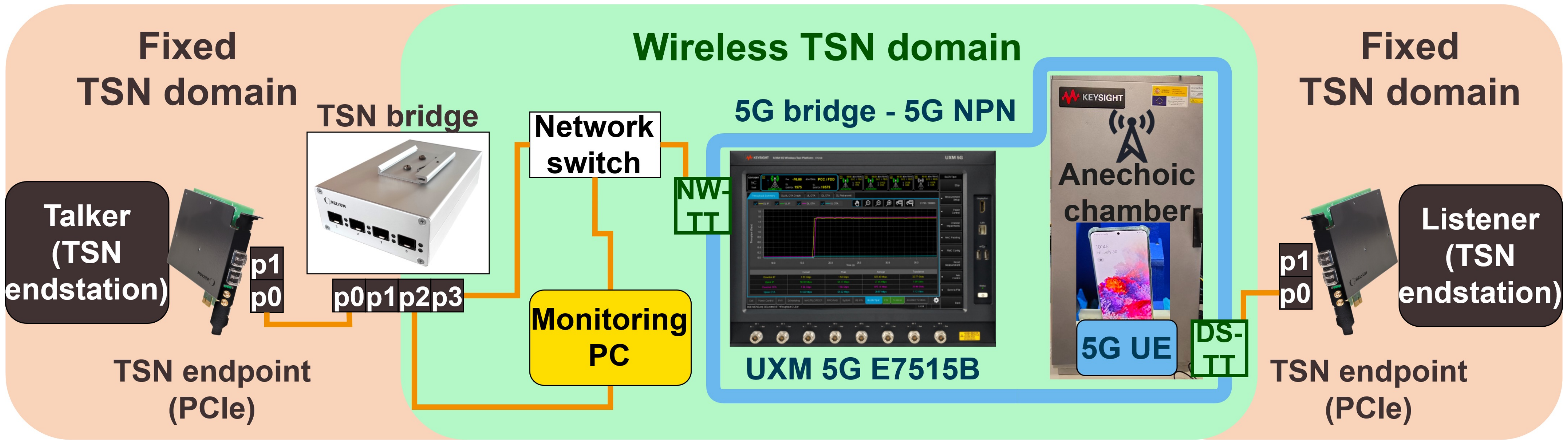






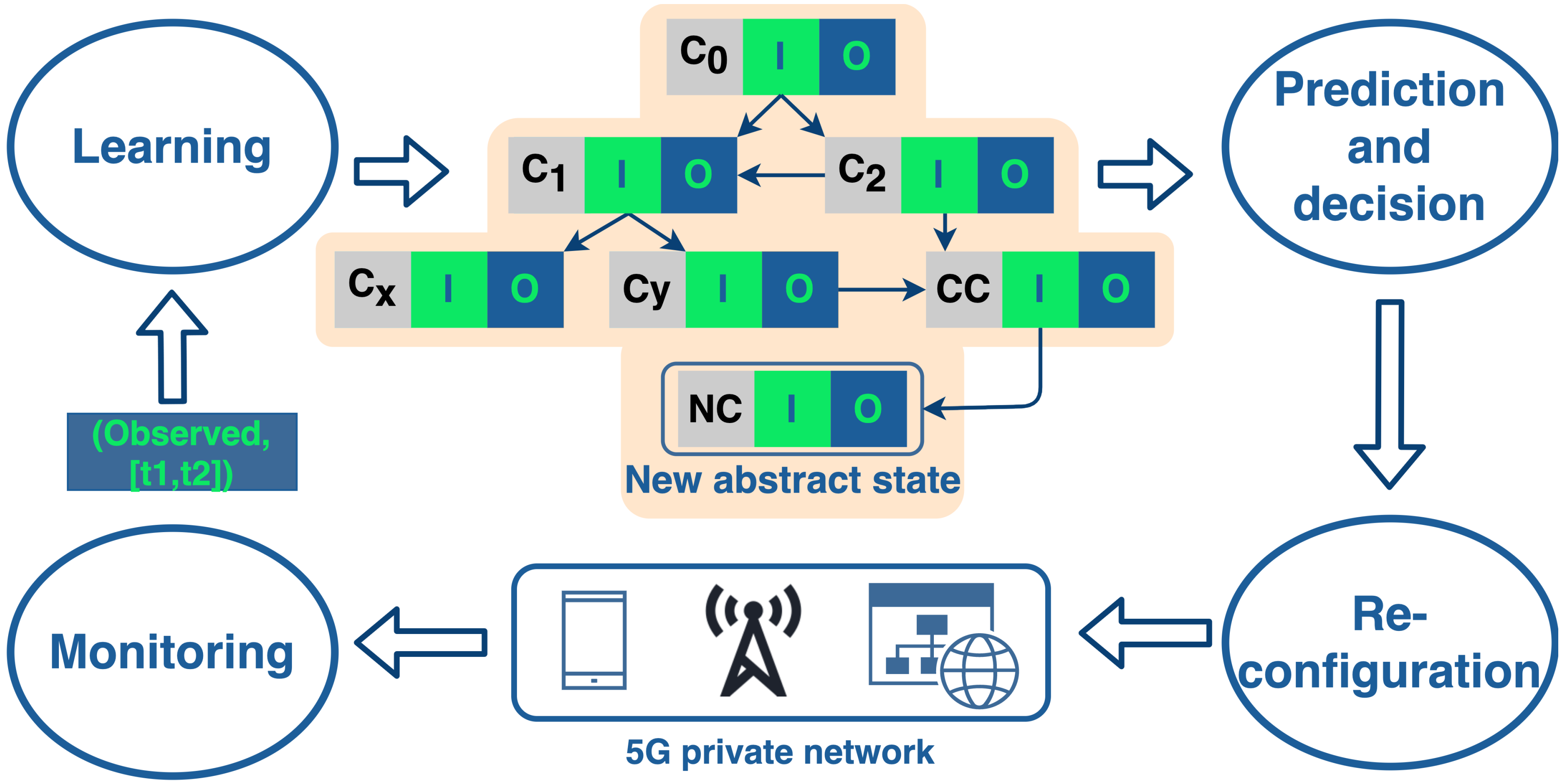
Experiments

- Learning algorithm implemented in Scala and Monitoring module with C
- Experiments:
  - Several hours over a realistic network
  - Considering only 2 endpoints
  - KPIs considered: throughput and jitter
  - Configuration parameters considered: **MCS**, PRB and Transmission Power
- Results:
  - Network traffic can be processed by the learning module to produce an automaton on real-time



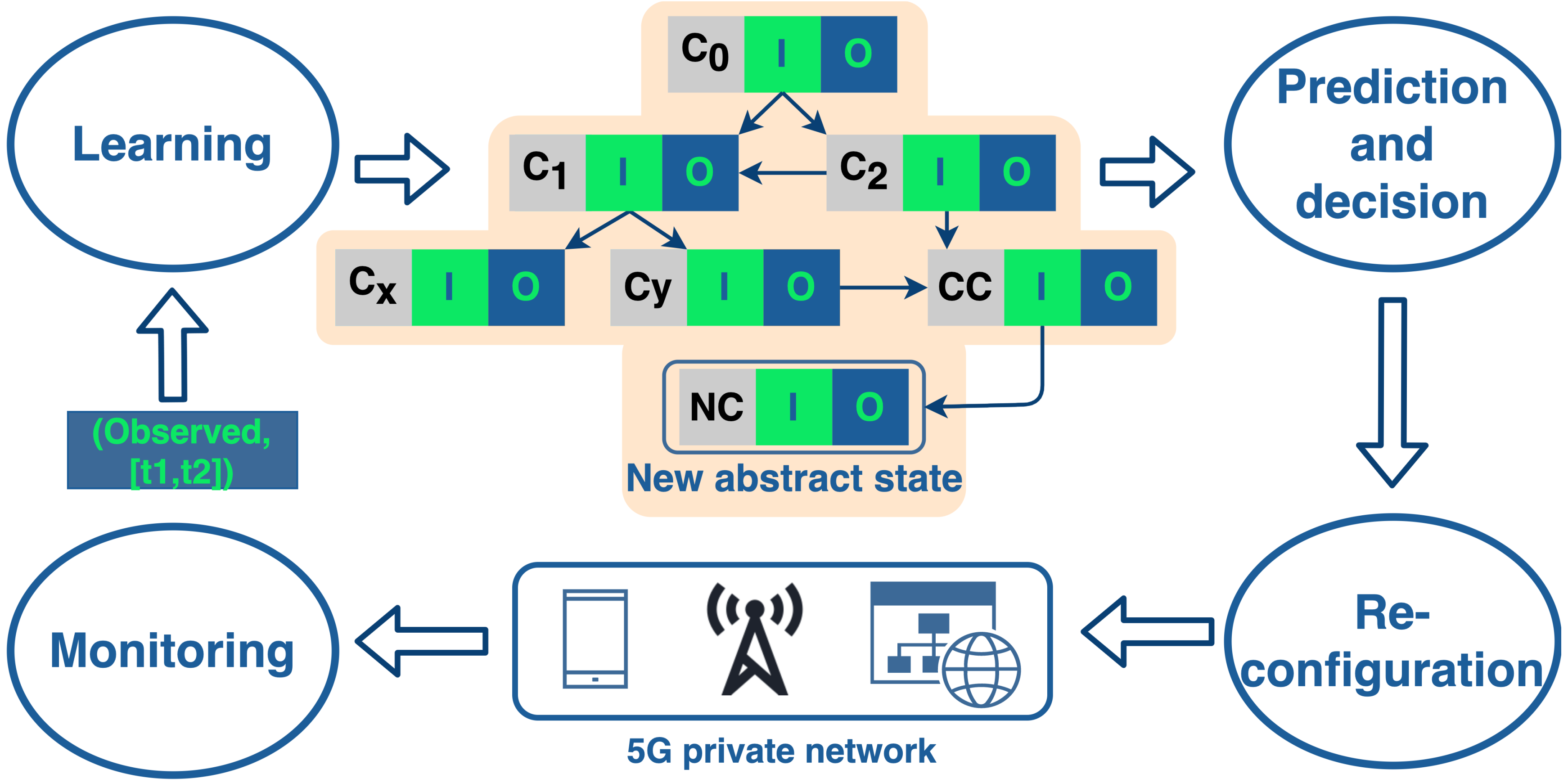
Conclusions & Future Work

- The implementation in Scala of the algorithm can process the network captures in real-time
- Learning automaton generates a model bounded mainly by the number of the network configurations



Conclusions & Future Work

- Extend the testbed:
  - Obtain traces with multiple TSN endpoints
  - Define new synchronization mechanisms between the 5G and the TSN network domains
- Extend the *Learn* algorithm:
  - Any number of configuration parameters and KPIs
  - Produce one/several automaton that represents complex scenarios
- Support real-time prediction and re-configuration features based on the learned automaton



Thank you!

Any questions?

Francisco Luque-Schempp
schempp@uma.es

SOFTWARE
UNIVERSIDAD DE MÁLAGA

MORSE