Adaptive Routing with Guaranteed Delay Bounds using Safe Reinforcement Learning

Gautham Nayak Seetanadi
Martina Maggio, Karl-Erik Årzen

Dept. of Automatic Control, Lund University

Problem Statement

- Route packets through a real-time network
- Give guarantees on total transmission delay
- Use reinforcement learning(**RL**) to explore the state-space

Problem Statement

DAG with worst case link delay

- Route packets through a ~~real-time~~ network
- Give guarantees on total transmission delay
- Use reinforcement learning(**RL**) to explore the state-space

Problem Statement

DAG with worst case link delay

Never violate preset deadline D_F

- Route packets through a real-time network
- Give guarantees on total transmission delay
- Use reinforcement learning(**RL**) to explore the state-space

Problem Statement

DAG with worst case link delay

Never violate preset deadline D_F

- Route packets through a real-time network
- Give guarantees on total transmission delay
- Use reinforcement learning(**RL**) to explore the state-space

Eliminate need for complex routing tables

Problem Statement

DAG with worst case link delay

dline D_F

Real-time routing requires guarantees

- RL is simple but powerful.
- RL is inherently stochastic.

- Ro
- Gi
- Us … space

Eliminate need for complex routing tables

- Route packets through a real-time network
- Give guarantees on total transmission delay
- Use reinforcement learning (**RL**) to explore the state-space

Try the algorithm yourself

https://github.com/AdaptiveRouting-using-RL/AdaptiveRoutingUsingRL

Motivation

- Typical transmission time over link varies over time

Motivation

- Typical transmission time over link varies over time
- Classical RL based solutions

Motivation

- Typical transmission time over link varies over time
- Classical RL based solutions
  - Reduce total transmission times
  - Do not provide guarantees

Motivation

- Typical transmission time over link varies over time
- Classical RL based solutions
  - Reduce total transmission times
  - Do not provide guarantees
- Real-time solution [1]

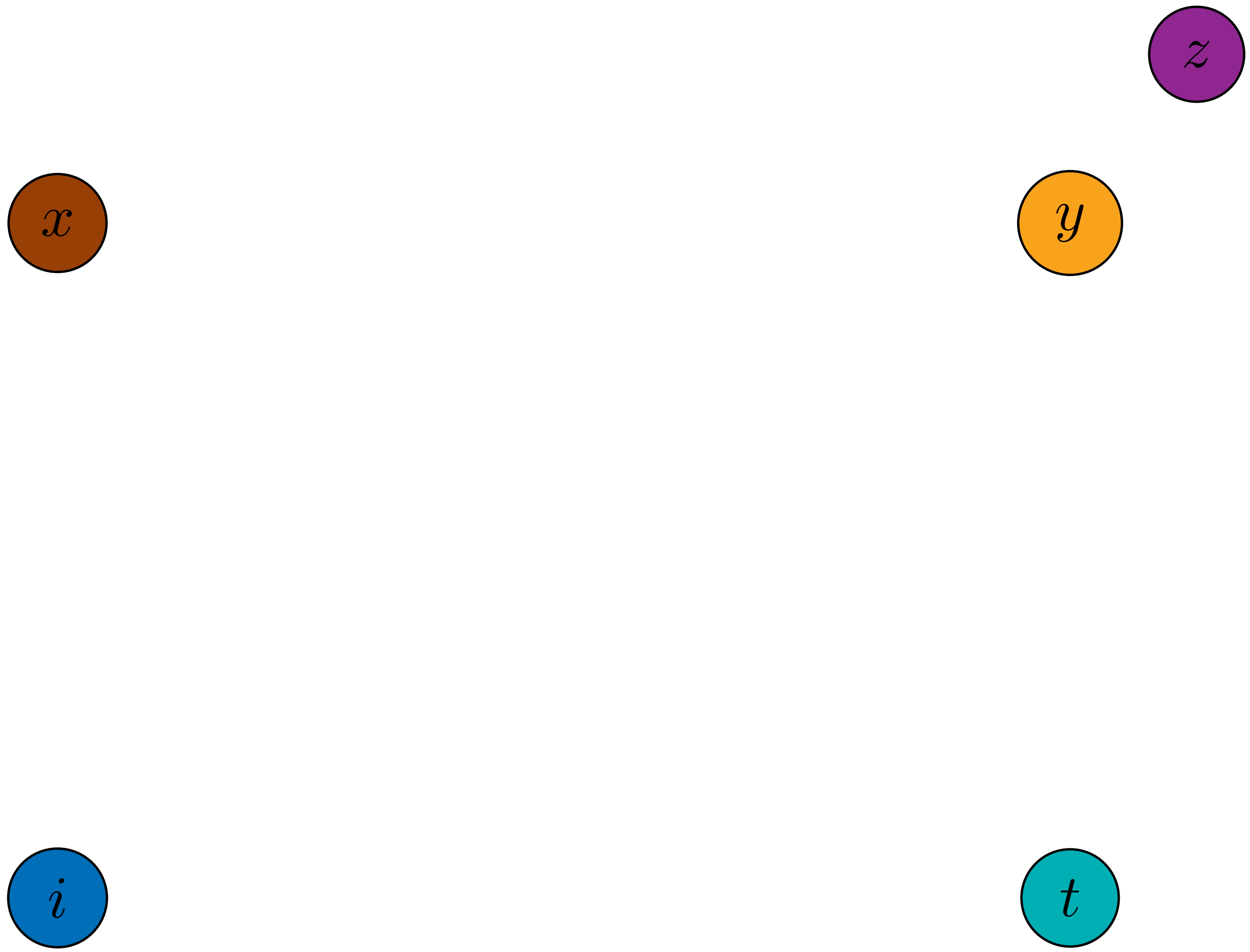
[1] S. Baruah, “Rapid routing with guaranteed delay bounds,” in 2018 IEEE Real-Time Systems Symposium (RTSS) , December 2018

Motivation

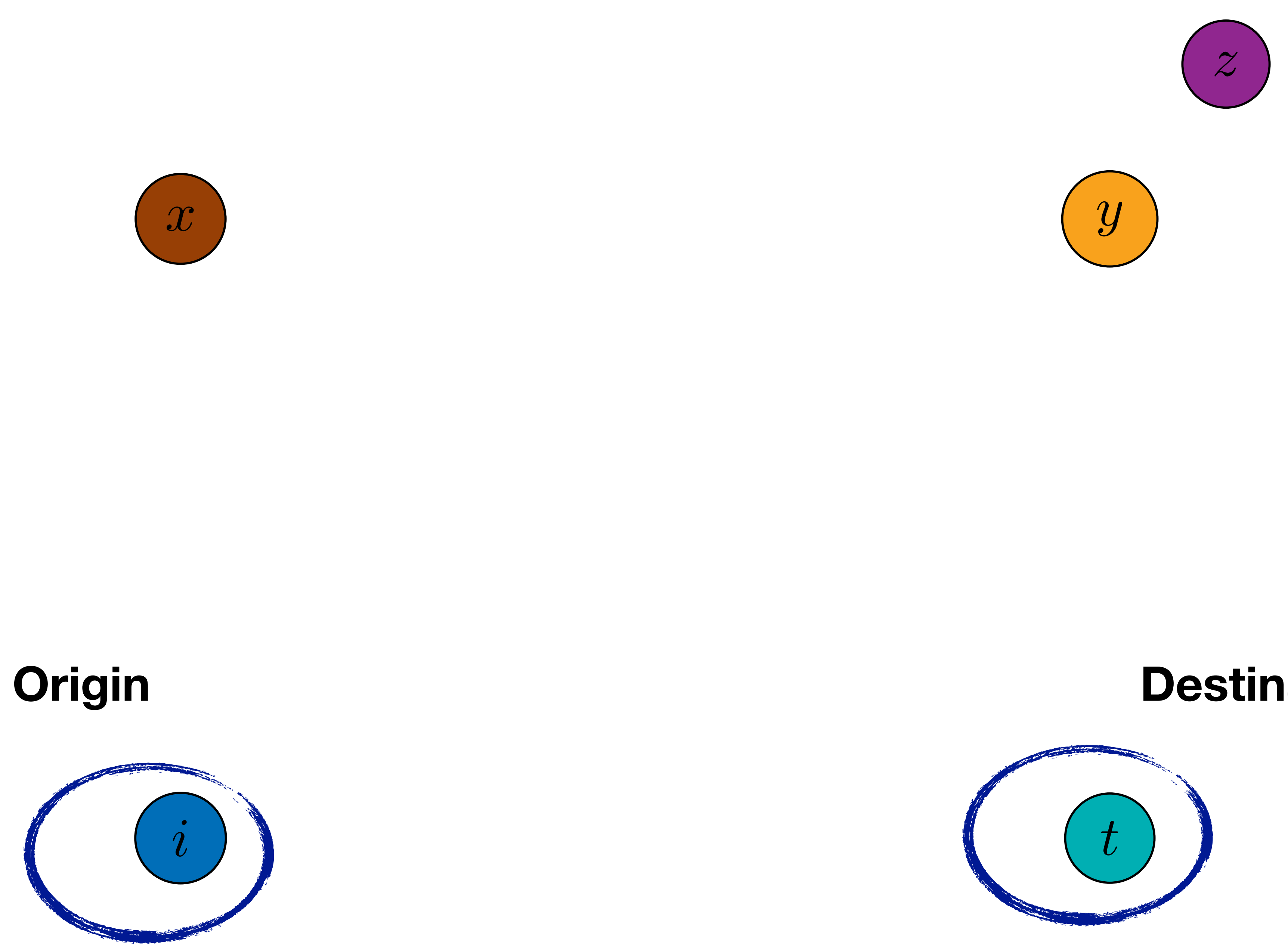
- Typical transmission time over link varies over time
- Classical RL based solutions
  - Reduce total transmission times
  - Do not provide guarantees
- Real-time solution [1]
  - Provide Guarantees
  - Depends on complex routing tables
  - Does not react to disturbances

[1] S. Baruah, “Rapid routing with guaranteed delay bounds,” in 2018 IEEE Real-Time Systems Symposium (RTSS) , December 2018

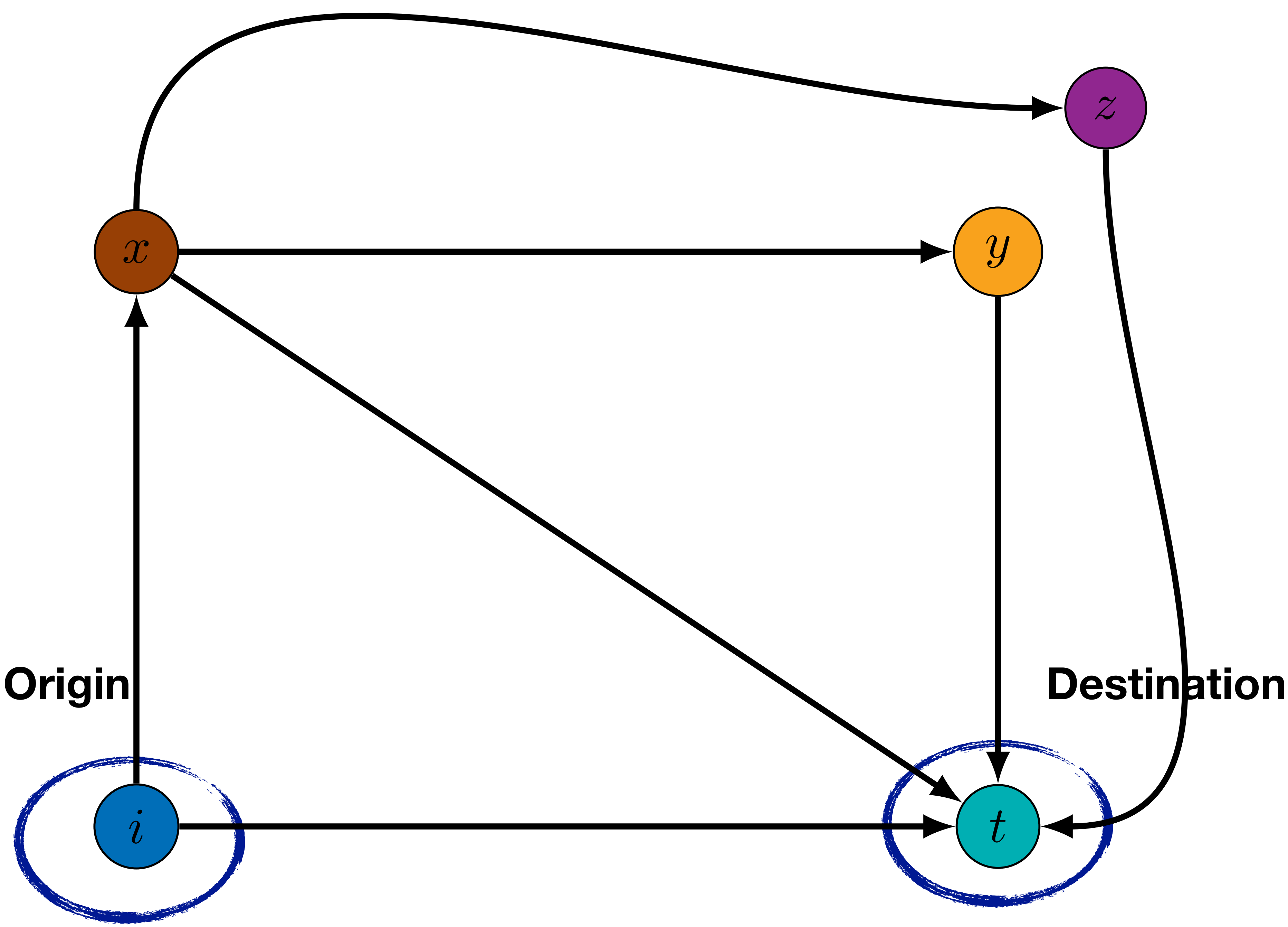
Example Network Overview



Example Network Overview

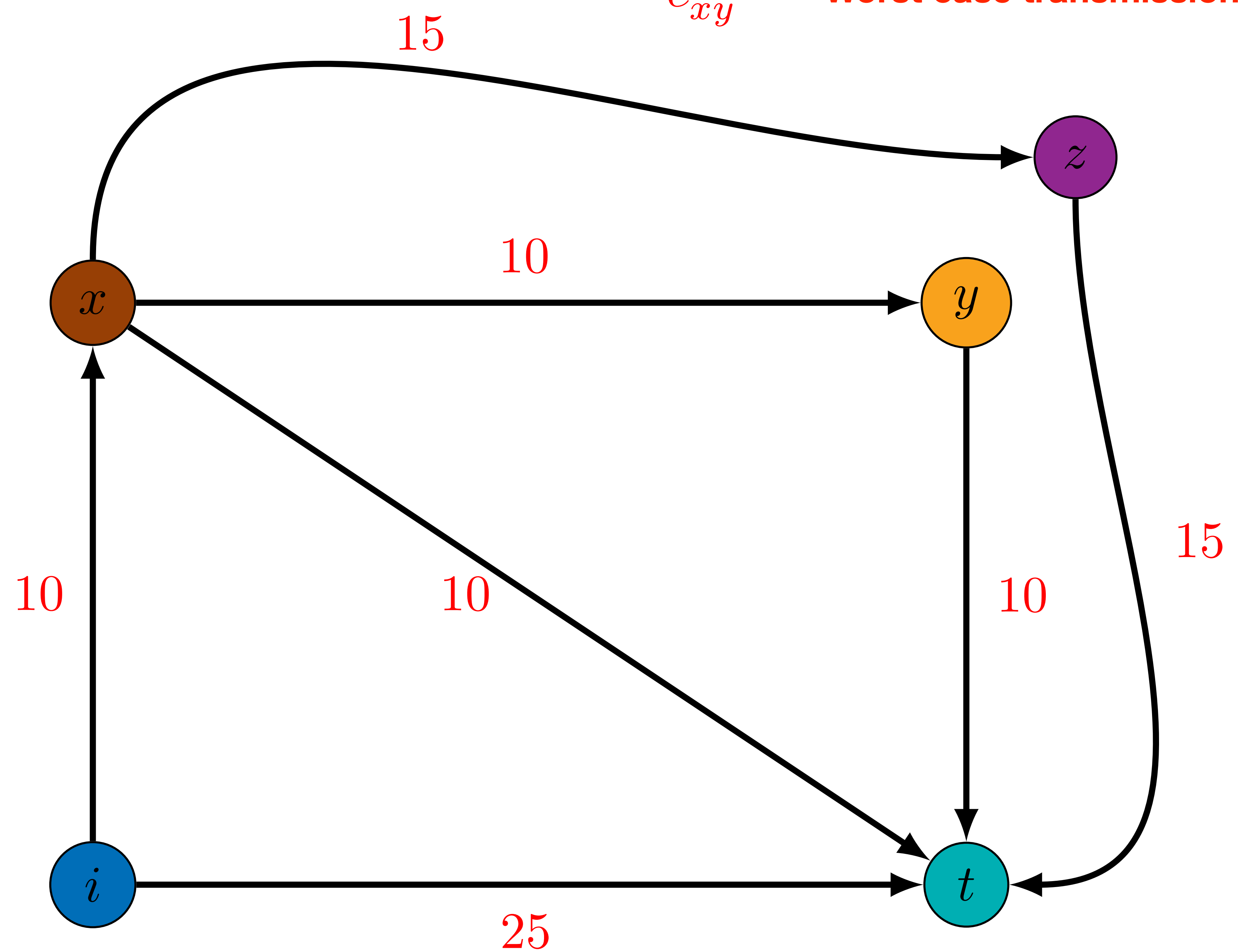


Example Network Overview

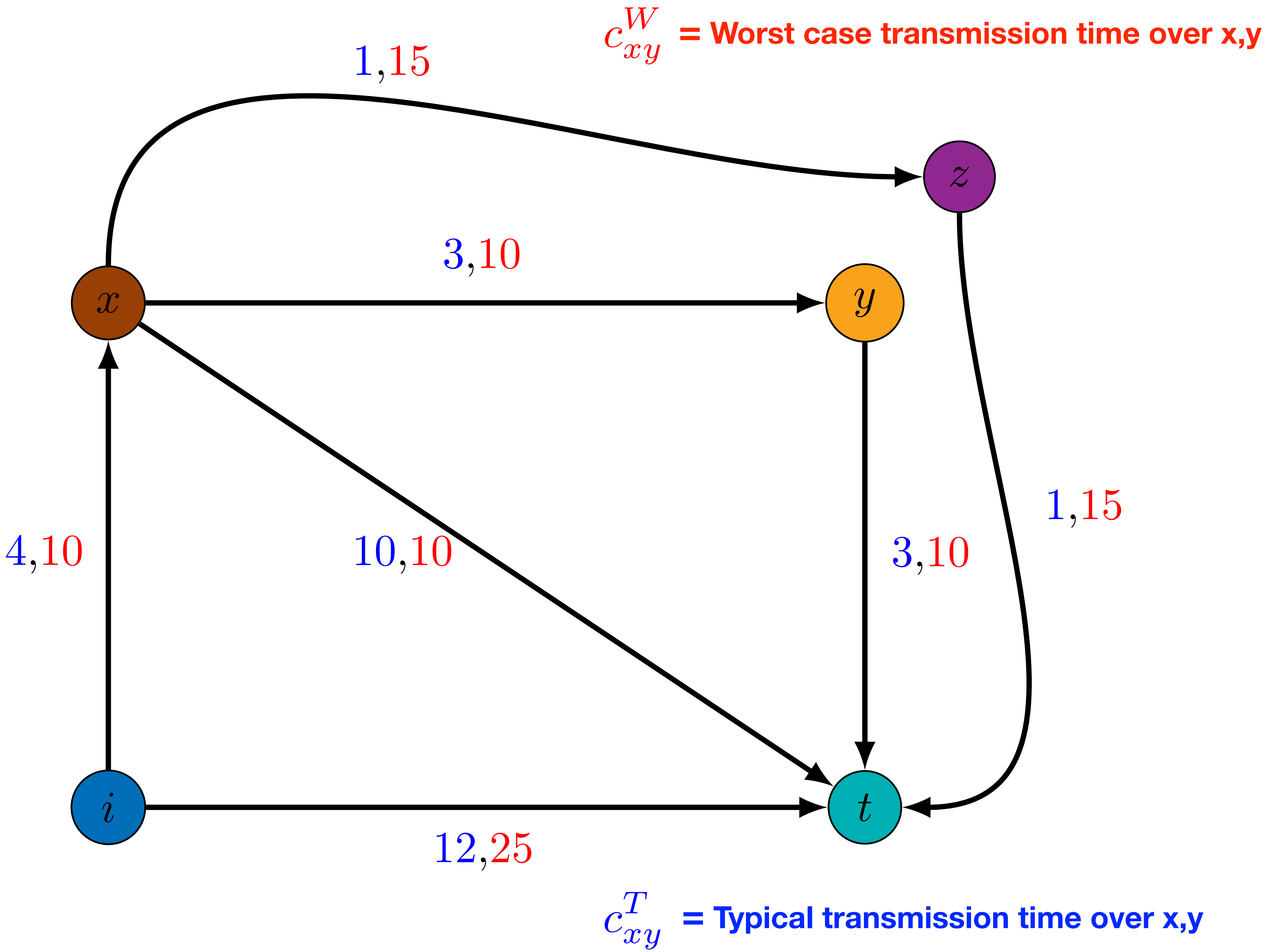


Example Network Overview

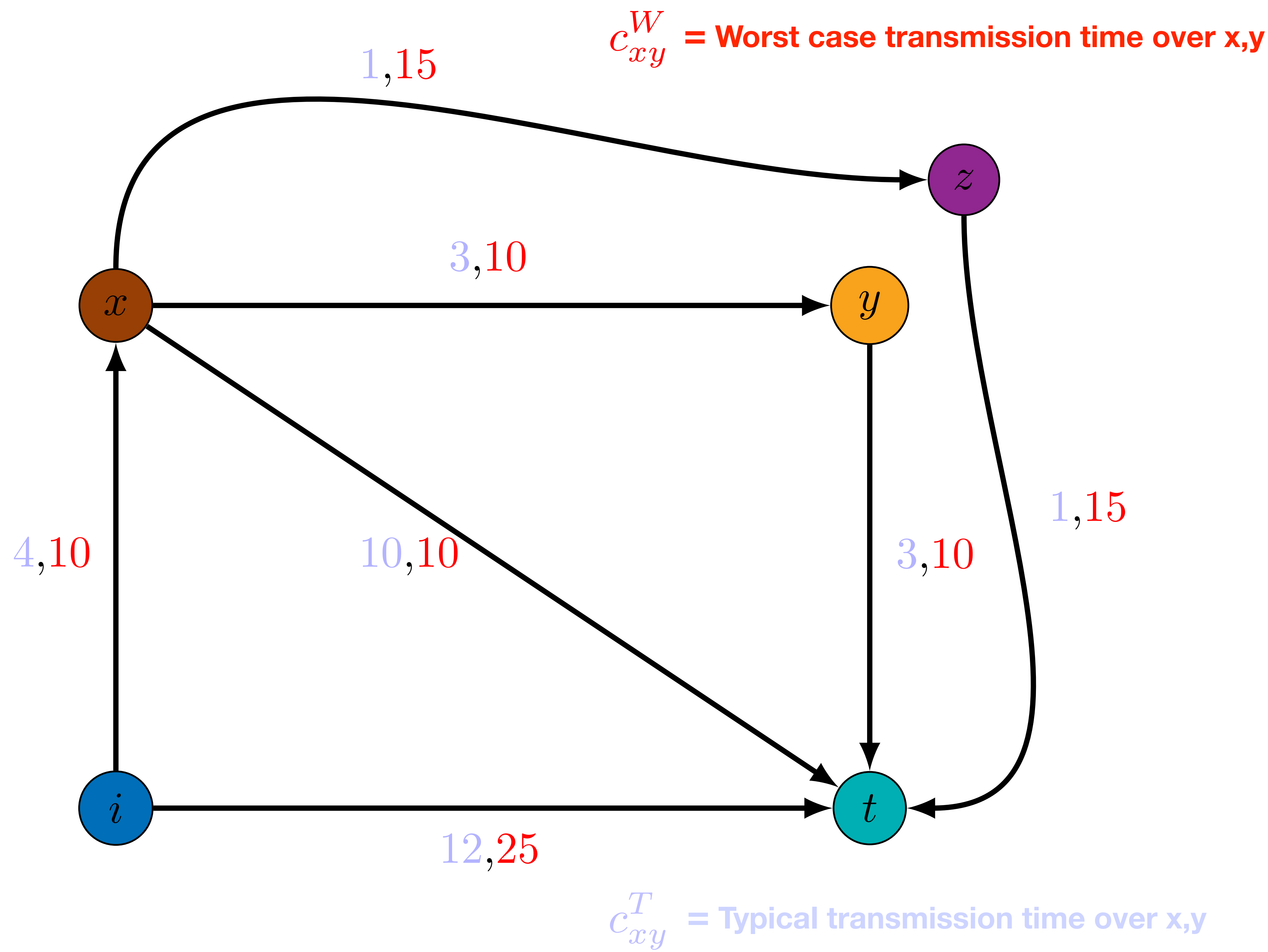
c^{W}_{xy} = Worst case transmission time over x,y



Example Network Overview

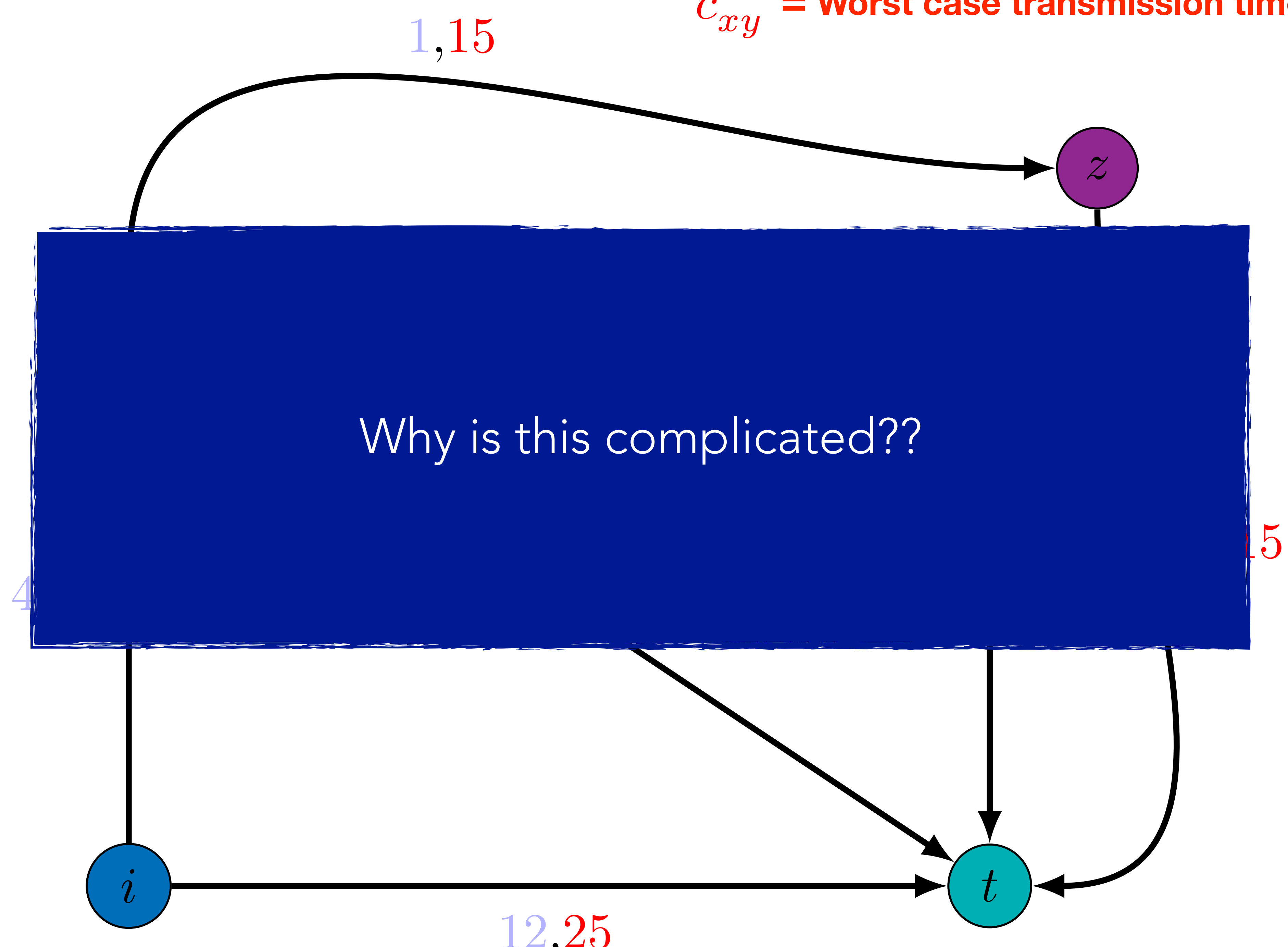


Example Network Overview



Example Network Overview

c^W_{xy} = Worst case transmission time over x,y



Why is this complicated??

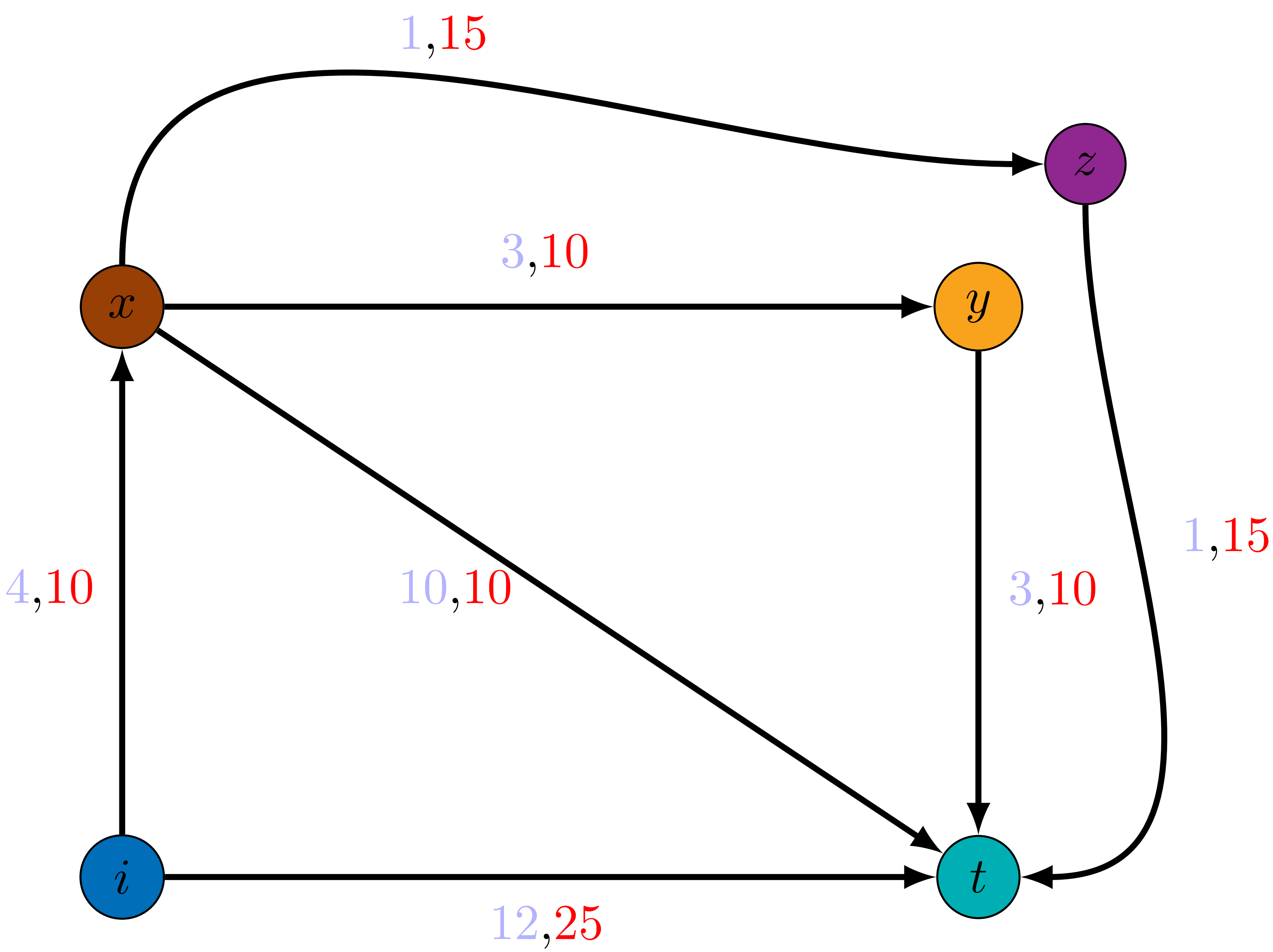
c^T_{xy} = Typical transmission time over x,y

Example Network Overview

Deadline

Typical delay to t

PATH	δ	D_F

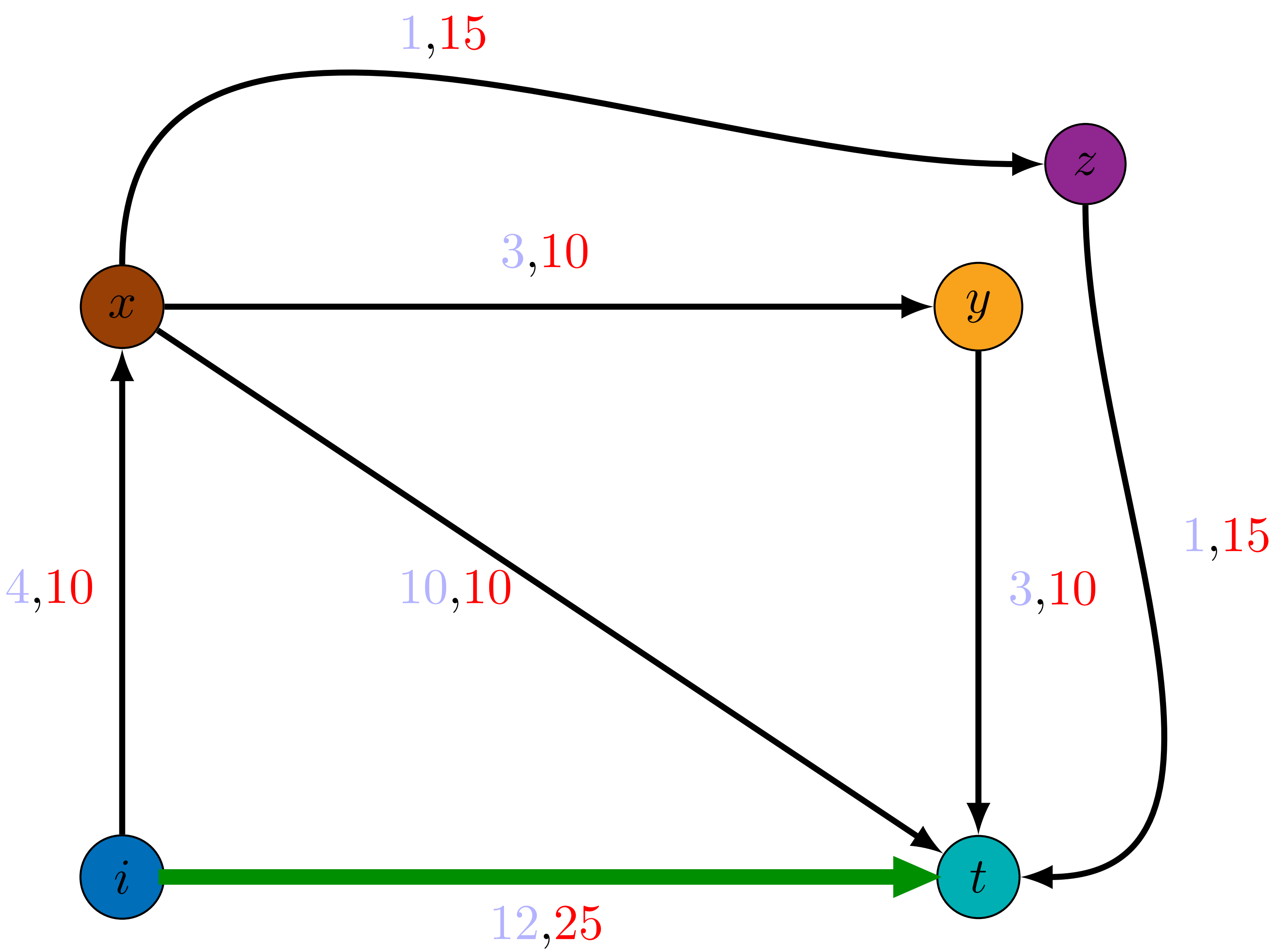


Example Network Overview

Deadline

Typical delay to t

PATH	δ	D_F
$i \rightarrow t$	12	25

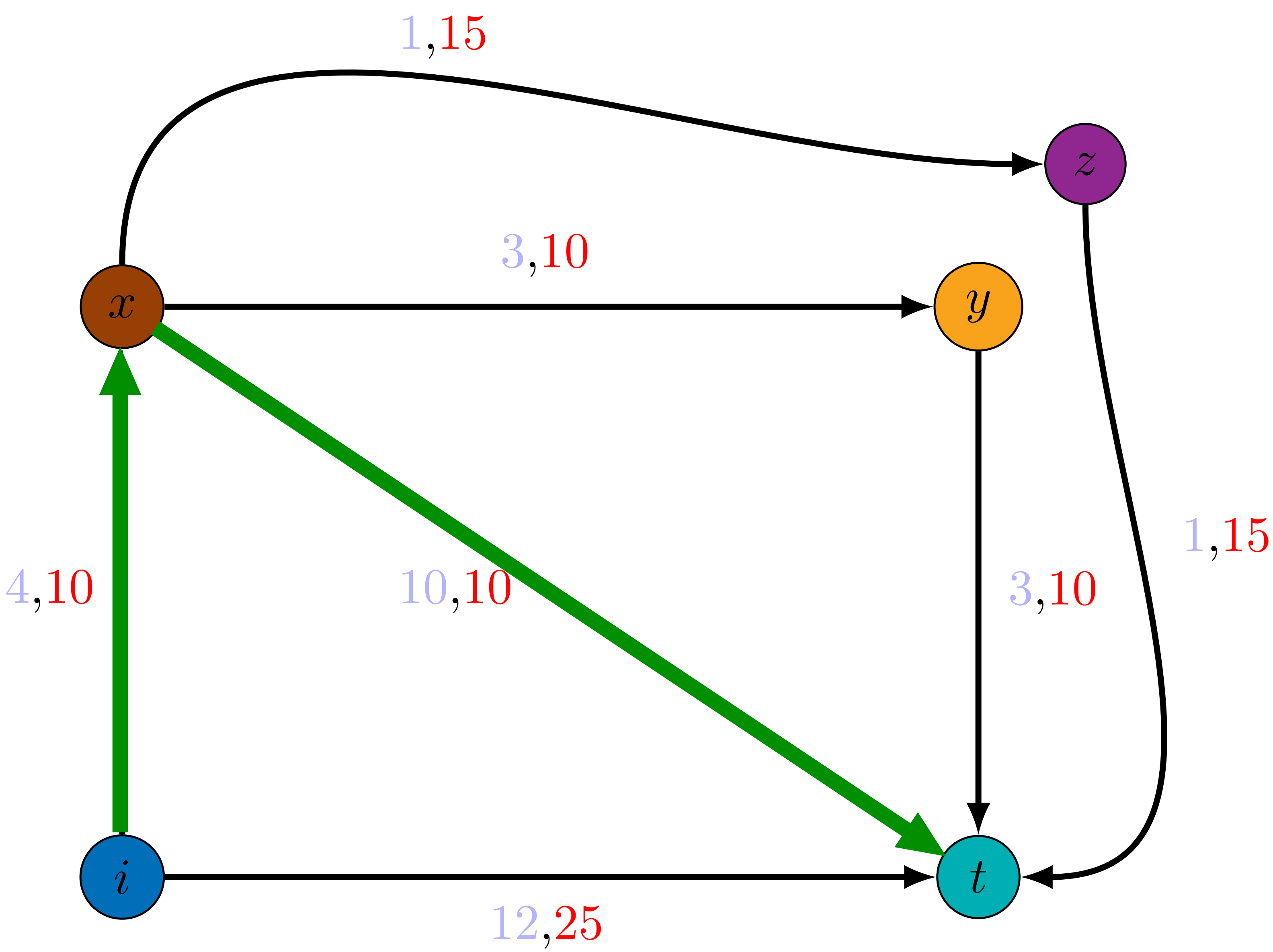


Example Network Overview

Deadline

Typical delay to t

PATH	δ	D_F
$i \to t$	12	25
$i \to x \to t$	14	20

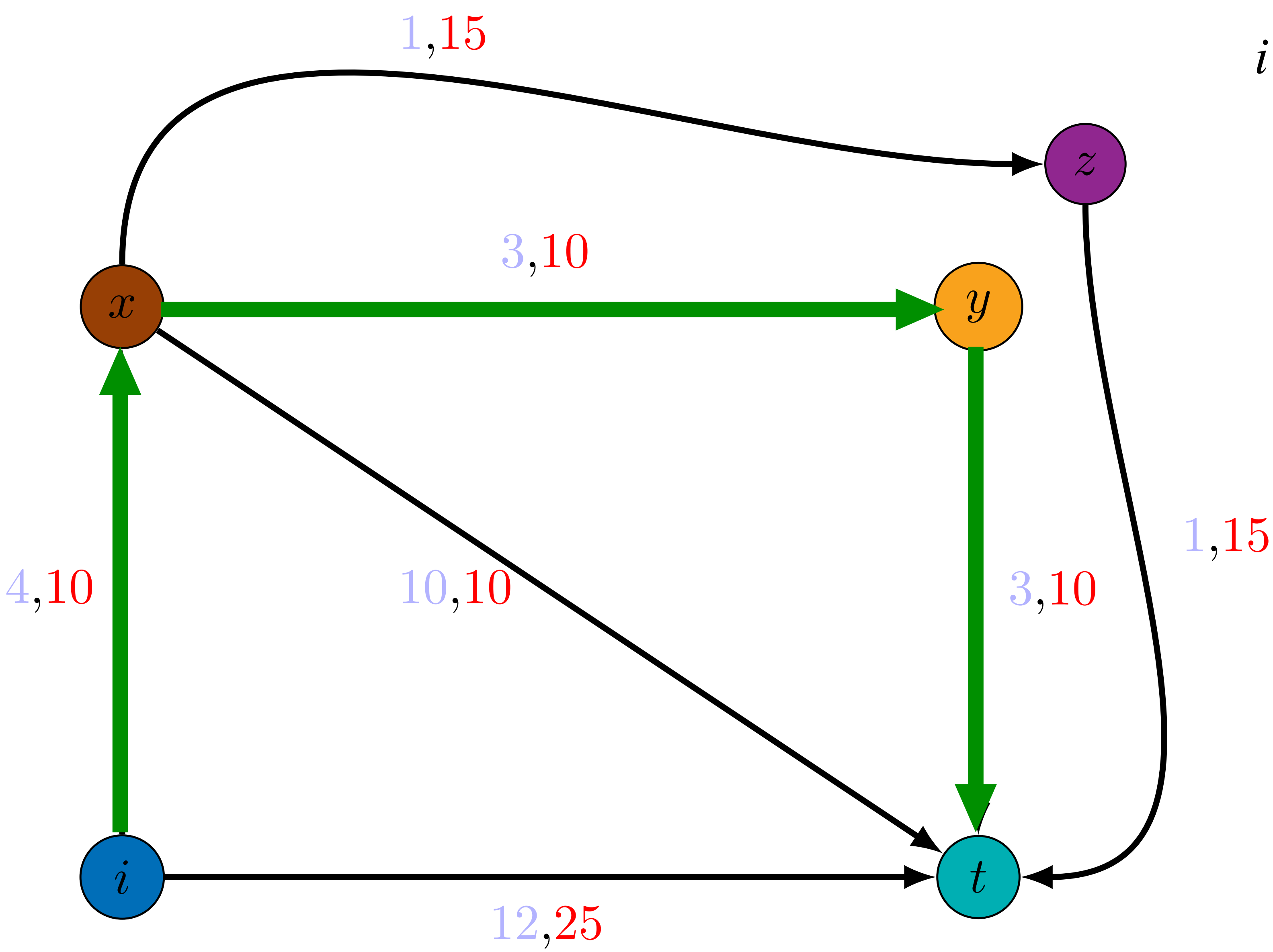


Example Network Overview

Deadline

Typical delay to t

PATH	δ	D_F
$i \to t$	12	25
$i \to x \to t$	14	20
$i \to x \to y \to t$	10	30

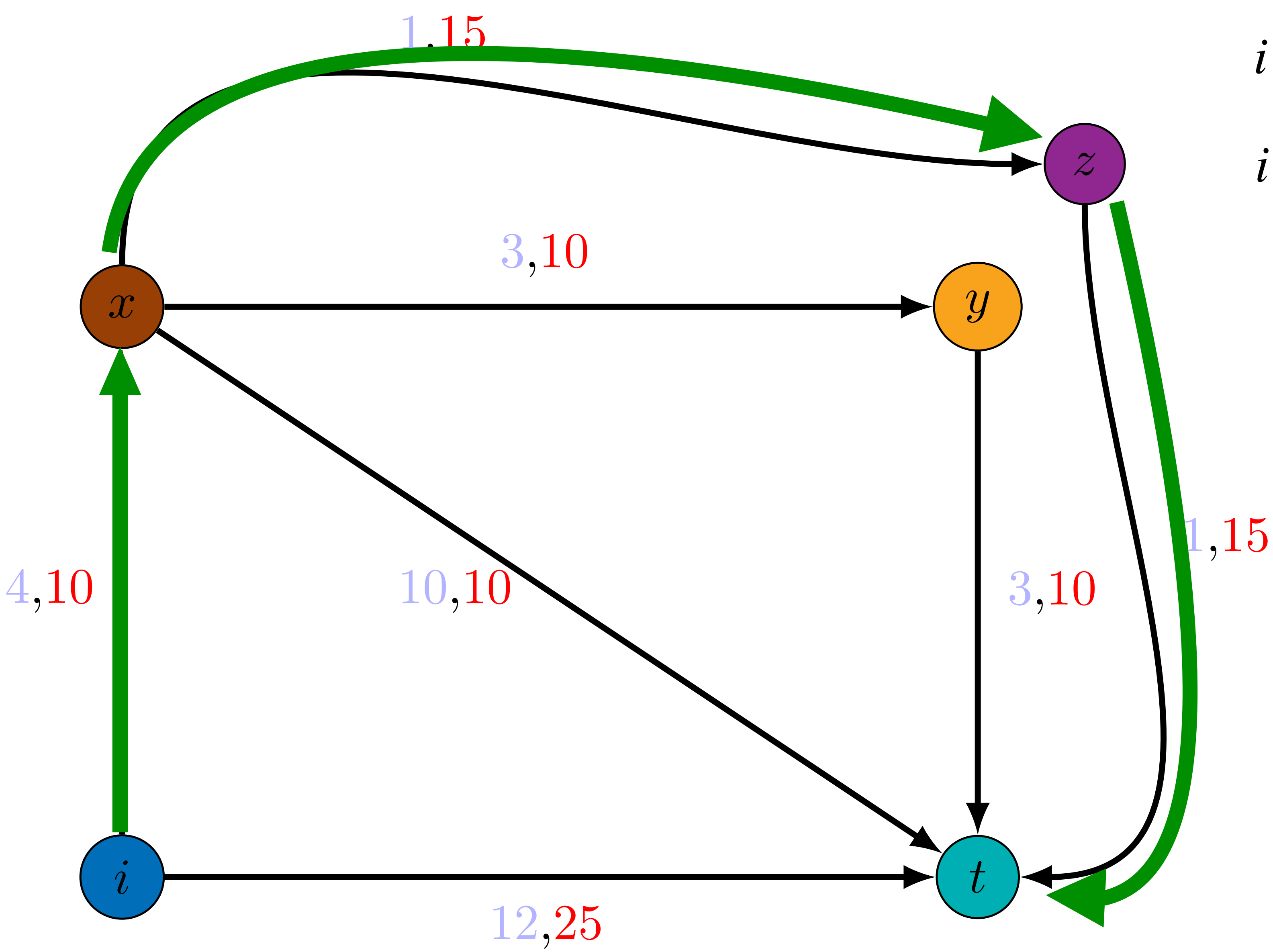


Example Network Overview

Deadline

Typical delay to t

PATH	δ	D_F
$i \to t$	12	25
$i \to x \to t$	14	20
$i \to x \to y \to t$	10	30
$i \to x \to z \to t$	6	40

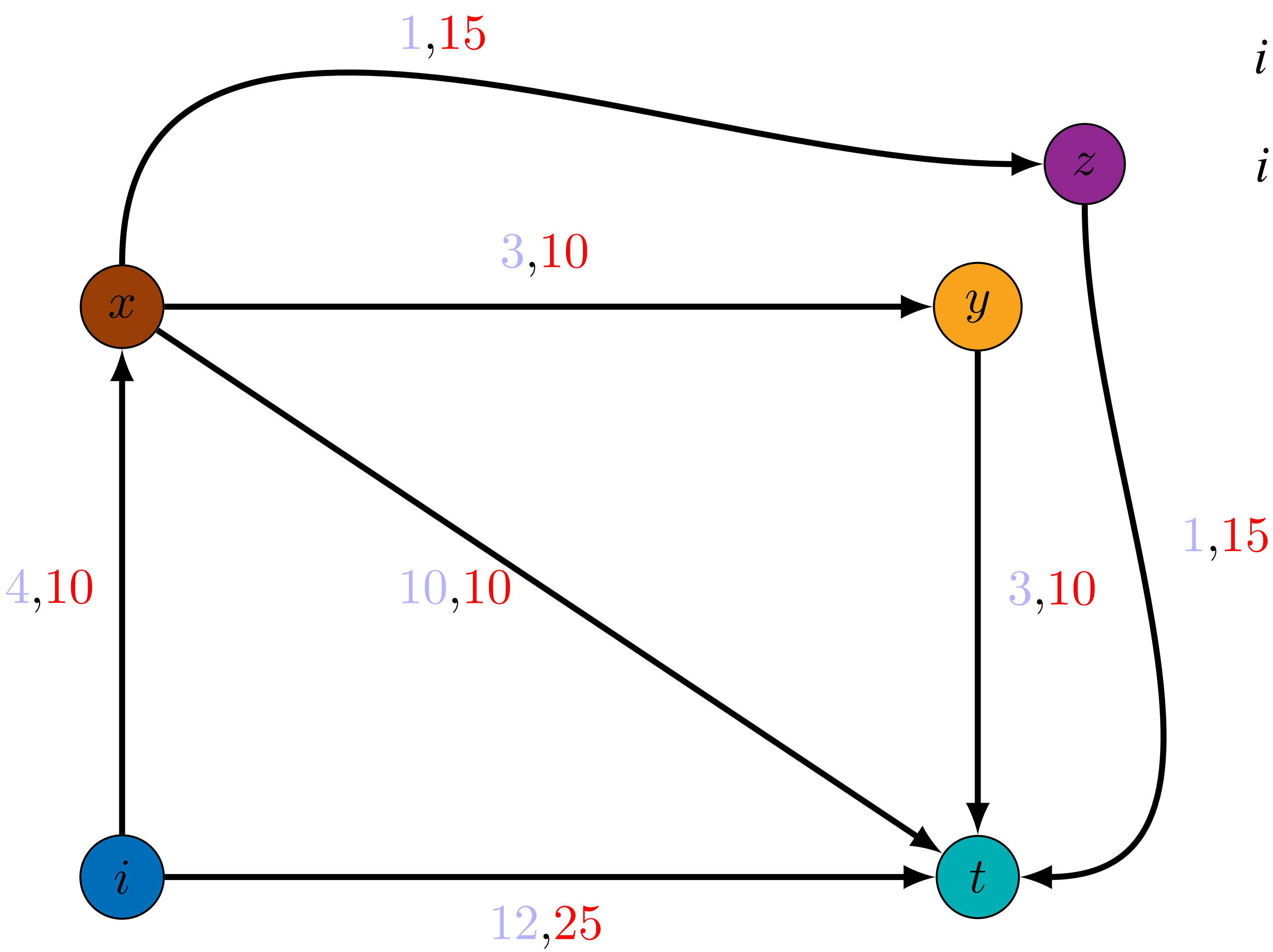


Example Network Overview

Deadline

Typical delay to t

PATH	δ	D_F
$i \to t$	12	25
$i \to x \to t$	14	20
$i \to x \to y \to t$	10	30
$i \to x \to z \to t$	6	40



The Beginning

Subject: Your talk today
Date: Wed, 12 Dec 2018 17:14:25 +0100
From: Karlerik karl-erik.arzen@control.lth.se
To: baruah@wustl.edu

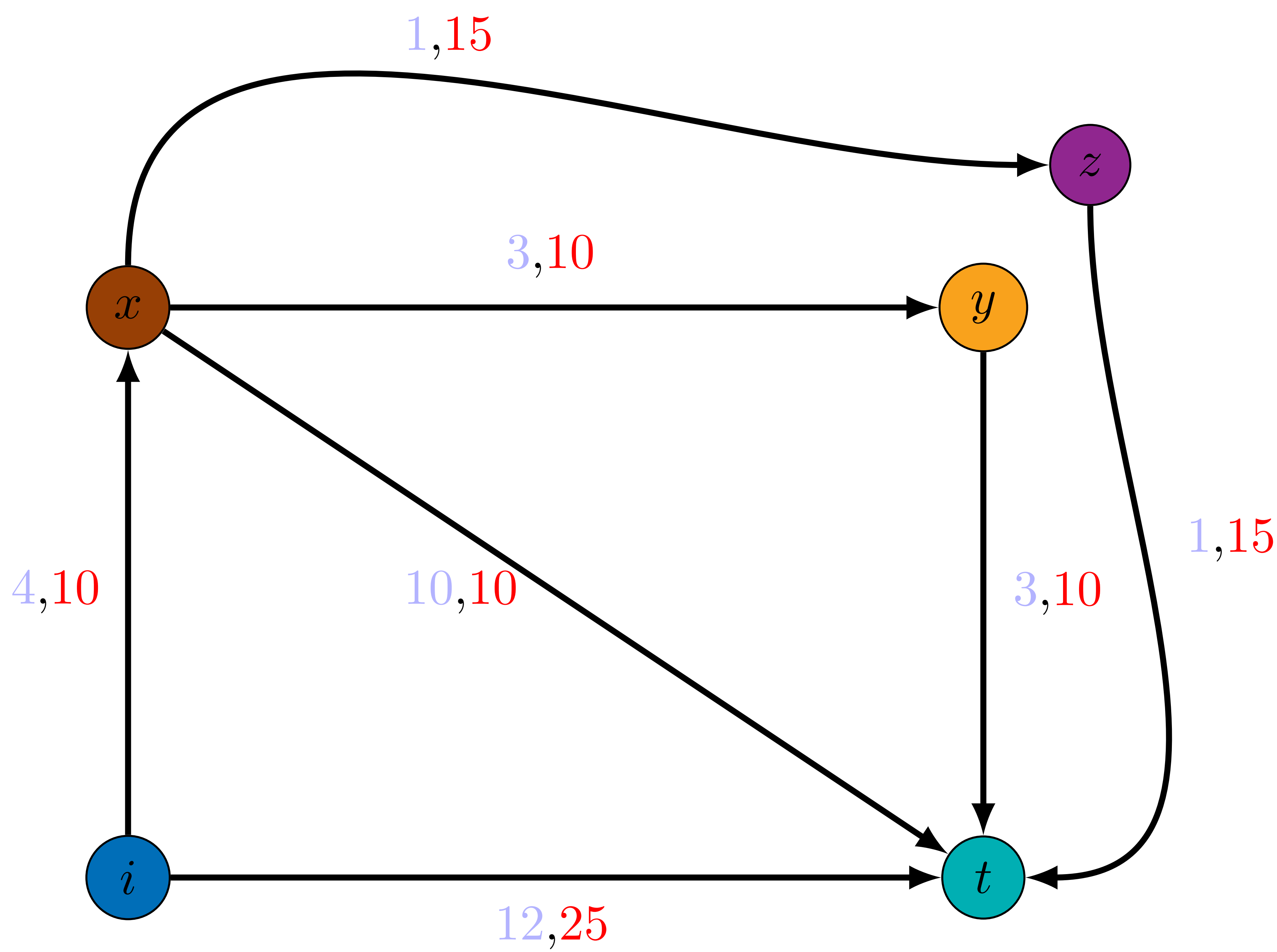
Sanjoy

Very interesting talk. I am currently trying to learn Reinforcement learning (RL). When I heard your talk I felt many similarities between what you are doing and policy learning and q-learning in RL. There might be an interesting connection.

Best

Karl-Erik

Previous work [1]



[1] S. Baruah, “Rapid routing with guaranteed delay bounds,” in 2018 IEEE Real-Time Systems Symposium (RTSS) , December 2018

Previous work [1]

Remaining delay bound

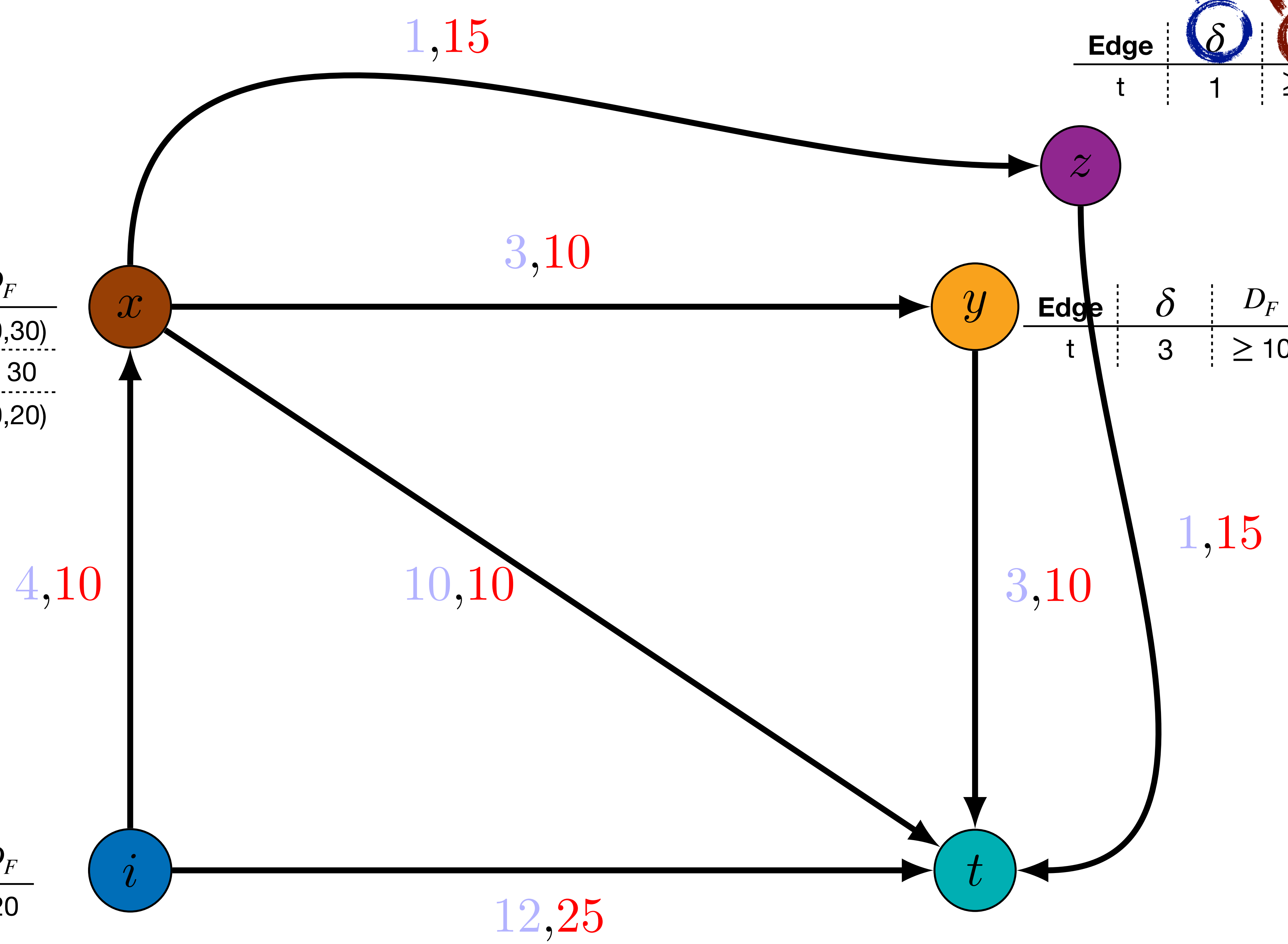
Typical delay to t

Edge	δ	D_F
t	1	$\geq$ 15

Edge	δ	D_F
y	6	[20,30)
z	2	$\geq$ 30
t	10	[10,20)

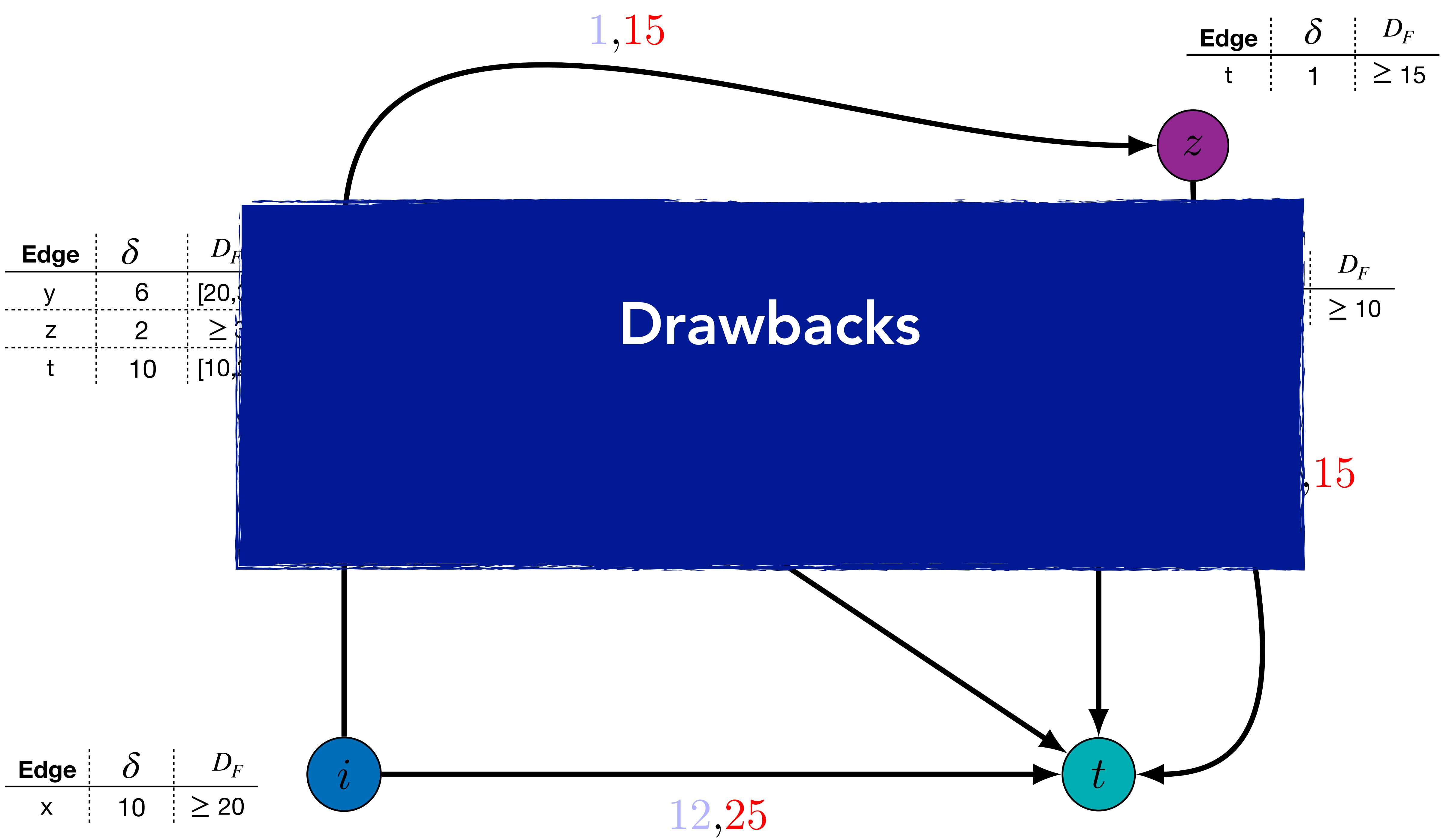
Edge	δ	D_F
t	3	$\geq$ 10

Edge	δ	D_F
x	10	$\geq$ 20



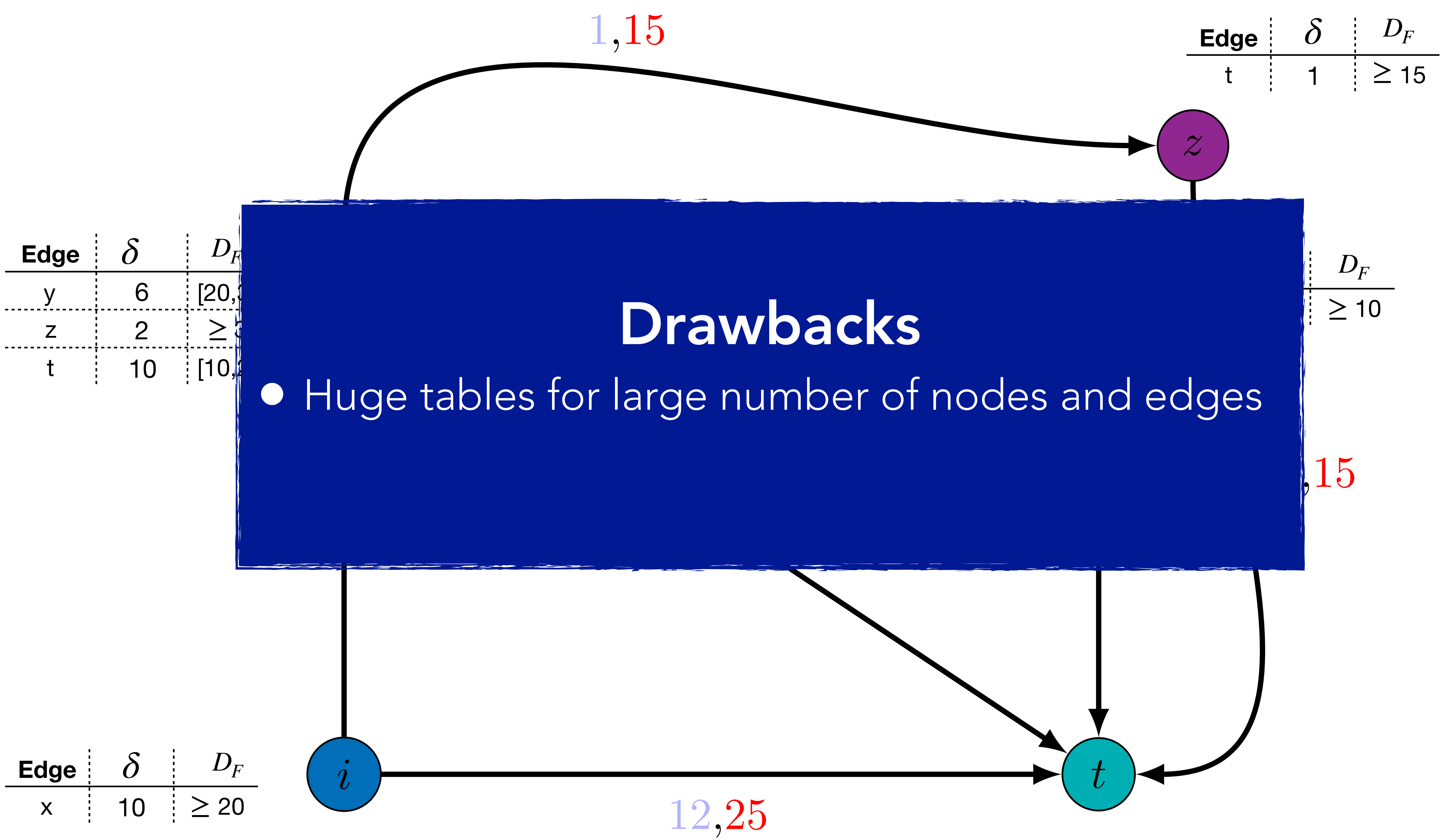
[1] S. Baruah, “Rapid routing with guaranteed delay bounds,” in 2018 IEEE Real-Time Systems Symposium (RTSS) , December 2018

Previous work [1]



[1] S. Baruah, “Rapid routing with guaranteed delay bounds,” in 2018 IEEE Real-Time Systems Symposium (RTSS) , December 2018

Previous work [1]

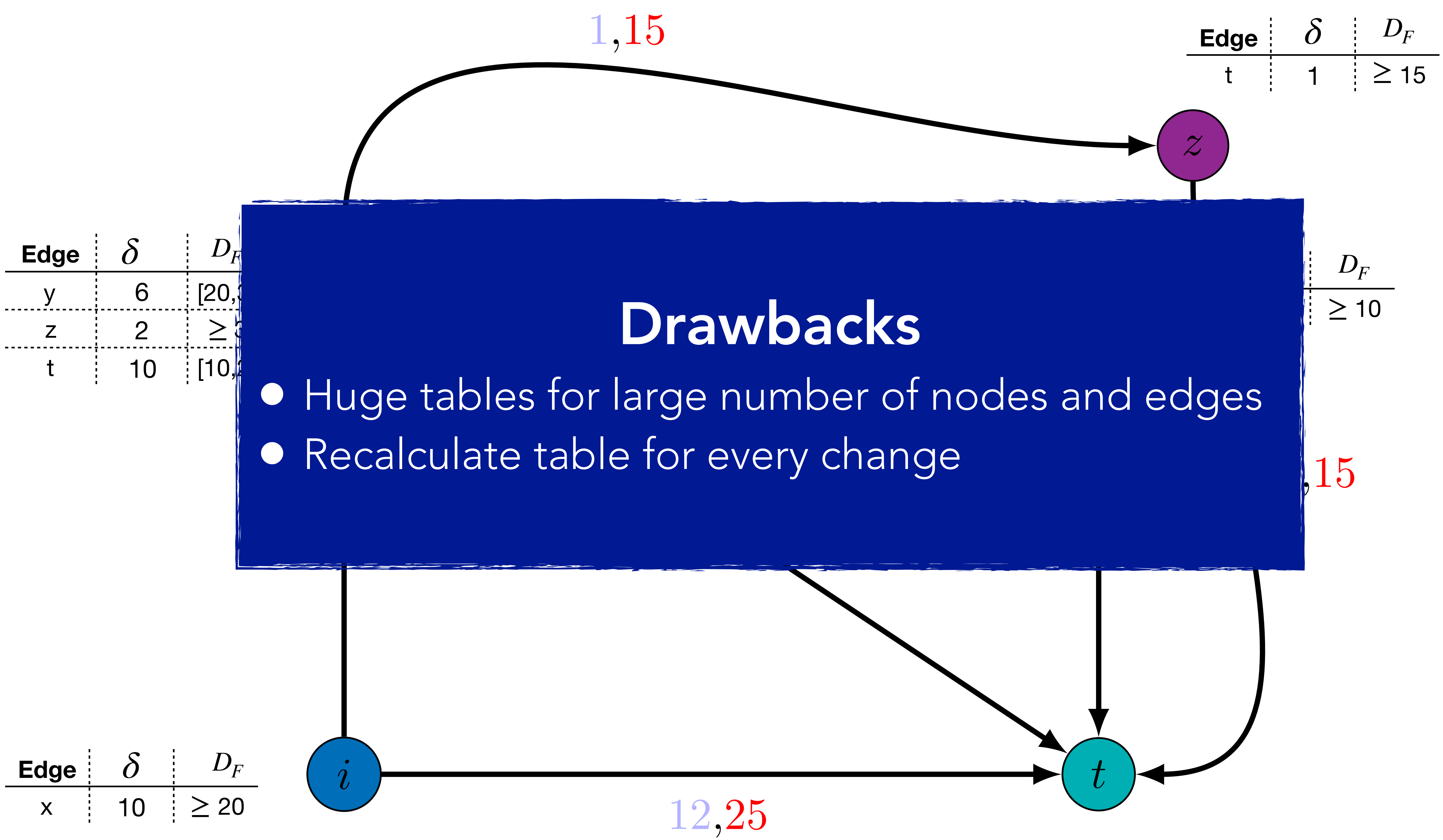


Drawbacks

- Huge tables for large number of nodes and edges

[1] S. Baruah, “Rapid routing with guaranteed delay bounds,” in 2018 IEEE Real-Time Systems Symposium (RTSS) , December 2018

Previous work [1]



Drawbacks

- Huge tables for large number of nodes and edges
- Recalculate table for every change

[1] S. Baruah, “Rapid routing with guaranteed delay bounds,” in 2018 IEEE Real-Time Systems Symposium (RTSS) , December 2018

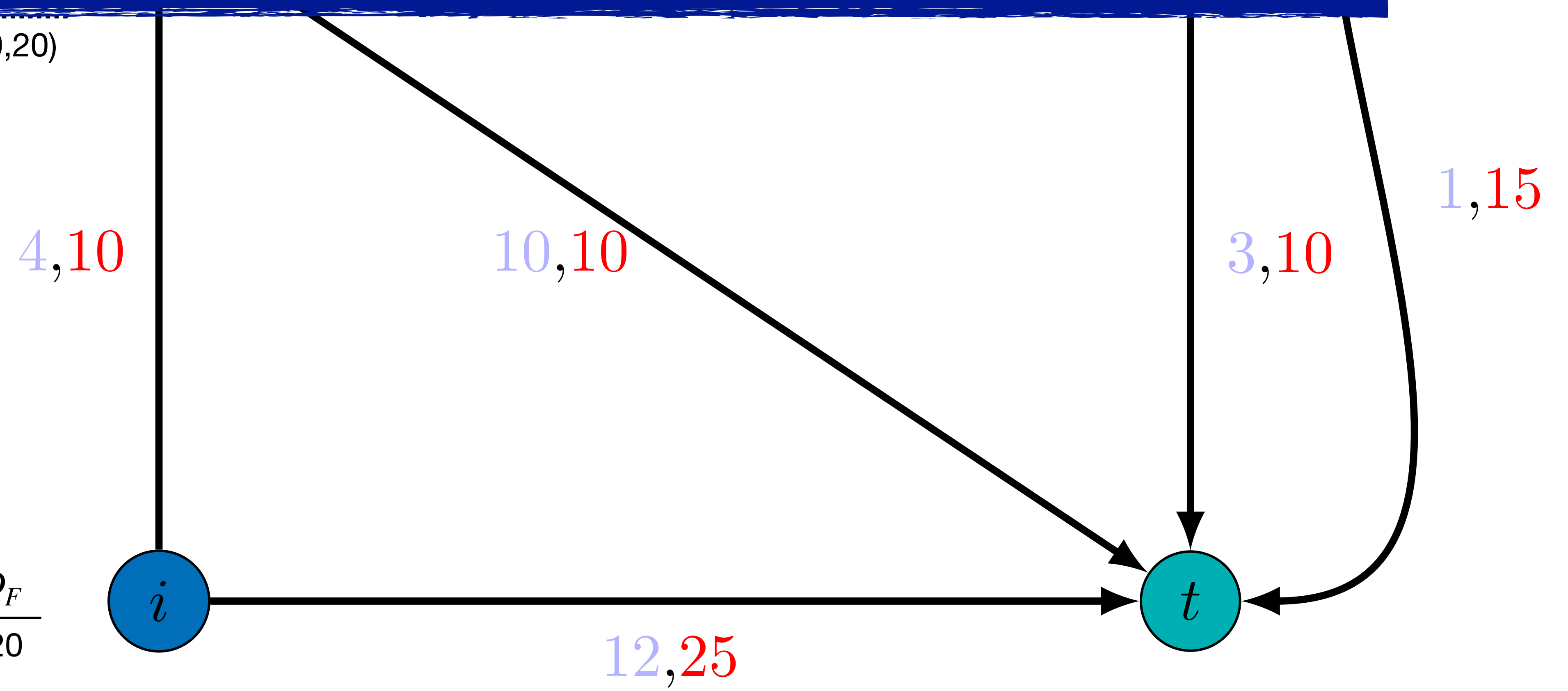
Previous work [1]

Our solution?

e	δ	D_F
	1	≥ 15

Edge	δ	
y	6	
z	2	
t	10	[10,20)

δ	D_F
3	≥ 10



Edge	δ	D_F
x	10	≥ 20

[1] S. Baruah, “Rapid routing with guaranteed delay bounds,” in 2018 IEEE Real-Time Systems Symposium (RTSS) , December 2018

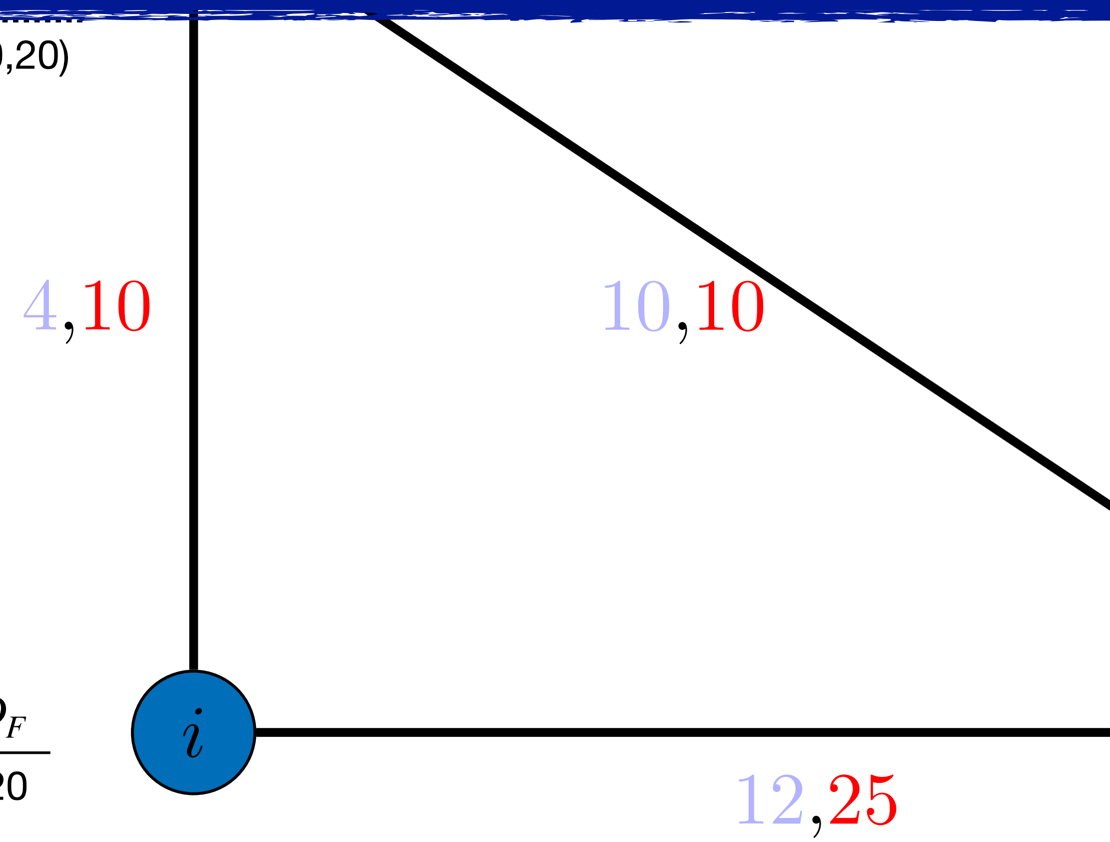
Previous work [1]

Our solution?
Use Reinforcement Learning

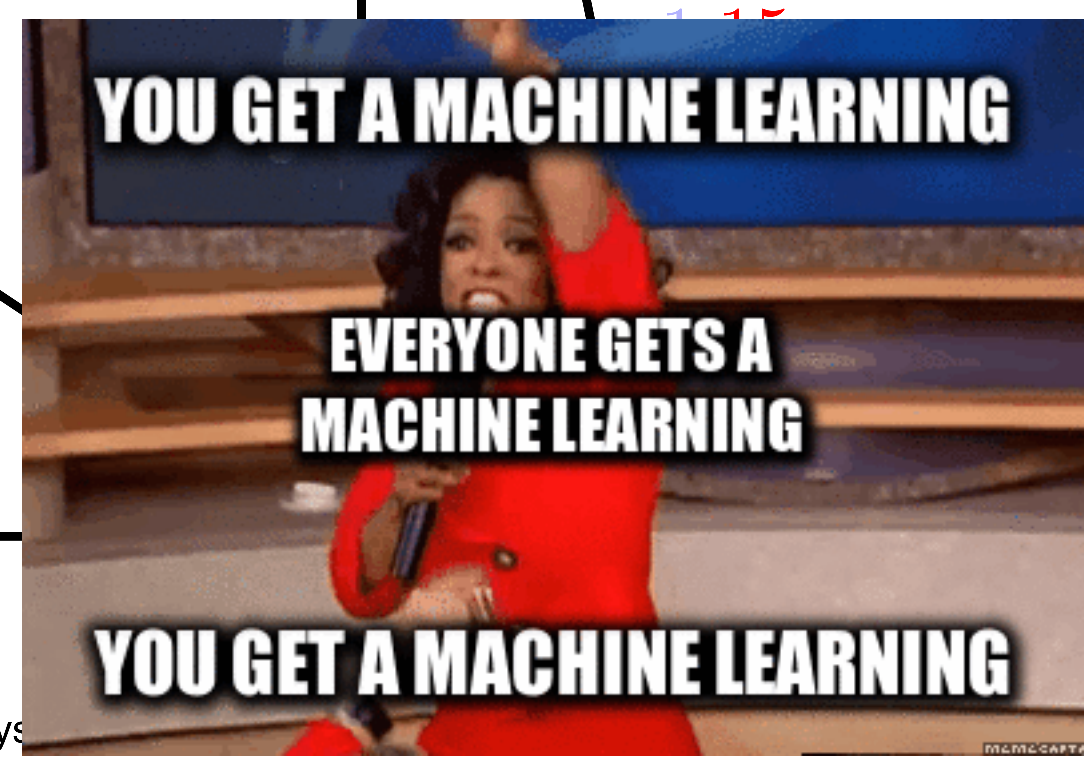
...e	δ	D_F
	1	$\geq$ 15

Edge	δ	
y	6	
z	2	
t	10	[10,20)

	δ	D_F
	3	$\geq$ 10



Edge	δ	D_F
x	10	$\geq$ 20



[1] S. Baruah, "Rapid routing with guaranteed delay bounds," in 2018 IEEE Real-Time Sys

Reinforcement Learning

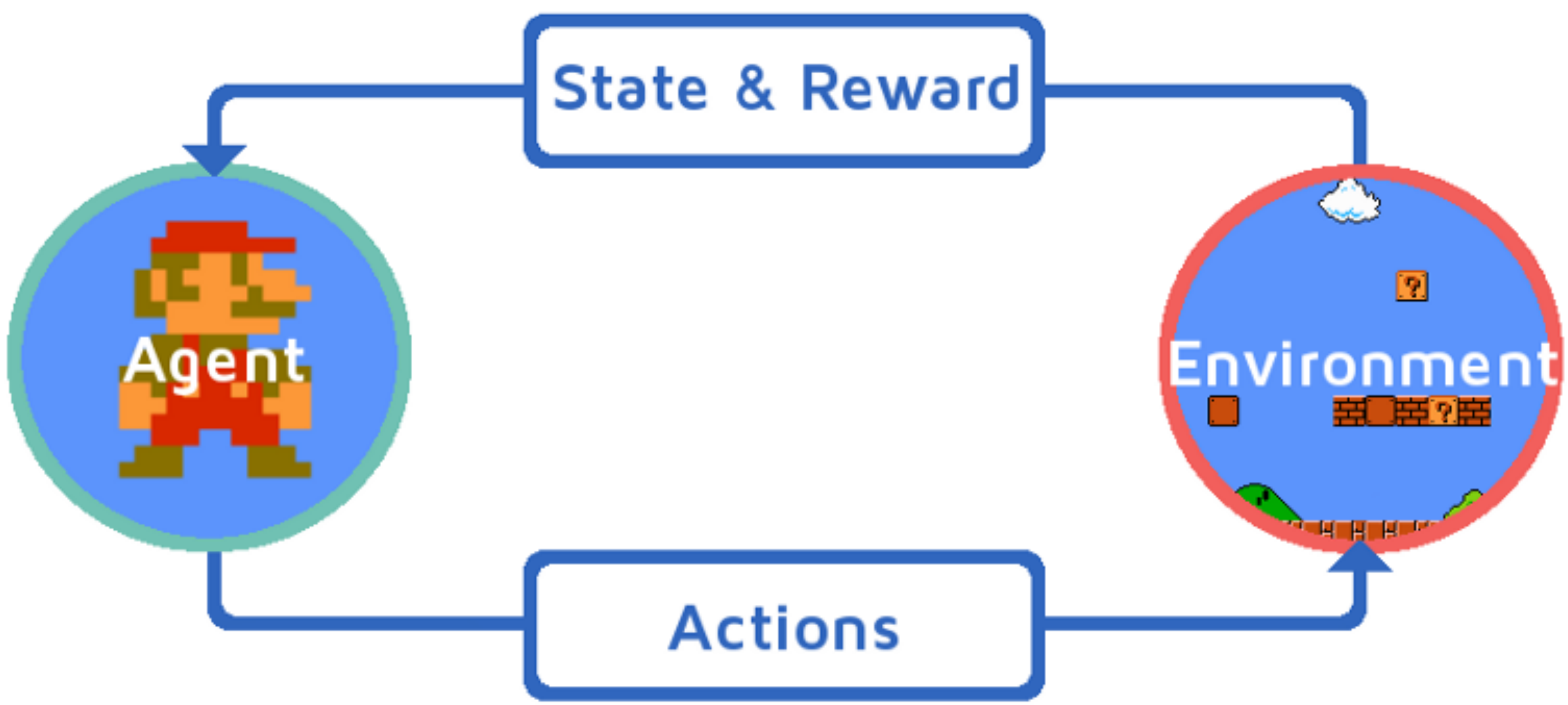
- Branch of machine learning that is unsupervised

Reinforcement Learning

- Branch of machine learning that is unsupervised
- Agent teaches itself how to behave by trial and error in episodic manner

Reinforcement Learning

- Branch of machine learning that is unsupervised
- Agent teaches itself how to behave by trial and error in episodic manner
- Learns to maximise a reward returned by the environment



	Lunch	Mario	Routing

	Lunch	Mario	Routing
Agent	You	Mario/Luigi	Node

	Lunch	Mario	Routing
Agent	You	Mario/Luigi	Node
Action Set (A)	• Inspira • Finn Inn • Govindas	Movement	Possible edges

	Lunch	Mario	Routing
Agent	You	Mario/Luigi	Node
Action Set (A)	• Inspira • Finn Inn • Govindas	Movement	Possible edges
Environment	Lund	Super Mario world	Network

	Lunch	Mario	Routing
Agent	You	Mario/Luigi	Node
Action Set (A)	• Inspira • Finn Inn • Govindas	Movement	Possible edges
Environment	Lund	Super Mario world	Network
State (S)	• Hungry? • Teaching? • Salary?	Place in the world • Goombas? • Peach?	• Current node • Time left

	Lunch	Mario	Routing
Agent	You	Mario/Luigi	Node
Action Set (A)	• Inspira • Finn Inn • Govindas	Movement	Possible edges
Environment	Lund	Super Mario world	Network
State (S)	• Hungry? • Teaching? • Salary?	Place in the world • Goombas? • Peach?	• Current node • Time left
Reward (R) (Immediate or Delayed)	Food Satisfaction	Points	Total amount of time saved

	Lunch	Mario	Routing
Agent	You	Mario/Luigi	Node
Action Set (A)	• Inspira • Finn Inn • Govindas	Movement	Possible edges
Environment	Lund	Super Mario world	Network
State (S)	• Hungry? • Teaching? • Salary?	Place in the world • Goombas? • Peach?	• Current node • Time left
Reward (R) (Immediate or Delayed)	Food Satisfaction	Points	Total amount of time saved
Policy (π) (Exploitation vs Exploration)	Greedy, ϵ- greedy		
	• Stick to Inspira • Try the new restaurant	• Jump on the goomba • Go down the pipe	• Choose the path with lowest delay • Explore an edge

	Lunch	Mario	Routing
Agent	You	Mario/Luigi	Node
Action Set (A)	• Inspira • Finn Inn • Govindas	Movement	Possible edges
Environment	Lund	Super Mario world	Network
State (S)	• Hungry? • Teaching? • Salary?	Place in the world • Goombas? • Peach?	• Current node • Time left
Reward (R) (Immediate or Delayed)	Food Satisfaction	Points	Total amount of time saved
Policy (π) (Exploitation vs Exploration)	Greedy, ϵ- greedy		
	• Stick to Inspira • Try the new restaurant	• Jump on the goomba • Go down the pipe	• Choose the path with lowest delay • Explore an edge
Value (V)	Expected reward of being in the current state.		

	Lunch	Mario	Routing
Agent	You	Mario/Luigi	Node
Action Set (A)	• Inspira • Finn Inn • Govindas	Movement	Possible edges
Environment	Lund	Super Mario world	Network
State (S)	• Hungry? • Teaching? • Salary?	Place in the world • Goombas? • Peach?	• Current node • Time left
Reward (R) (Immediate or Delayed)	Food Satisfaction	Points	Total amount of time saved
Policy (π) (Exploitation vs Exploration)	Greedy, ϵ- greedy		
	• Stick to Inspira • Try the new restaurant	• Jump on the goomba • Go down the pipe	• Choose the path with lowest delay • Explore an edge
Value (V)	Expected reward of being in the current state.		
Q-Value (Q)	Similar to V but maps state-action pairs to rewards.		

Reinforcement learning

- The routing problem can be formalised as a Markov Decision Process (**MDP**) consisting of:

Reinforcement learning

- The routing problem can be formalised as a Markov Decision Process (**MDP**) consisting of:
    - Finite set of states (**S**)

Reinforcement learning

- The routing problem can be formalised as a Markov Decision Process (**MDP**) consisting of:
    - Finite set of states (**S**)
    - Finite set of actions from each state (**A**)

Reinforcement learning

- The routing problem can be formalised as a Markov Decision Process (**MDP**) consisting of:
  - Finite set of states (**S**)
  - Finite set of actions from each state (**A**)
  - Probability of transition from one state to another (**P**)

Reinforcement learning

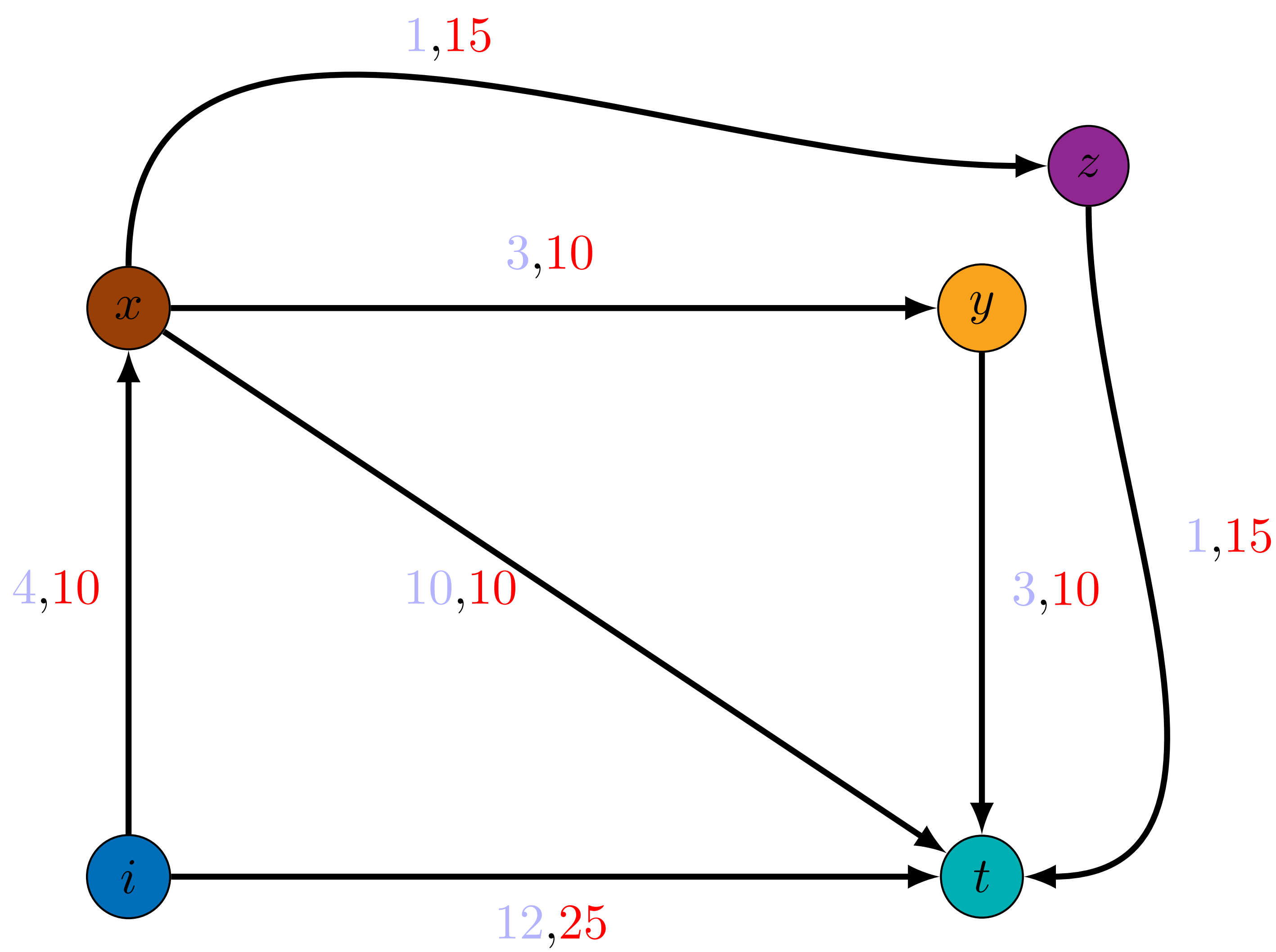
- The routing problem can be formalised as a Markov Decision Process (**MDP**) consisting of:
  - Finite set of states (**S**)
  - Finite set of actions from each state (**A**)
  - Probability of transition from one state to another (**P**)
  - Rewards from each state (**R**)

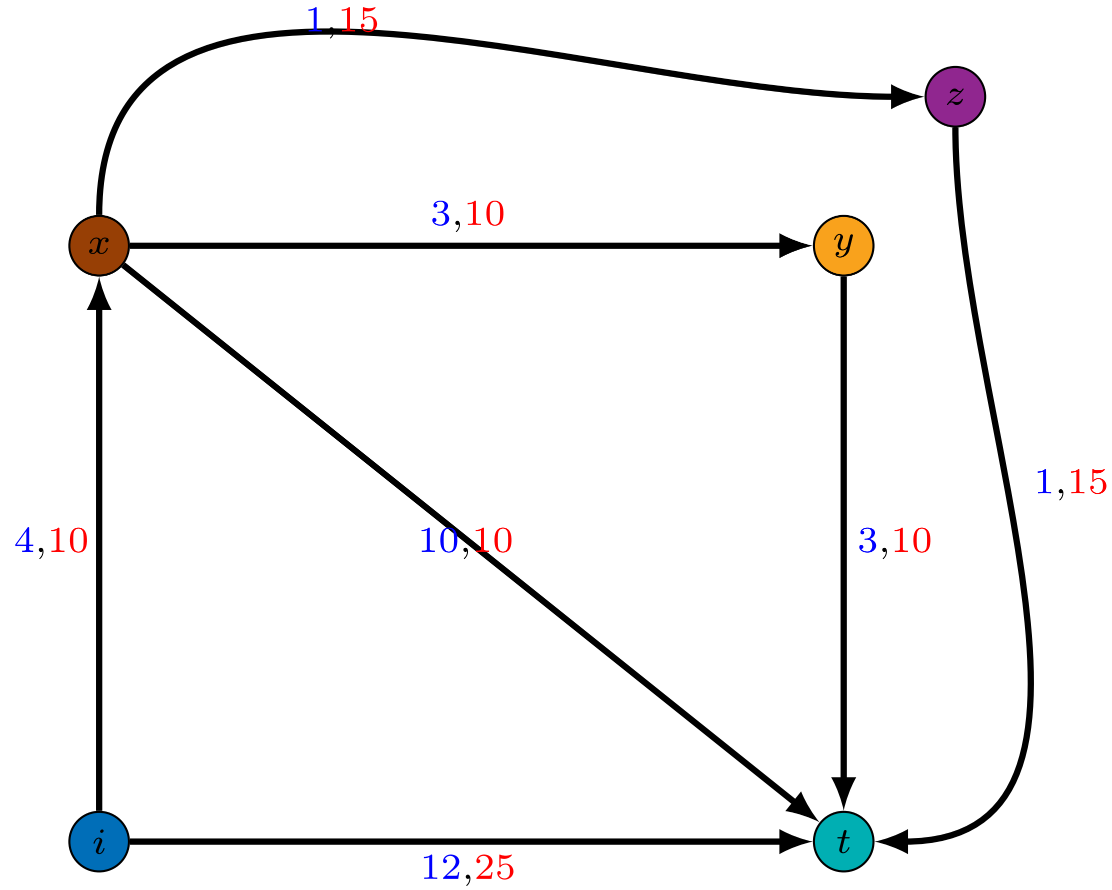
Reinforcement learning

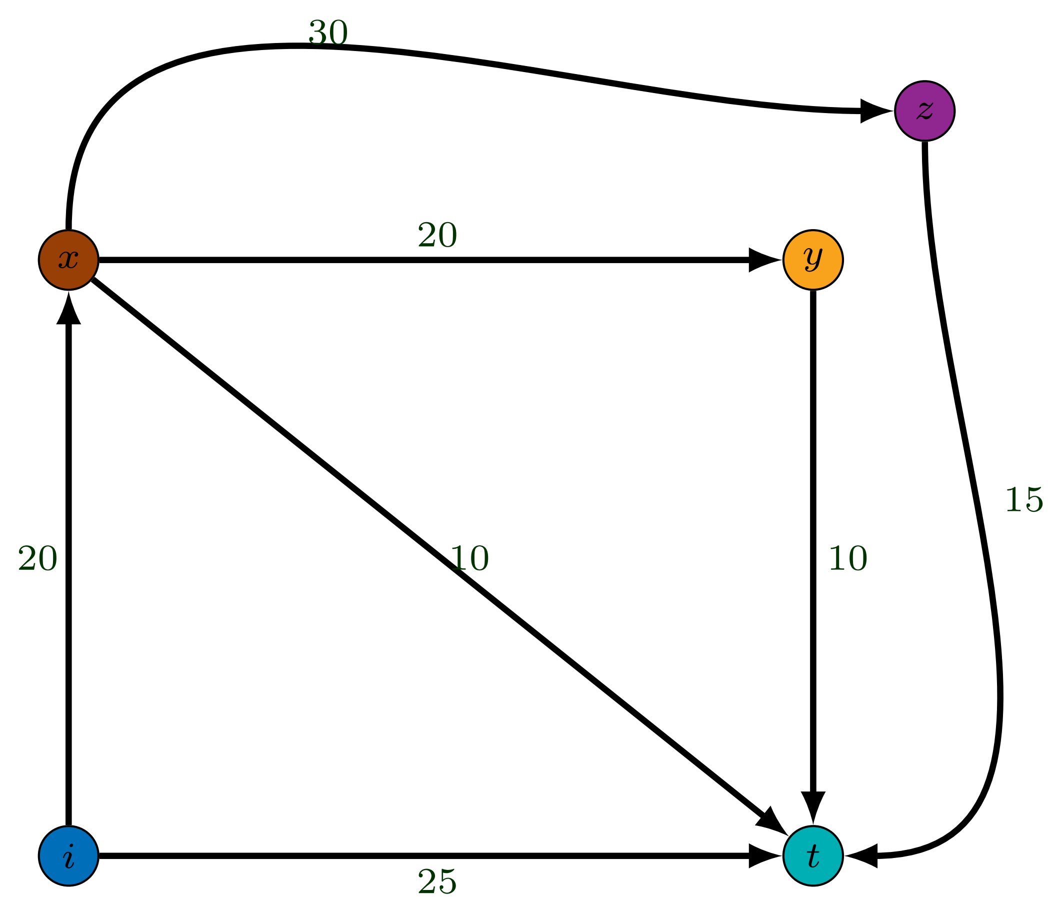
- The routing problem can be formalised as a Markov Decision Process (**MDP**) consisting of:
  - Finite set of states (**S**)
  - Finite set of actions from each state (**A**)
  - Probability of transition from one state to another (**P**)
  - Rewards from each state (**R**)
- In our state space, we encode the current vertex and total time elapsed from the beginning

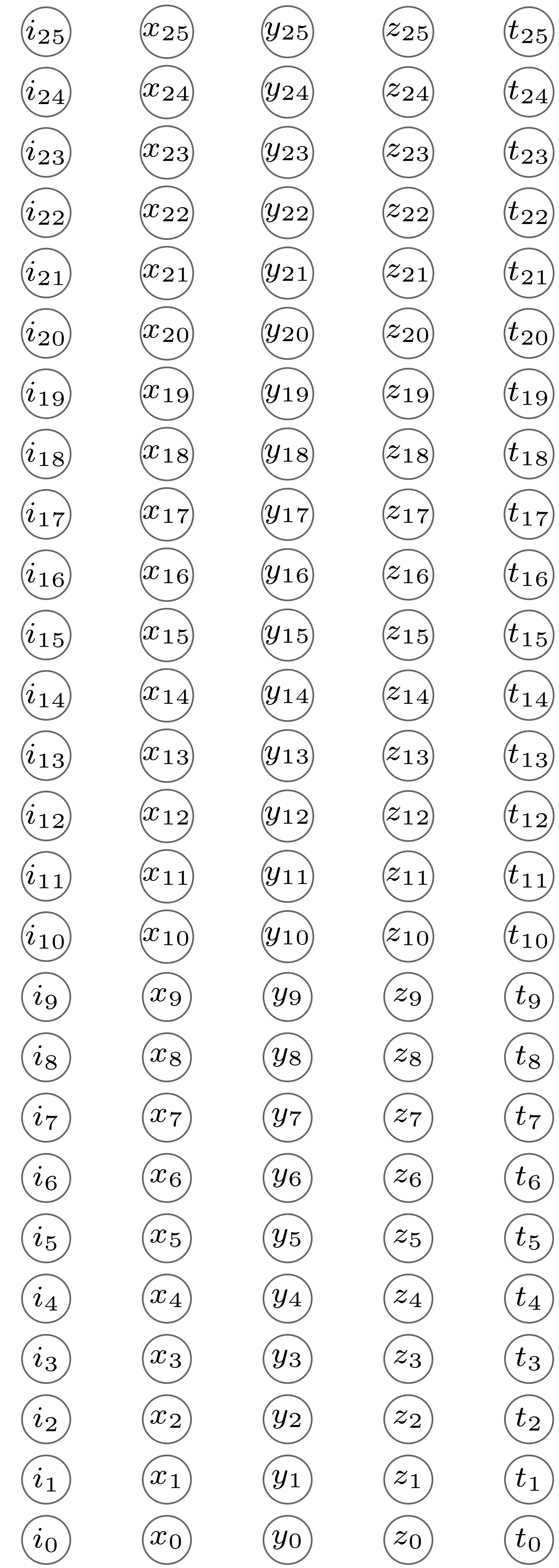
State-Space Example

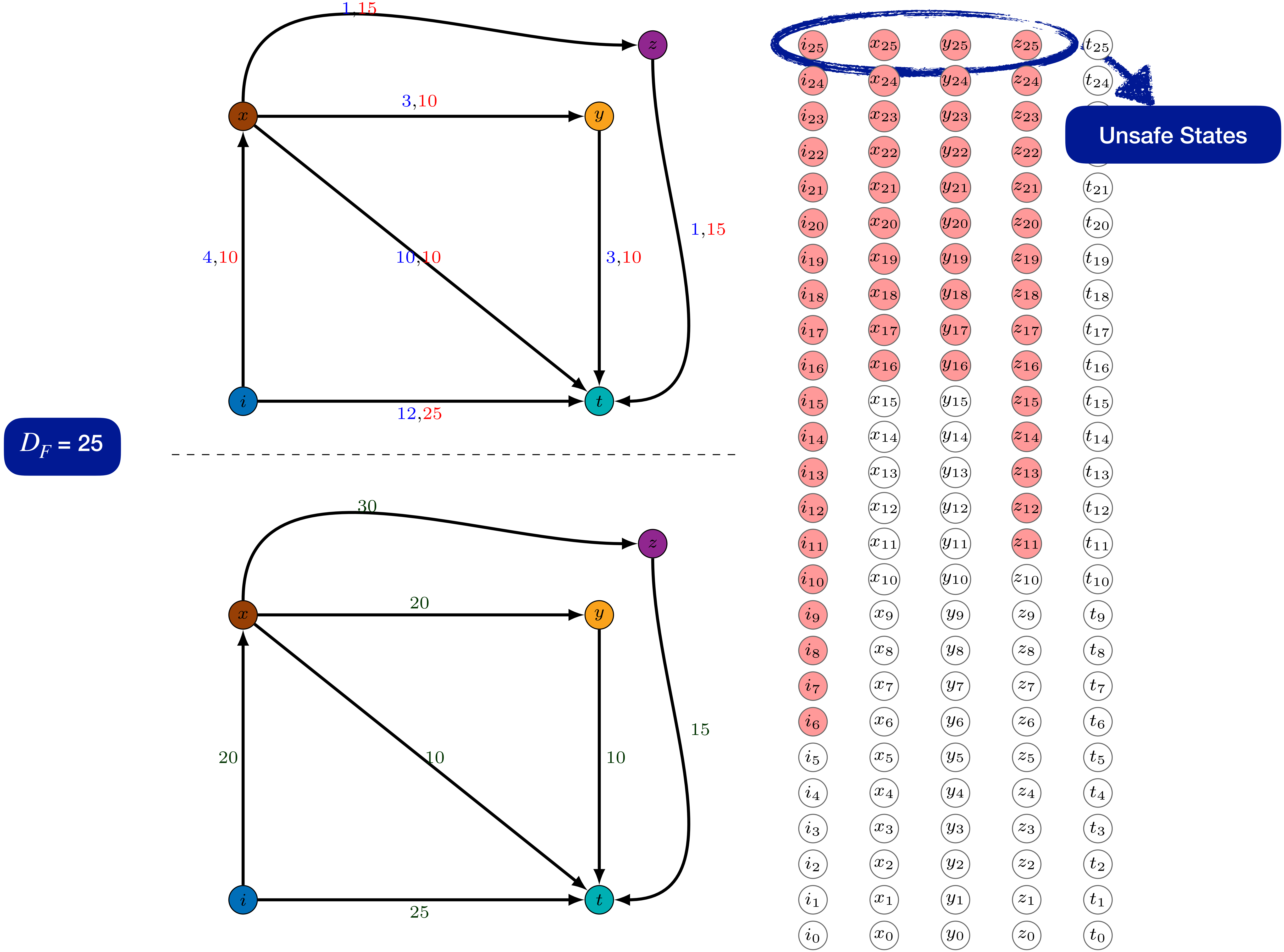
- Consider a scenario with :-
    - Source node **i**, destination node **t**
    - Maximum admissible time, $D_F = 25$




1,15
z
x
3,10
y
4,10
10,10
3,10
1,15
i
t
12,25

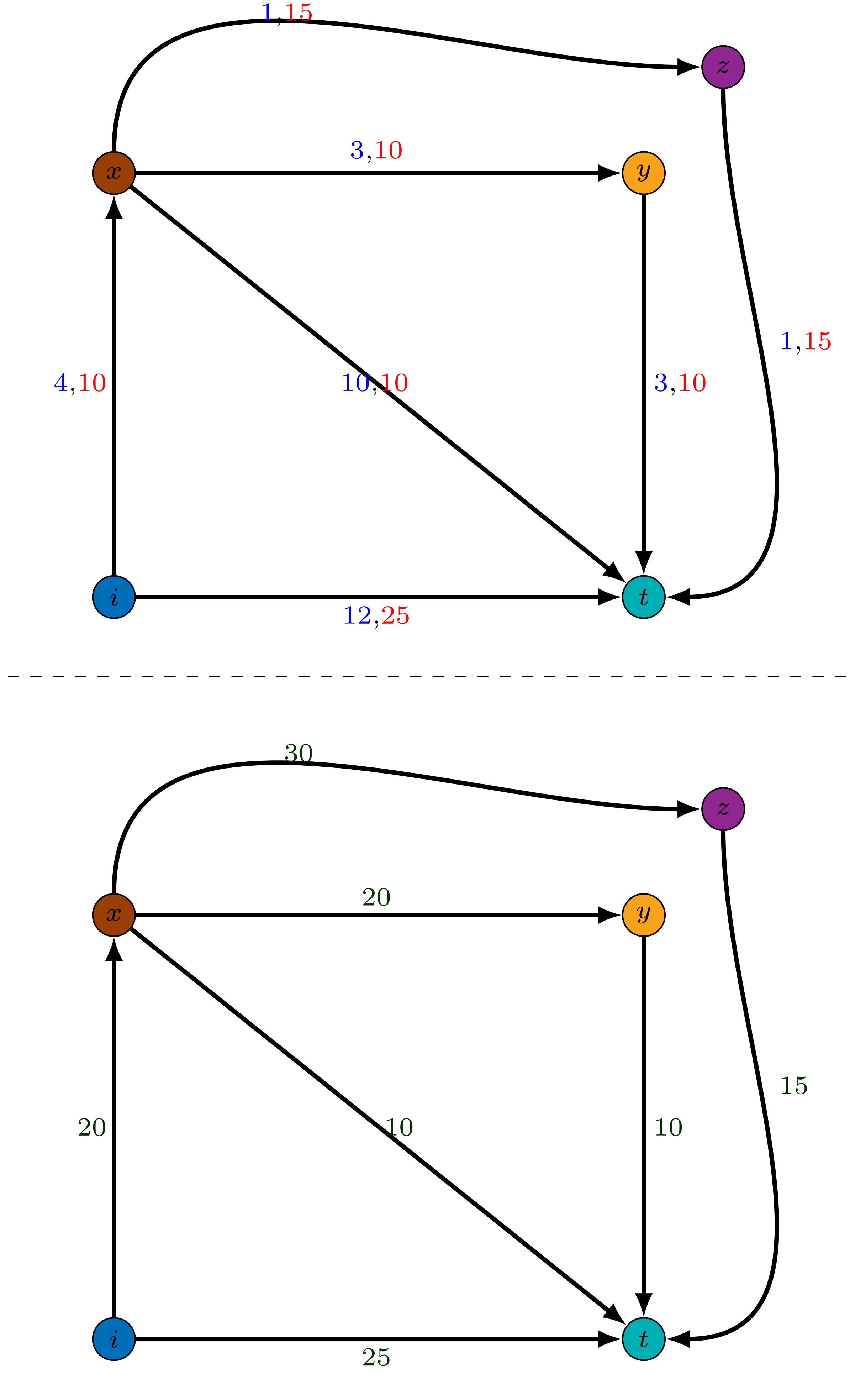

30
z
x
20
y
20
10
10
15
i
t
25


i_{25} x_{25} y_{25} z_{25} t_{25}
i_{24} x_{24} y_{24} z_{24} t_{24}
i_{23} x_{23} y_{23} z_{23} t_{23}
i_{22} x_{22} y_{22} z_{22} t_{22}
i_{21} x_{21} y_{21} z_{21} t_{21}
i_{20} x_{20} y_{20} z_{20} t_{20}
i_{19} x_{19} y_{19} z_{19} t_{19}
i_{18} x_{18} y_{18} z_{18} t_{18}
i_{17} x_{17} y_{17} z_{17} t_{17}
i_{16} x_{16} y_{16} z_{16} t_{16}
i_{15} x_{15} y_{15} z_{15} t_{15}
i_{14} x_{14} y_{14} z_{14} t_{14}
i_{13} x_{13} y_{13} z_{13} t_{13}
i_{12} x_{12} y_{12} z_{12} t_{12}
i_{11} x_{11} y_{11} z_{11} t_{11}
i_{10} x_{10} y_{10} z_{10} t_{10}
i_{9} x_{9} y_{9} z_{9} t_{9}
i_{8} x_{8} y_{8} z_{8} t_{8}
i_{7} x_{7} y_{7} z_{7} t_{7}
i_{6} x_{6} y_{6} z_{6} t_{6}
i_{5} x_{5} y_{5} z_{5} t_{5}
i_{4} x_{4} y_{4} z_{4} t_{4}
i_{3} x_{3} y_{3} z_{3} t_{3}
i_{2} x_{2} y_{2} z_{2} t_{2}
i_{1} x_{1} y_{1} z_{1} t_{1}
i_{0} x_{0} y_{0} z_{0} t_{0}

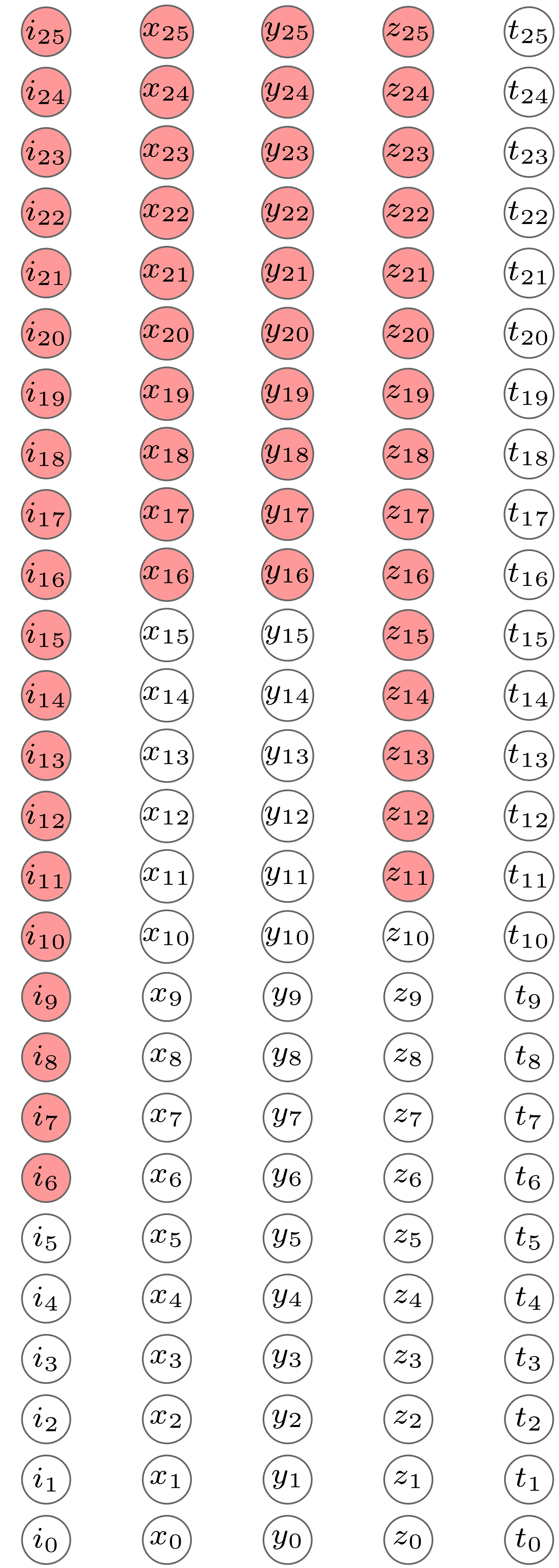


Unsafe States
D_F = 25

D_F = 25

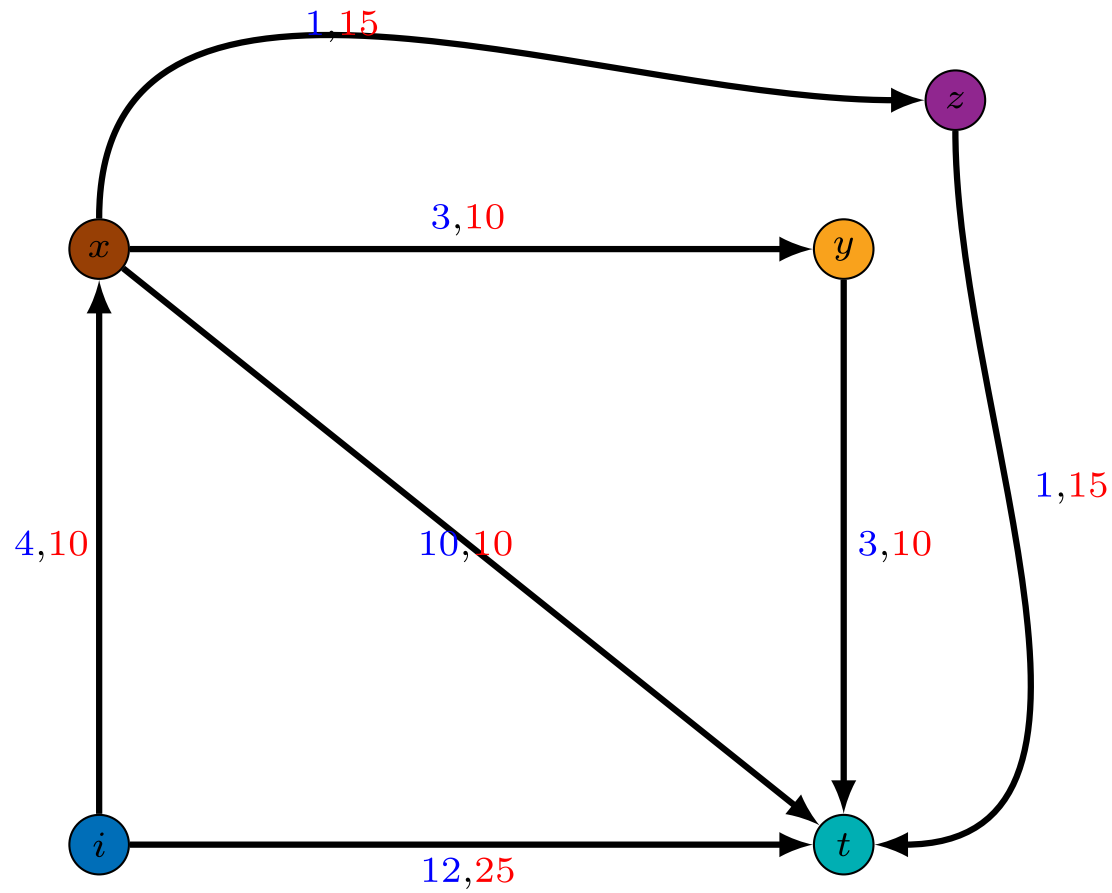


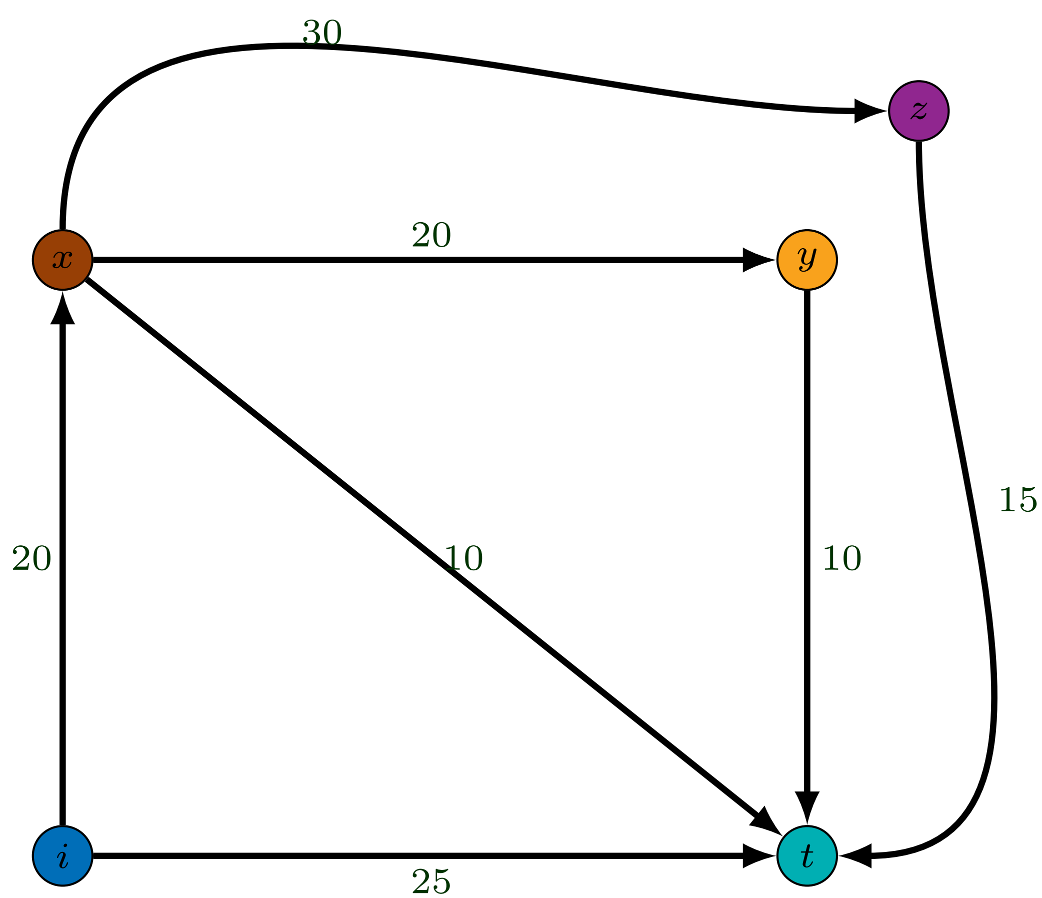
1,15
z
3,10
x
y
1,15
4,10
10,10
3,10
i
t
12,25
30
z
20
x
y
15
20
10
10
i
t
25

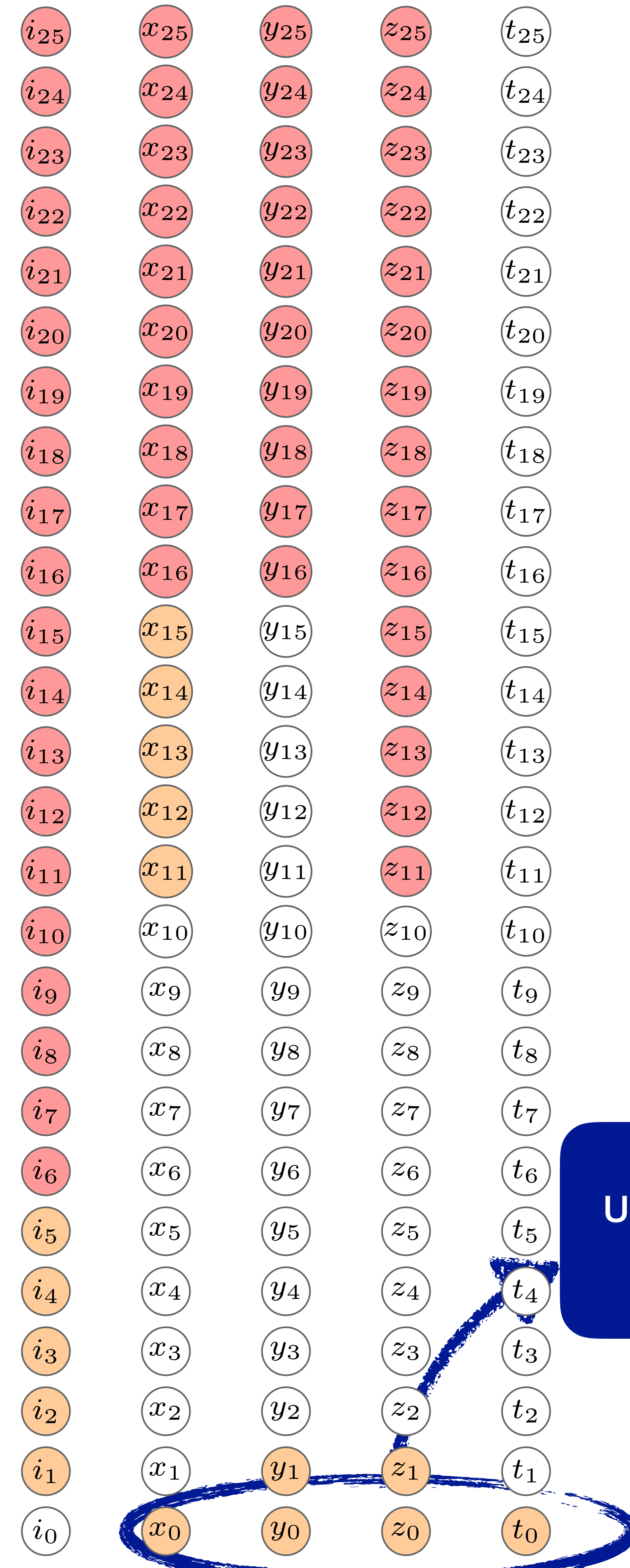


i_{25} x_{25} y_{25} z_{25} t_{25}
i_{24} x_{24} y_{24} z_{24} t_{24}
i_{23} x_{23} y_{23} z_{23} t_{23}
i_{22} x_{22} y_{22} z_{22} t_{22}
i_{21} x_{21} y_{21} z_{21} t_{21}
i_{20} x_{20} y_{20} z_{20} t_{20}
i_{19} x_{19} y_{19} z_{19} t_{19}
i_{18} x_{18} y_{18} z_{18} t_{18}
i_{17} x_{17} y_{17} z_{17} t_{17}
i_{16} x_{16} y_{16} z_{16} t_{16}
i_{15} x_{15} y_{15} z_{15} t_{15}
i_{14} x_{14} y_{14} z_{14} t_{14}
i_{13} x_{13} y_{13} z_{13} t_{13}
i_{12} x_{12} y_{12} z_{12} t_{12}
i_{11} x_{11} y_{11} z_{11} t_{11}
i_{10} x_{10} y_{10} z_{10} t_{10}
i_9 x_9 y_9 z_9 t_9
i_8 x_8 y_8 z_8 t_8
i_7 x_7 y_7 z_7 t_7
i_6 x_6 y_6 z_6 t_6
i_5 x_5 y_5 z_5 t_5
i_4 x_4 y_4 z_4 t_4
i_3 x_3 y_3 z_3 t_3
i_2 x_2 y_2 z_2 t_2
i_1 x_1 y_1 z_1 t_1
i_0 x_0 y_0 z_0 t_0

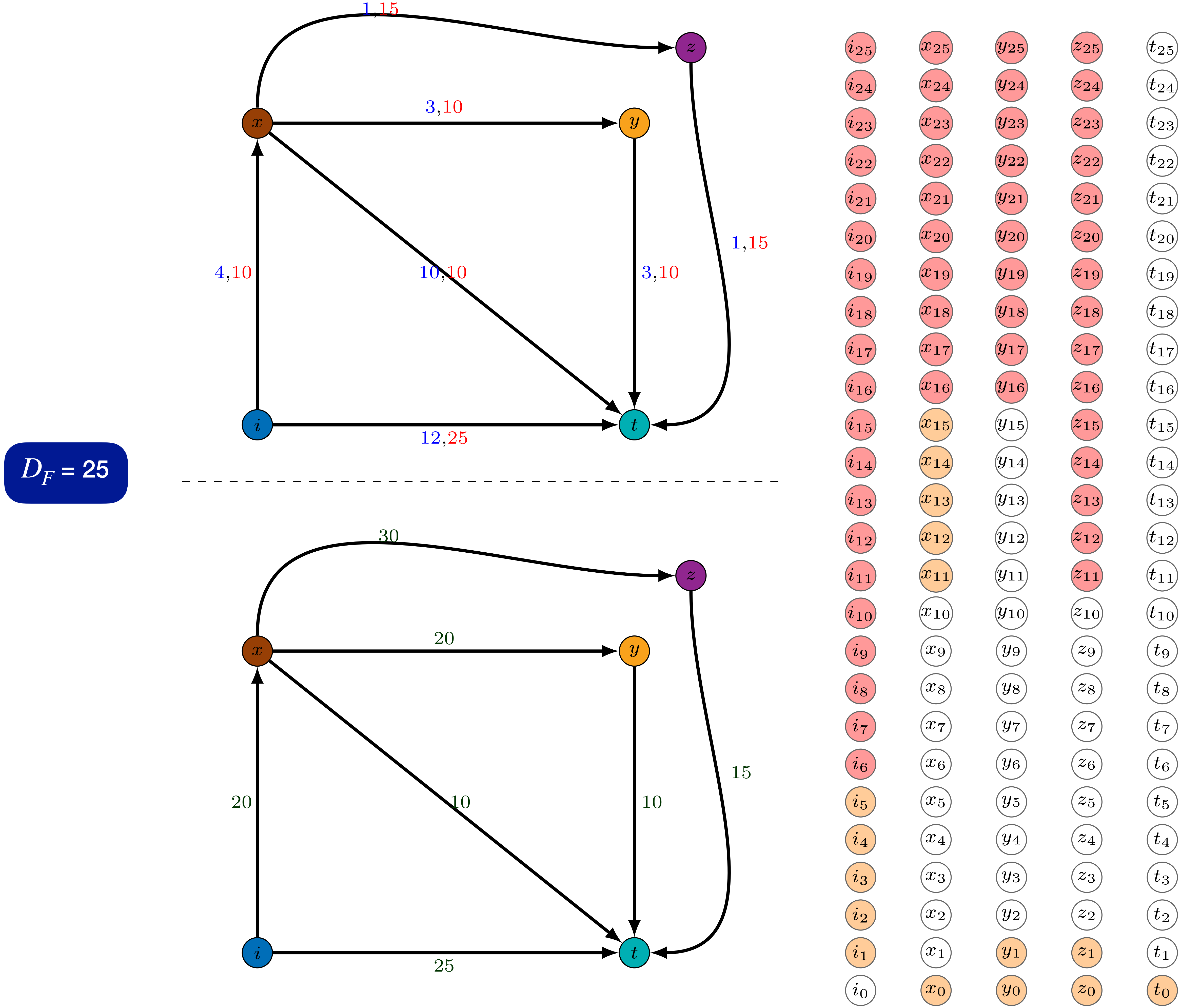
D_F = 25







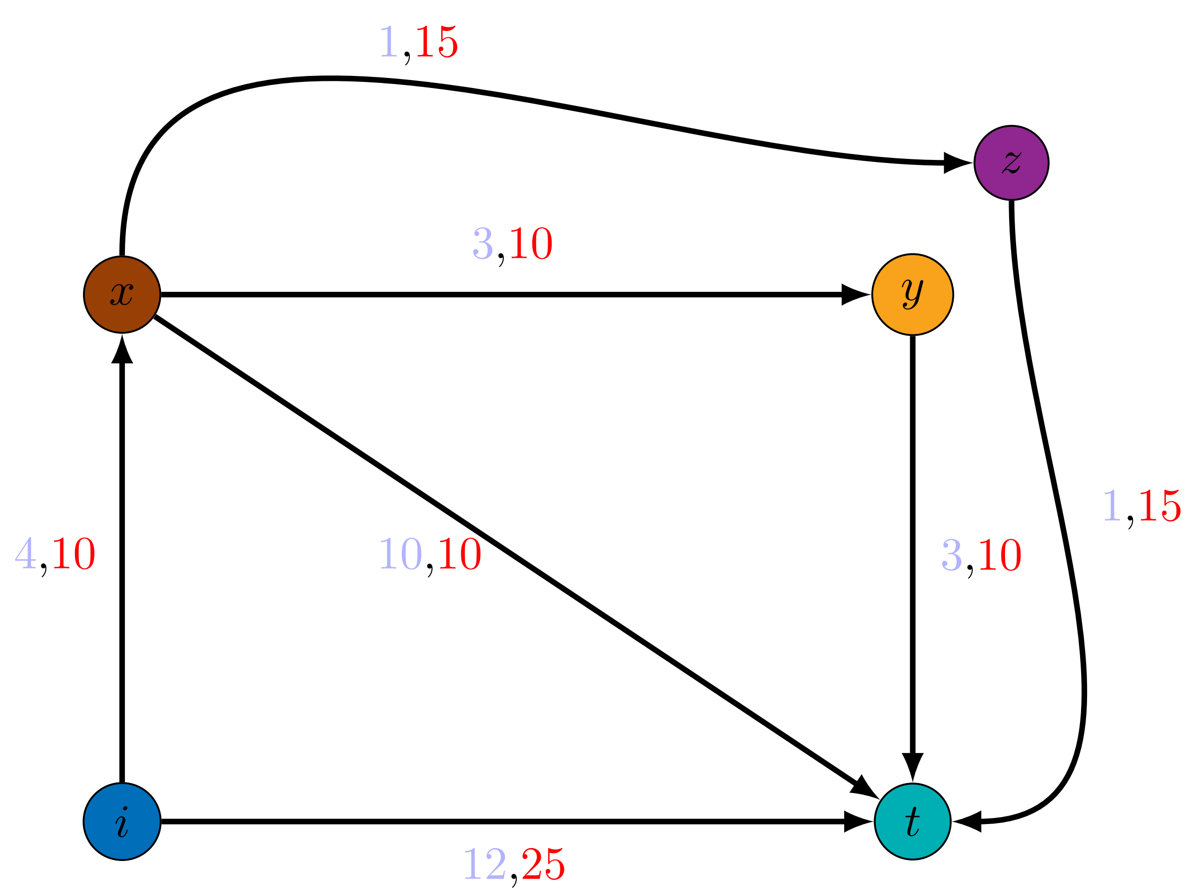
Unreachable states



D_F = 25
1,15
3,10
4,10
10,10
3,10
1,15
12,25
30
20
20
10
10
15
25
i_{25} x_{25} y_{25} z_{25} t_{25}
i_{24} x_{24} y_{24} z_{24} t_{24}
i_{23} x_{23} y_{23} z_{23} t_{23}
i_{22} x_{22} y_{22} z_{22} t_{22}
i_{21} x_{21} y_{21} z_{21} t_{21}
i_{20} x_{20} y_{20} z_{20} t_{20}
i_{19} x_{19} y_{19} z_{19} t_{19}
i_{18} x_{18} y_{18} z_{18} t_{18}
i_{17} x_{17} y_{17} z_{17} t_{17}
i_{16} x_{16} y_{16} z_{16} t_{16}
i_{15} x_{15} y_{15} z_{15} t_{15}
i_{14} x_{14} y_{14} z_{14} t_{14}
i_{13} x_{13} y_{13} z_{13} t_{13}
i_{12} x_{12} y_{12} z_{12} t_{12}
i_{11} x_{11} y_{11} z_{11} t_{11}
i_{10} x_{10} y_{10} z_{10} t_{10}
i_9 x_9 y_9 z_9 t_9
i_8 x_8 y_8 z_8 t_8
i_7 x_7 y_7 z_7 t_7
i_6 x_6 y_6 z_6 t_6
i_5 x_5 y_5 z_5 t_5
i_4 x_4 y_4 z_4 t_4
i_3 x_3 y_3 z_3 t_3
i_2 x_2 y_2 z_2 t_2
i_1 x_1 y_1 z_1 t_1
i_0 x_0 y_0 z_0 t_0

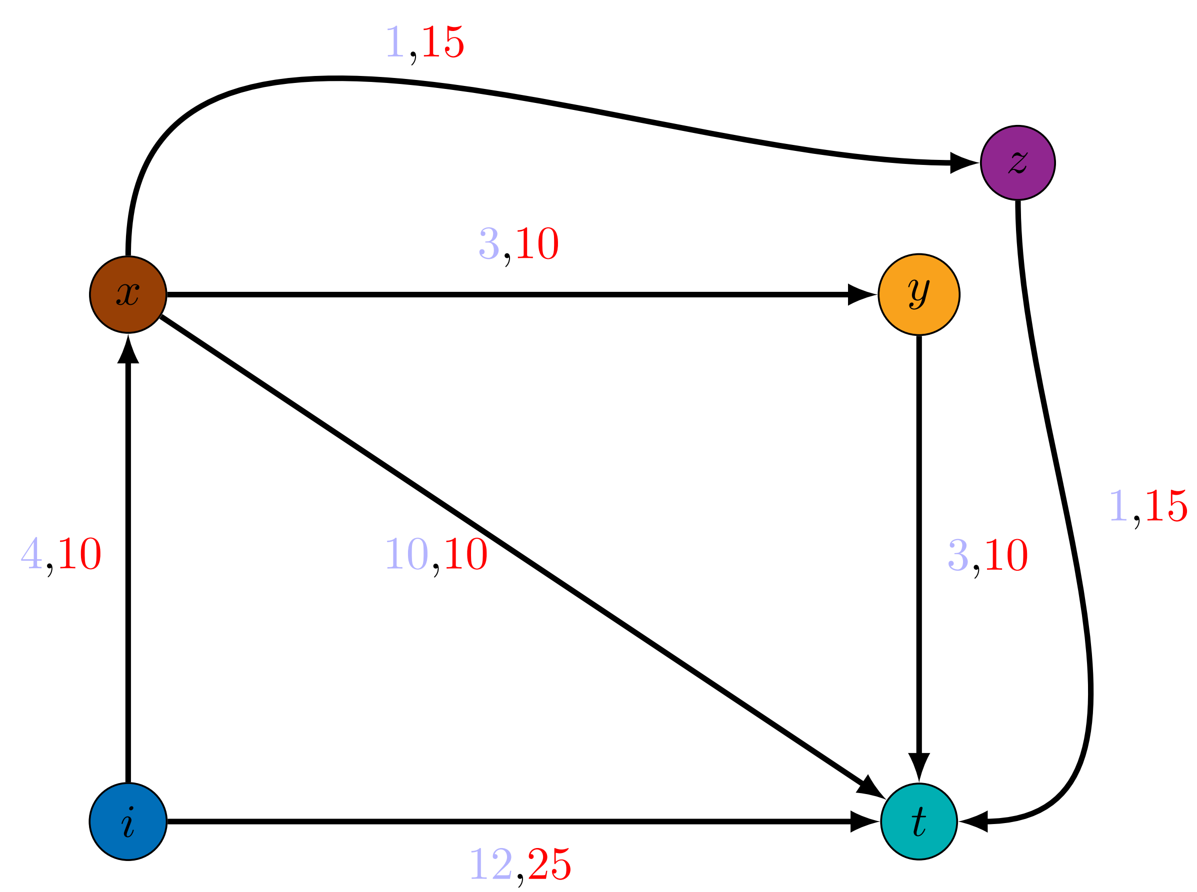
Dijkstras shortest path

- Versatile algorithm to find shortest path from starting to target node in a weighted graph



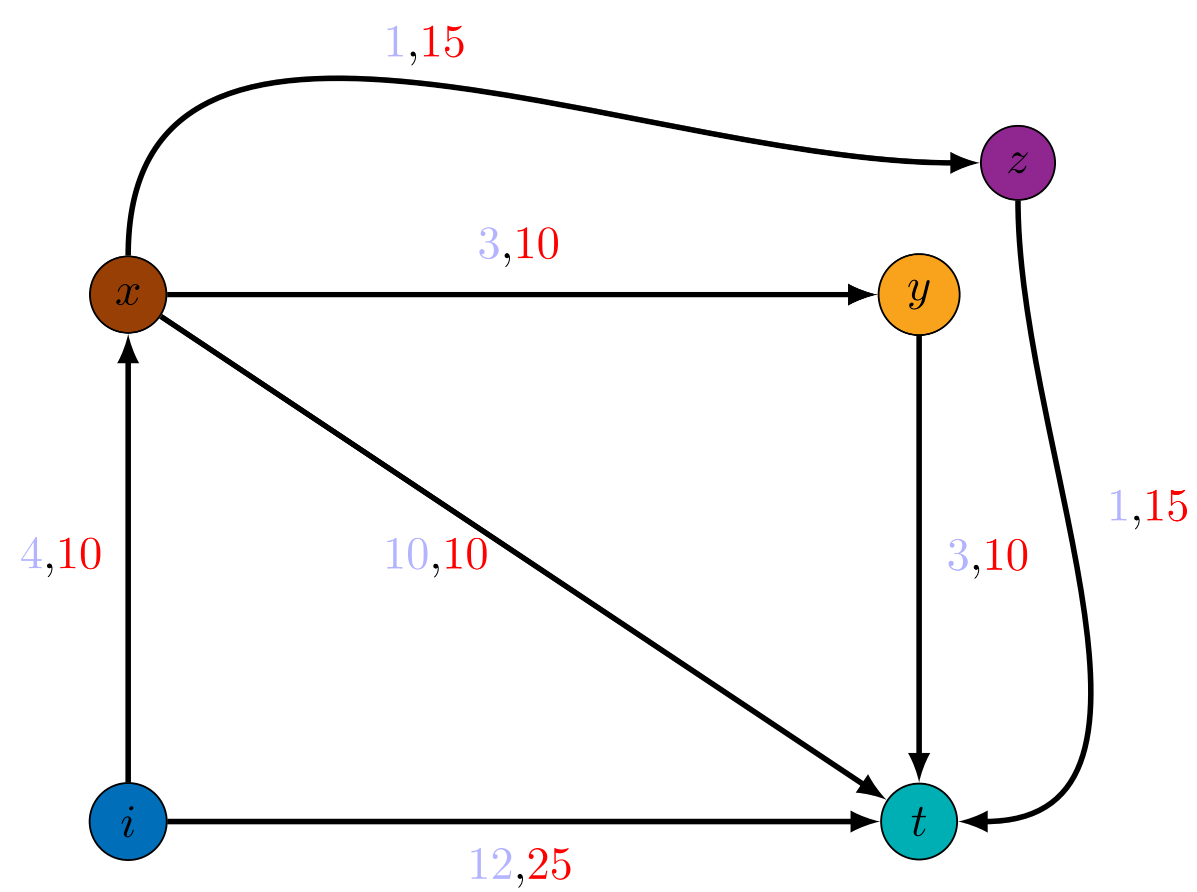
Dijkstras shortest path

- Versatile algorithm to find shortest path from starting to target node in a weighted graph
- Forms the basis of pre-processing stage

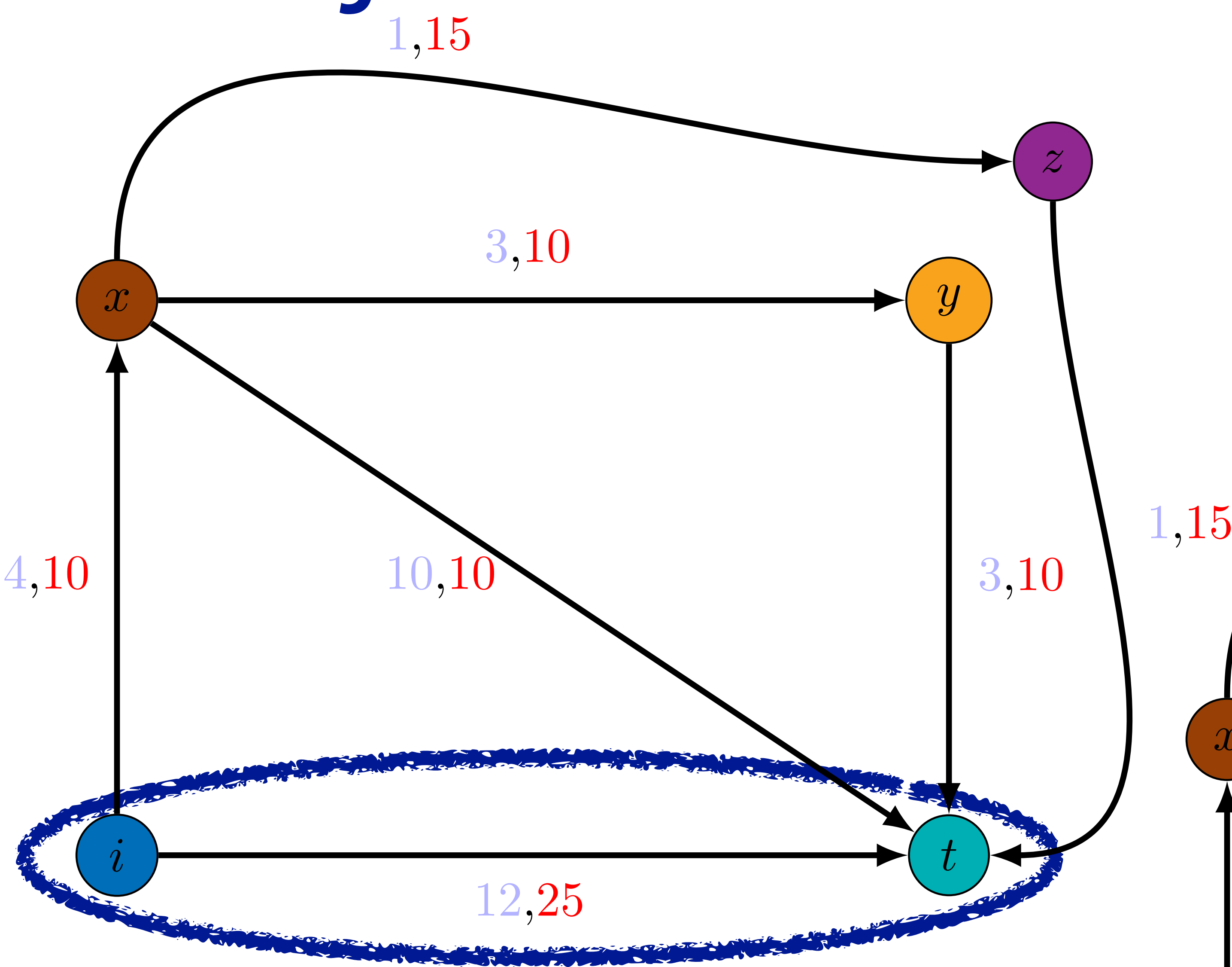


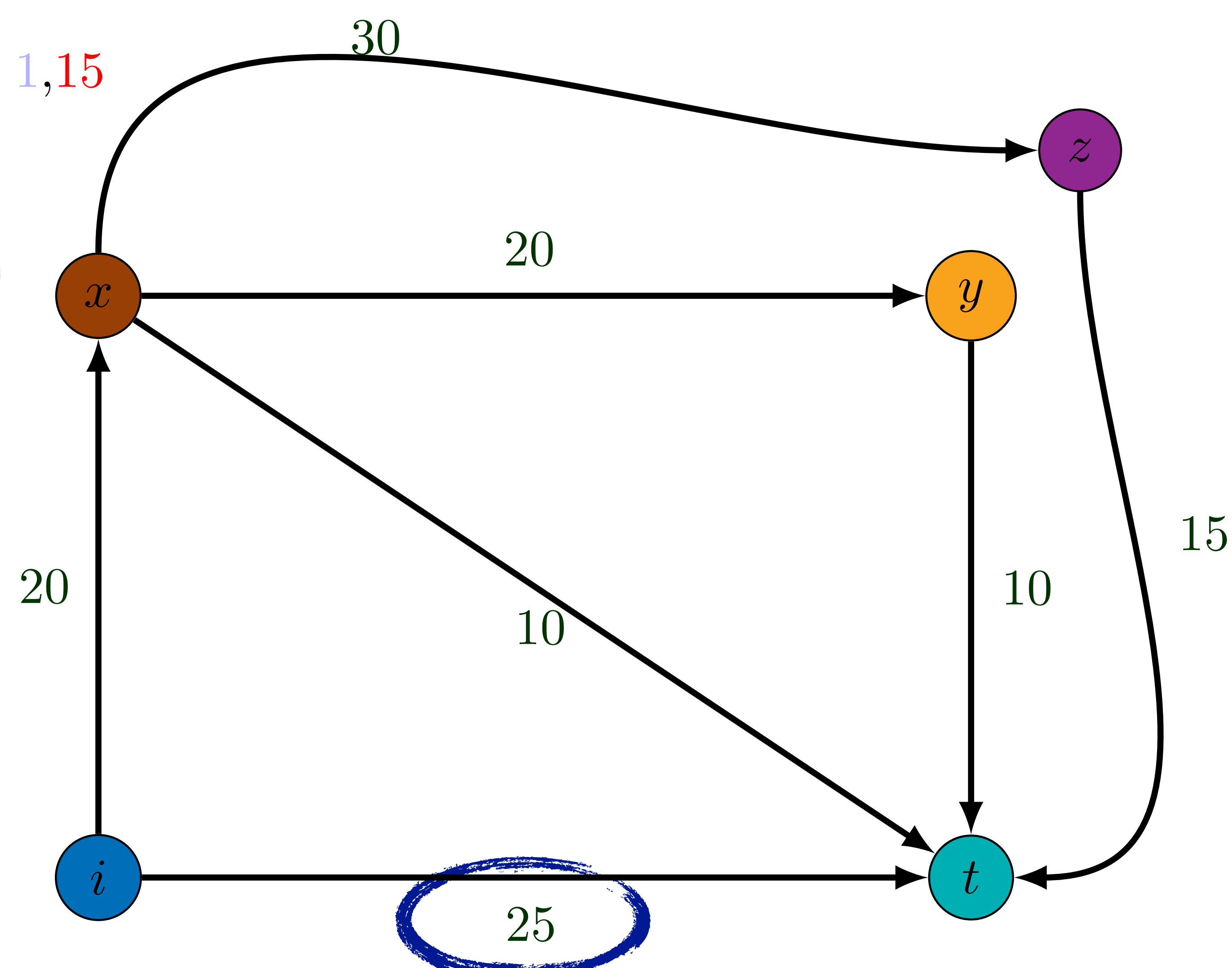
Dijkstras shortest path

- Versatile algorithm to find shortest path from starting to target node in a weighted graph
- Forms the basis of pre-processing stage
- Gives the tightest deadline D_F that can be guaranteed over each link

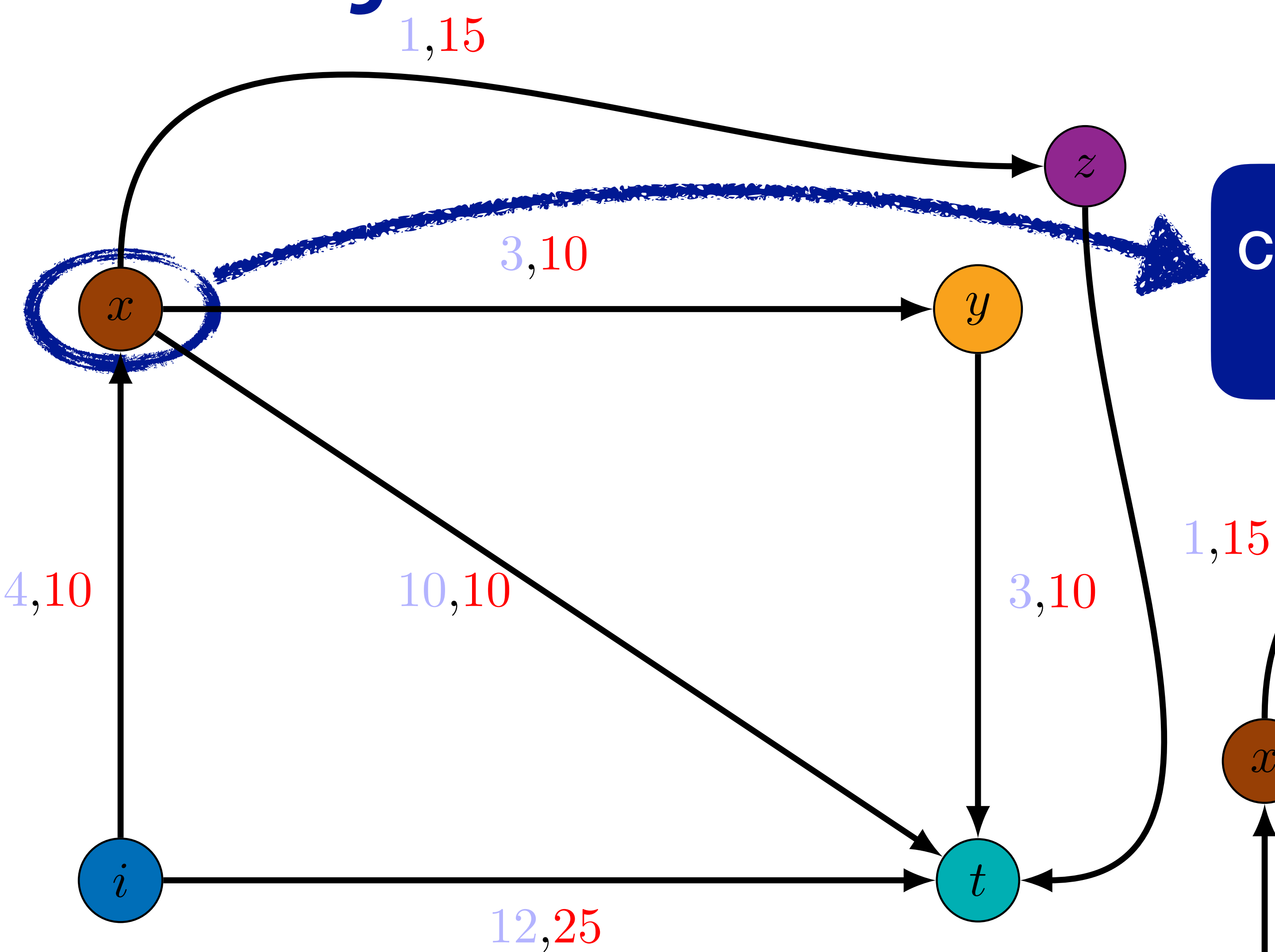


Dijkstras shortest path

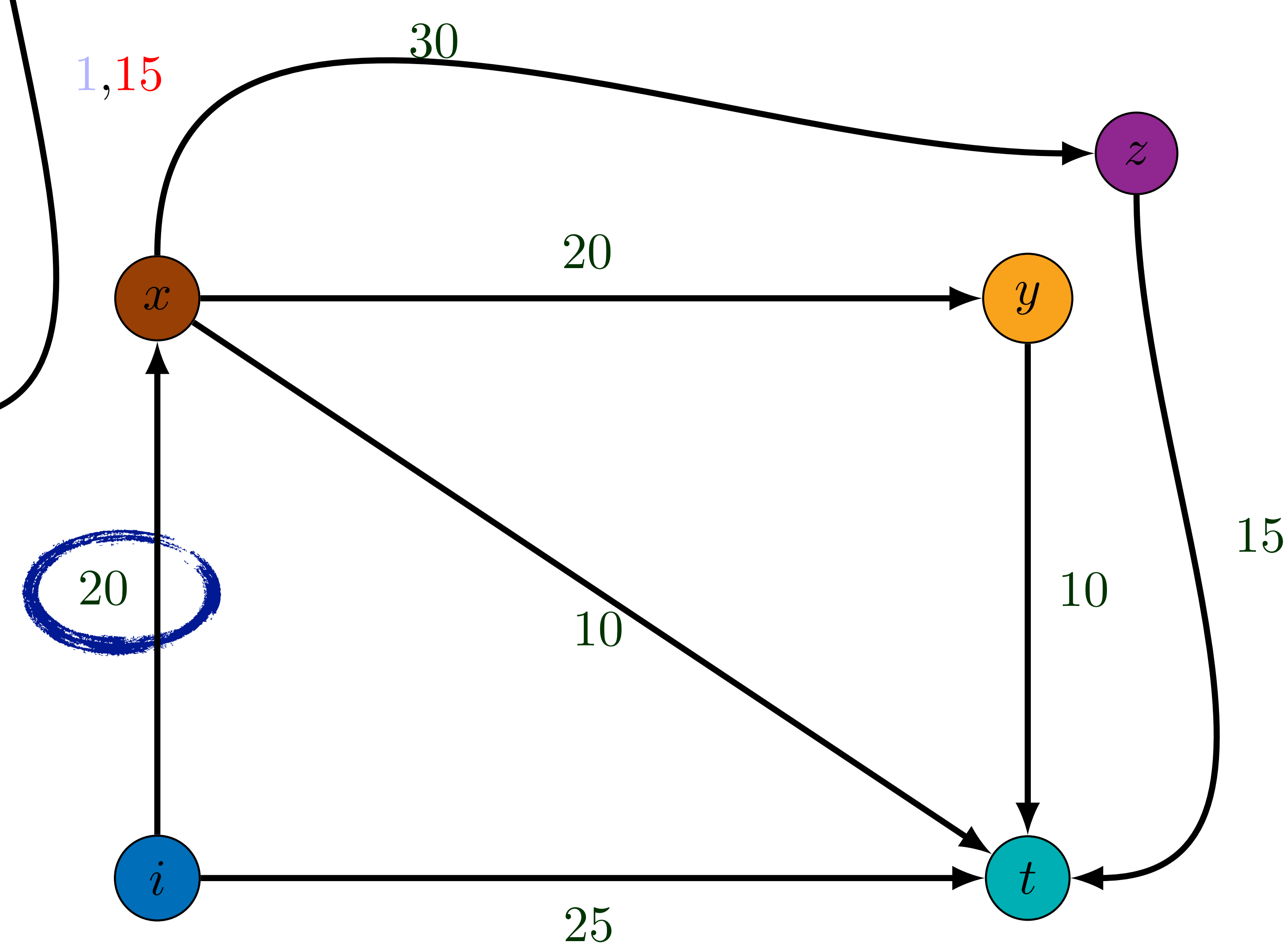




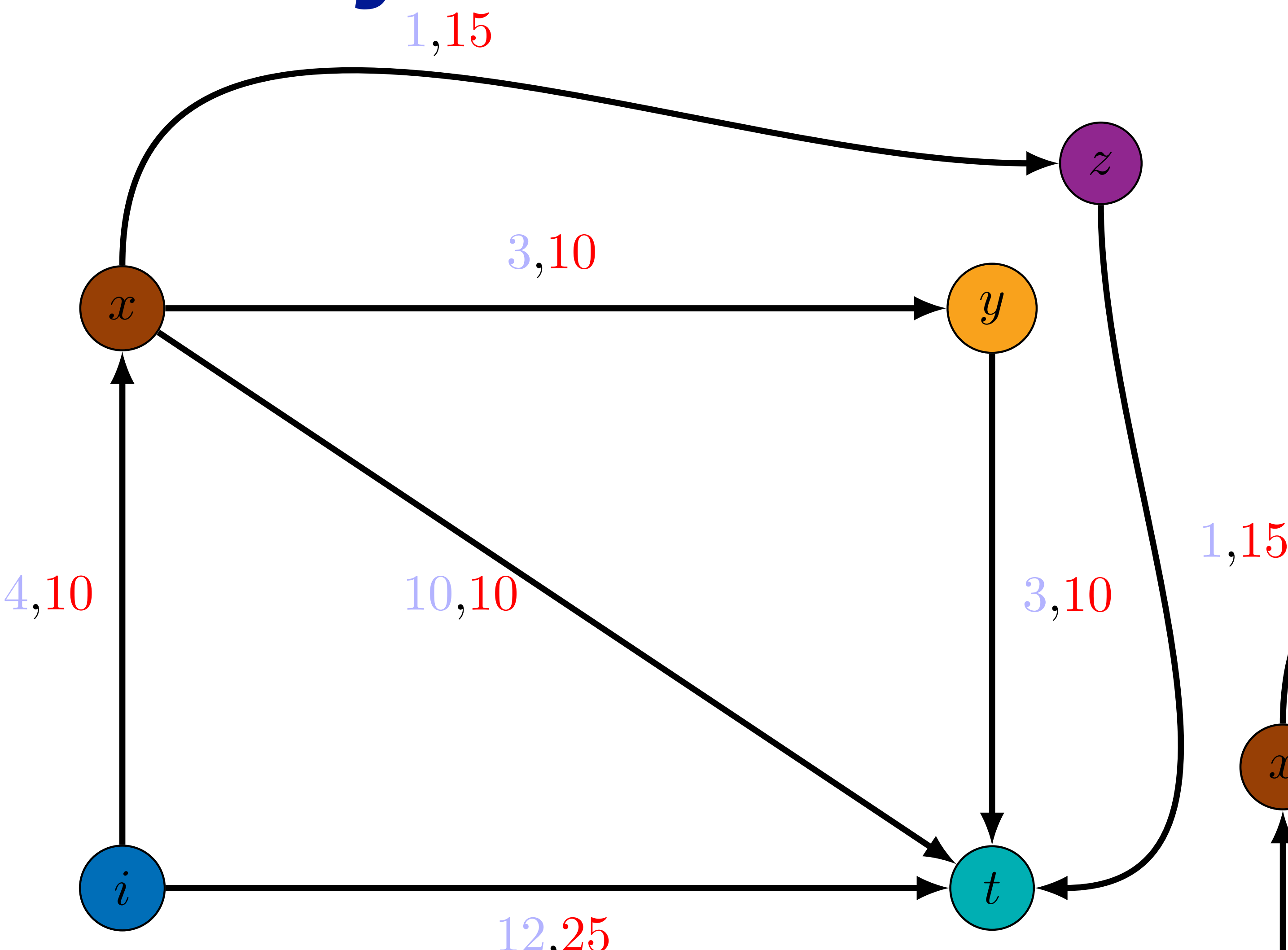
Dijkstras shortest path

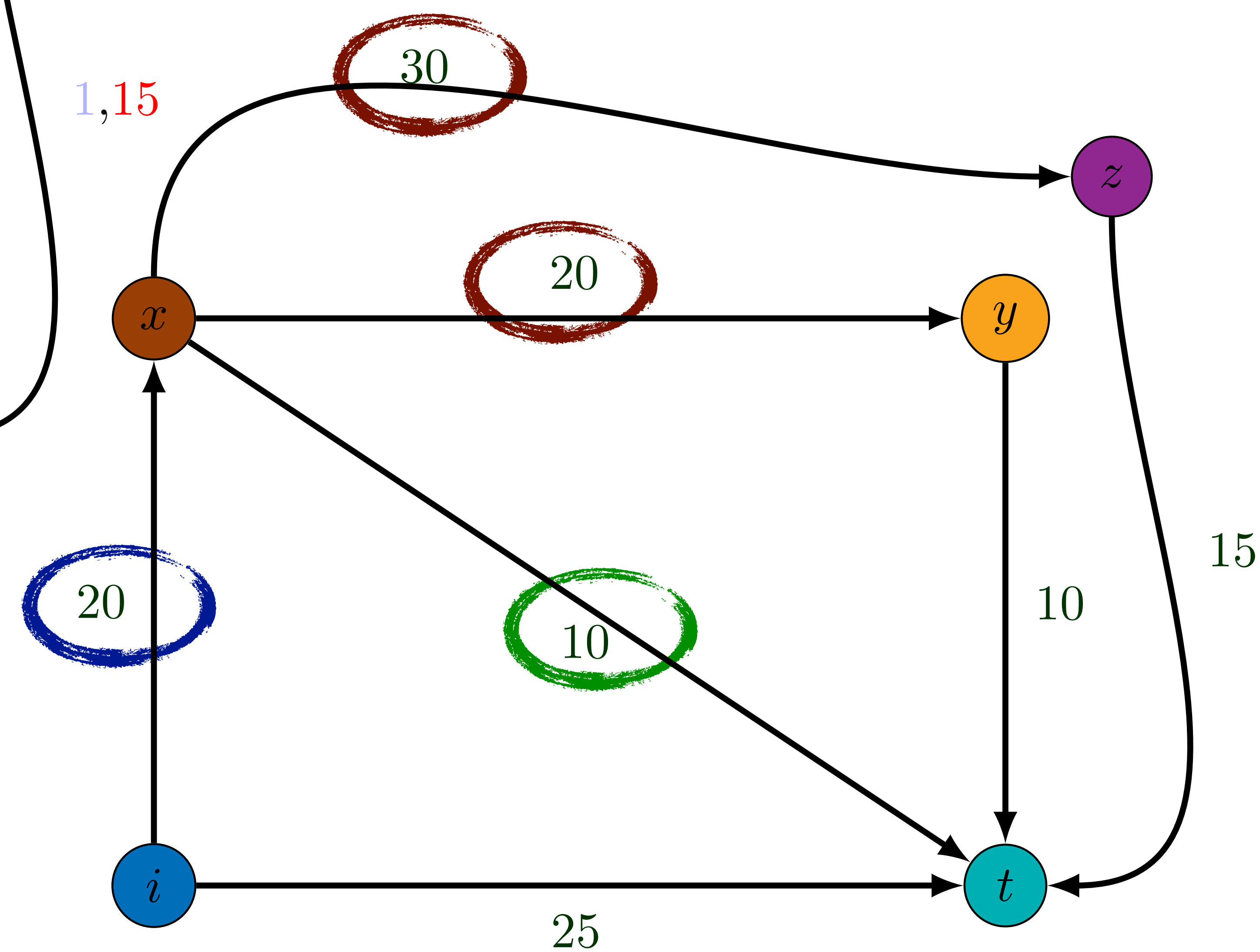


Calculate shortest path from next node to destination



Dijkstras shortest path





Reduced computation when run from destination to source

Pre-processing phase

- Required irrespective of the algorithm used to provide guarantees

Pre-processing phase

- Required irrespective of the algorithm used to provide guarantees
- Uses Dijkstra's algorithm to find smallest delay that can be guaranteed to the destination

Pre-processing phase

- Required irrespective of the algorithm used to provide guarantees
- Uses Dijkstra's algorithm to find smallest delay that can be guaranteed to the destination
- Calculated for each link in the network

Routing with safe reinforcement learning

Routing with safe reinforcement learning

- Chooses optimal path in dynamic environmental conditions

Routing with safe reinforcement learning

- Chooses optimal path in dynamic environmental conditions
- While ensuring never to violate deadline restrictions

Thanks to the truncated state-space

Routing with safe reinforcement learning

- Chooses optimal path in dynamic environmental conditions
- While ensuring never to violate deadline restrictions
- Algorithm has two phases

Routing with safe reinforcement learning

- Chooses optimal path in dynamic environmental conditions
- While ensuring never to violate deadline restrictions
- Algorithm has two phases
  - Pre-processing phase (Dijkstras algorithm)

Routing with safe reinforcement learning

- Chooses optimal path in dynamic environmental conditions
- While ensuring never to violate deadline restrictions
- Algorithm has two phases
  - Pre-processing phase (Dijkstras algorithm)
  - Run-time phase

Routing with safe reinforcement learning

- Chooses optimal path in dynamic environmental conditions
- While ensuring never to violate deadline restrictions
- Algorithm has two phases
    - Pre-processing phase (Dijkstras algorithm)
    - Run-time phase
- The environment returns the reward after each episode

Routing with safe reinforcement learning

- Value iteration updates a value of being in a state, improving future episodes

Routing with safe reinforcement learning

- Value iteration updates a value of being in a state, improving future episodes

- Popular Methods
  - Monte Carlo value estimation
  - Temporal Difference(TD) Learning

Routing with safe reinforcement learning

- Value iteration updates a value of being in a state, improving future episodes

- Popular Methods

  - Monte Carlo value estimation

  - Temporal Difference(TD) Learning

- Exploration using ϵ-greedy approach

  - Taking only safe edges ensures safe learning

Value Update

- Monte-Carlo Methods

Value Update

- Monte-Carlo Methods
  - Useful but leads a lot of back-propagation.
  - This increases messages in the network which we want to reduce

Value Update

- Monte-Carlo Methods
  - Useful but leads a lot of back-propagation.
  - This increases messages in the network which we want to reduce
- Temporal Difference(TD) Learning
  - Learning without waiting for episode to end

Value Update

- Monte-Carlo Methods
    - Useful but leads a lot of back-propagation.
    - This increases messages in the network which we want to reduce
- Temporal Difference(TD) Learning
  - Learning without waiting for episode to end
  - Special case TD(0) depends only on the value of current and next state-action pairs
  - $Q(s,a) = Q(s,a) + \alpha \cdot (R + \max(\gamma Q(s',a')) - Q(s,a))$

Exploration

- ϵ-greedy exploration gives stochastic convergence guarantees
- Ensures that all feasible paths in the network will eventually be explored

Run-time phase

- Run at every node when a packet arrives

Run-time phase

- Run at every node when a packet arrives
- Requires only current and next node Q-values
    - TD-Learning removes reward back propagation

Run-time phase

- Run at every node when a packet arrives
- Requires only current and next node Q-values
    - TD-Learning removes reward back propagation
- Decentralised approach as each node makes its own decisions

Run-time phase

- Run at every node when a packet arrives
- Requires only current and next node Q-values
  - TD-Learning removes reward back propagation
- Decentralised approach as each node makes its own decisions

Algorithm 1 Node Logic (u)

for Every packet **do**
    if u = source node i **then**
        $D_u = D_F$ // Initialise the deadline for each episode
        $\delta_{it} = 0$ // Initiliaze total delay for packet = 0
    for each edge $(u \rightarrow v)$ **do**
        if $c_{uv} > D_u$ **then** // Unsafe Edge
            $P(u|v) = 0$
        else if $Q(u, v) = max(Q(u, a \in A))$ **then**
            $P(u|v) = (1 - \epsilon)$
        else
            $P(u|v) = \epsilon/(size(\mathcal{F} - 1))$
    Choose edge $(u \rightarrow v)$ with P
    Observe δ_{uv}
    δ_{it} += δ_{uv}
    $D_v = D_u - \delta_{uv}$
    R = Environment Reward Function(v, δ_{it})
    $Q(u, v)$ = Value iteration from Equation
    if $v = t$ **then**
        DONE

Run-time phase

- Run at every node when a packet arrives
- Requires only current and next node Q-values
  - TD-Learning removes reward back propagation
- Decentralised approach as each node makes its own decisions

Algorithm 1 Node Logic (u)

for Every packet **do**
    if $u =$ source node i **then**
        $D_u = D_F$ // Initialise the deadline for each episode
        $\delta_{it} = 0$ // Initiliaze total delay for packet $= 0$
    for each edge $(u \to v)$ **do**
        if $c_{uv} > D_u$ **then** // Unsafe Edge
            $P(u|v) = 0$
        else if $Q(u,v) = max(Q(u, a \in A))$ **then**
            $P(u|v) = (1 - \epsilon)$
        else
            $P(u|v) = \epsilon/(size(\mathcal{F} - 1))$
    Choose edge $(u \to v)$ with P
    Observe δ_{uv}
    $\delta_{it} \mathrel{+}= \delta_{uv}$
    $D_v = D_u - \delta_{uv}$
    $R =$ Environment Reward Function(v, δ_{it})
    $Q(u,v) =$ Value iteration from Equation
    if $v = t$ **then**
        DONE

Reward Function

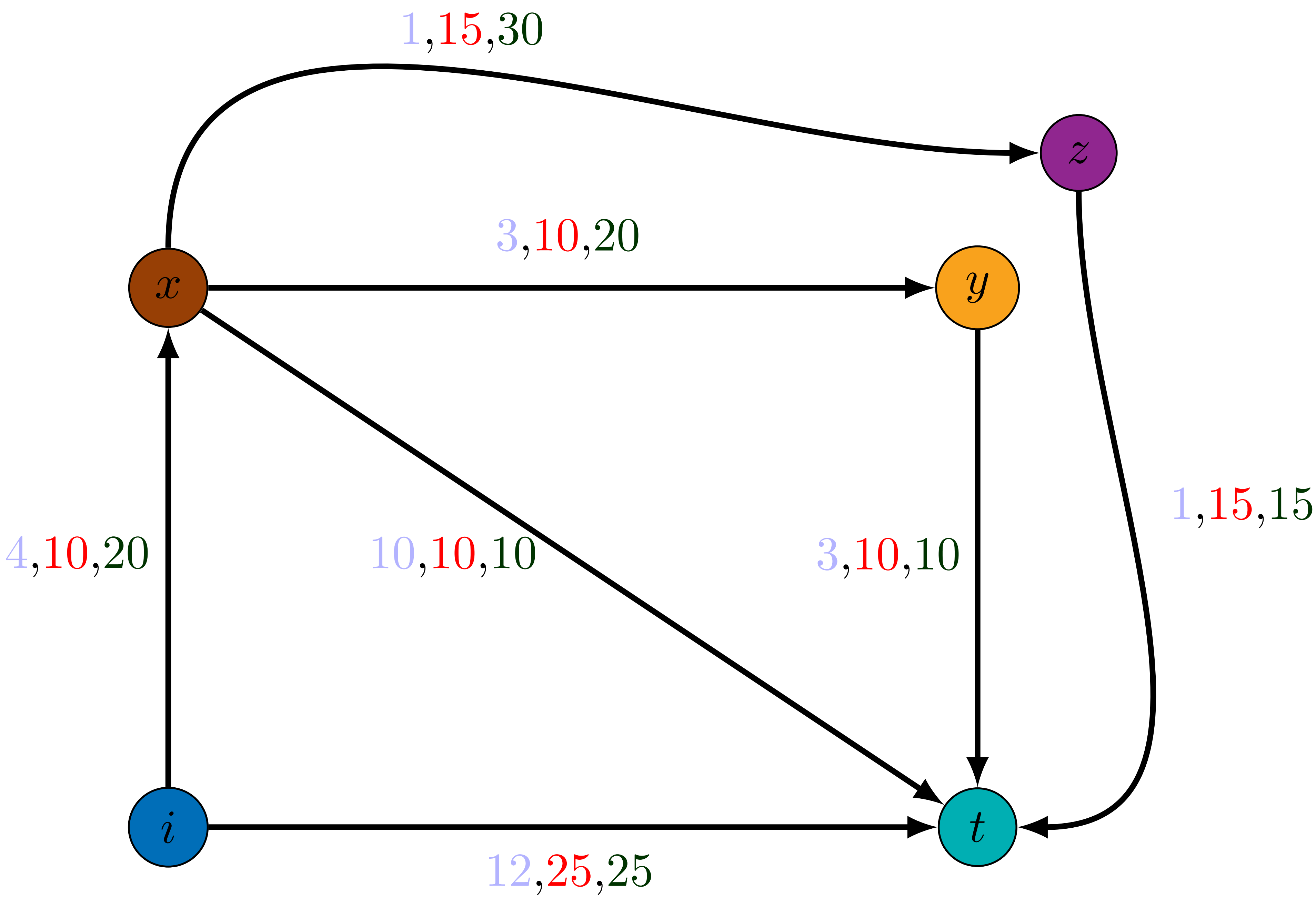
- Reward assigned by the environment
- At the end of every episode/packet transmission
- Propagates to other nodes through TD(0)

Algorithm 1 Environment Reward Function(v, δ_{it})

1: Assigns the reward at the end of transmission
2: **if** $v = t$ **then**
3: $\quad R = D_F - \delta_{it}$
4: **else**
5: $\quad R = 0$

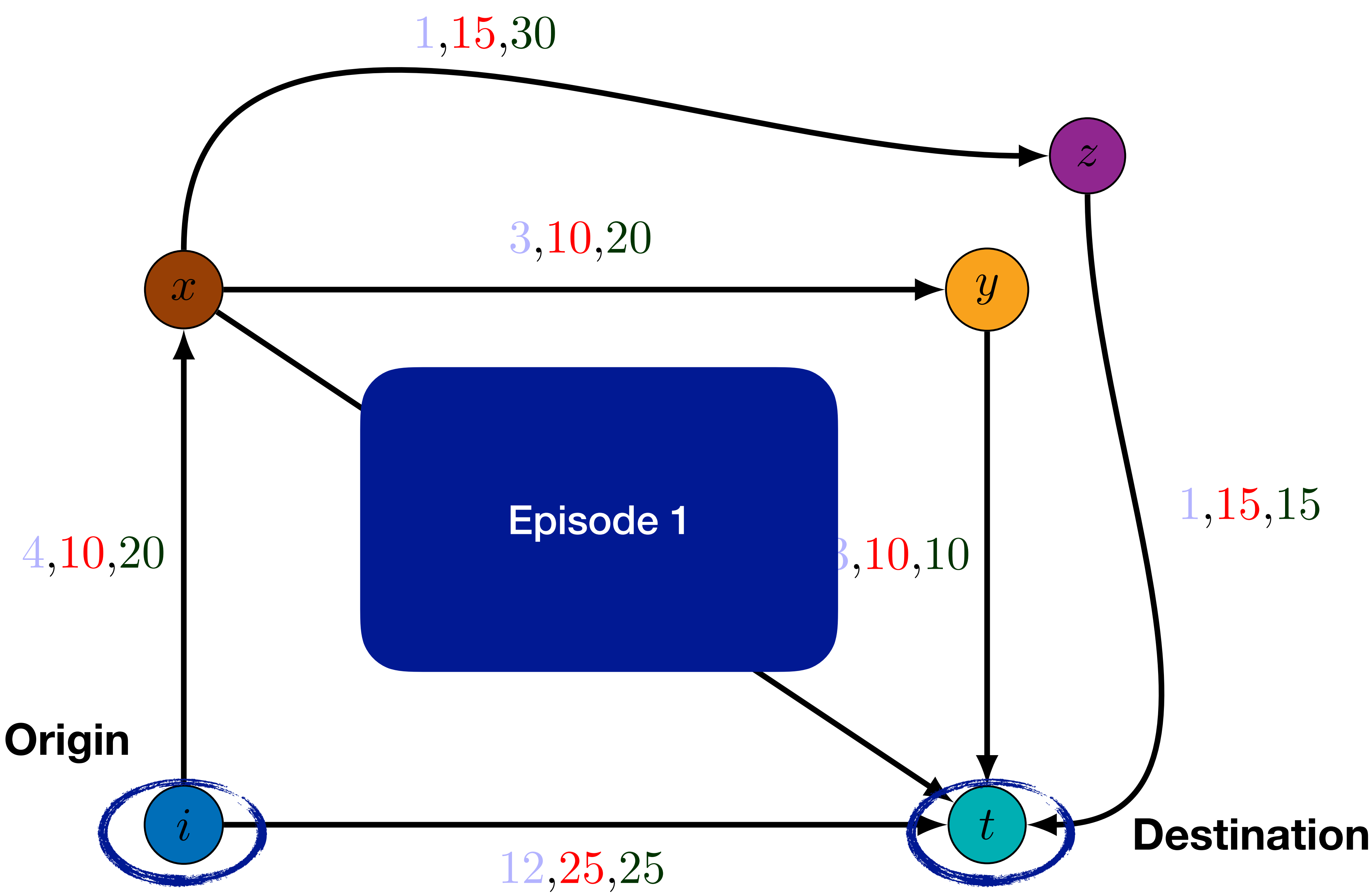
An Example. $D_F = 25$

Episode	Path	Transmission Time



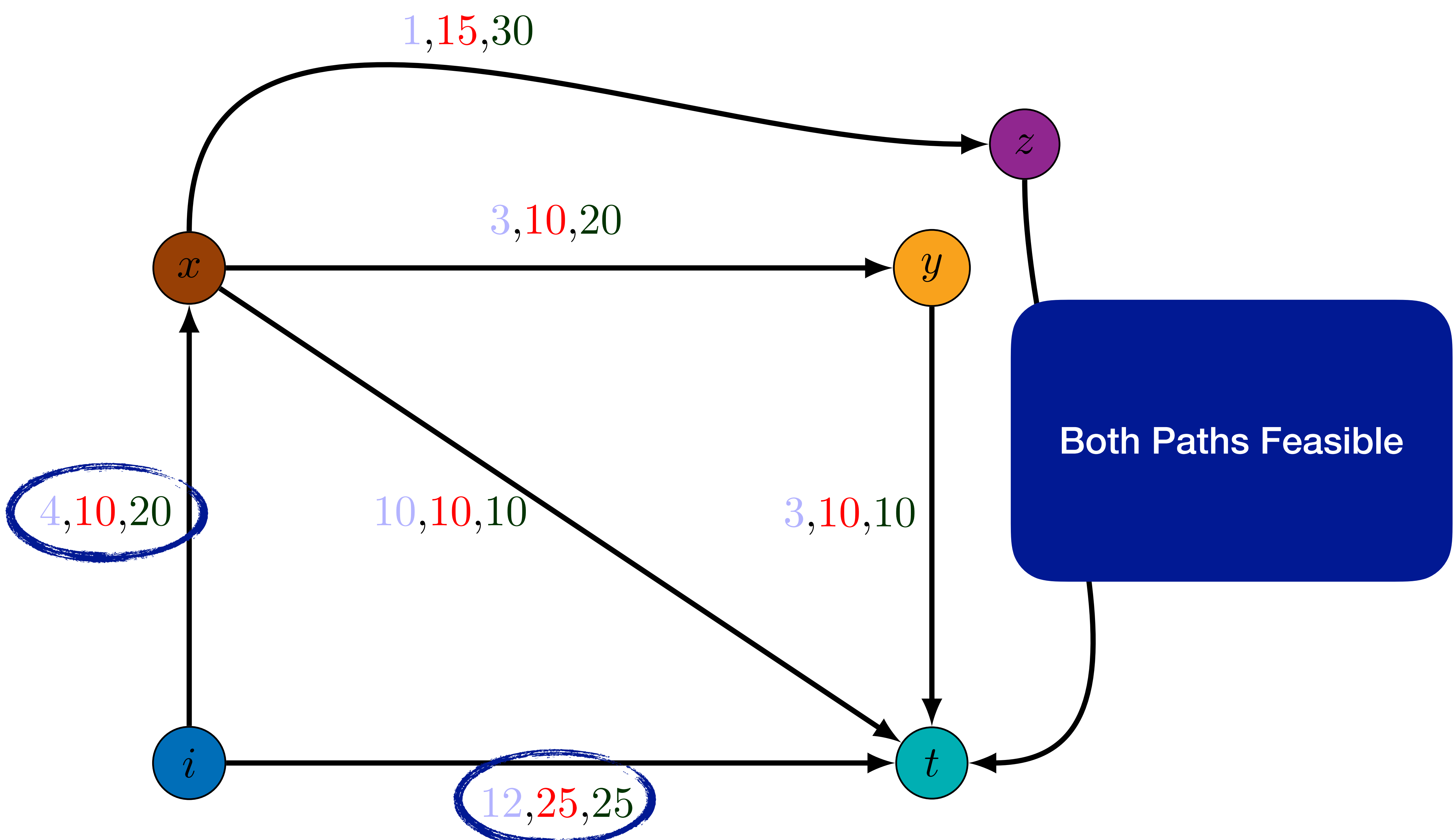
An Example. $D_F = 25$

Episode	Path	Transmission Time



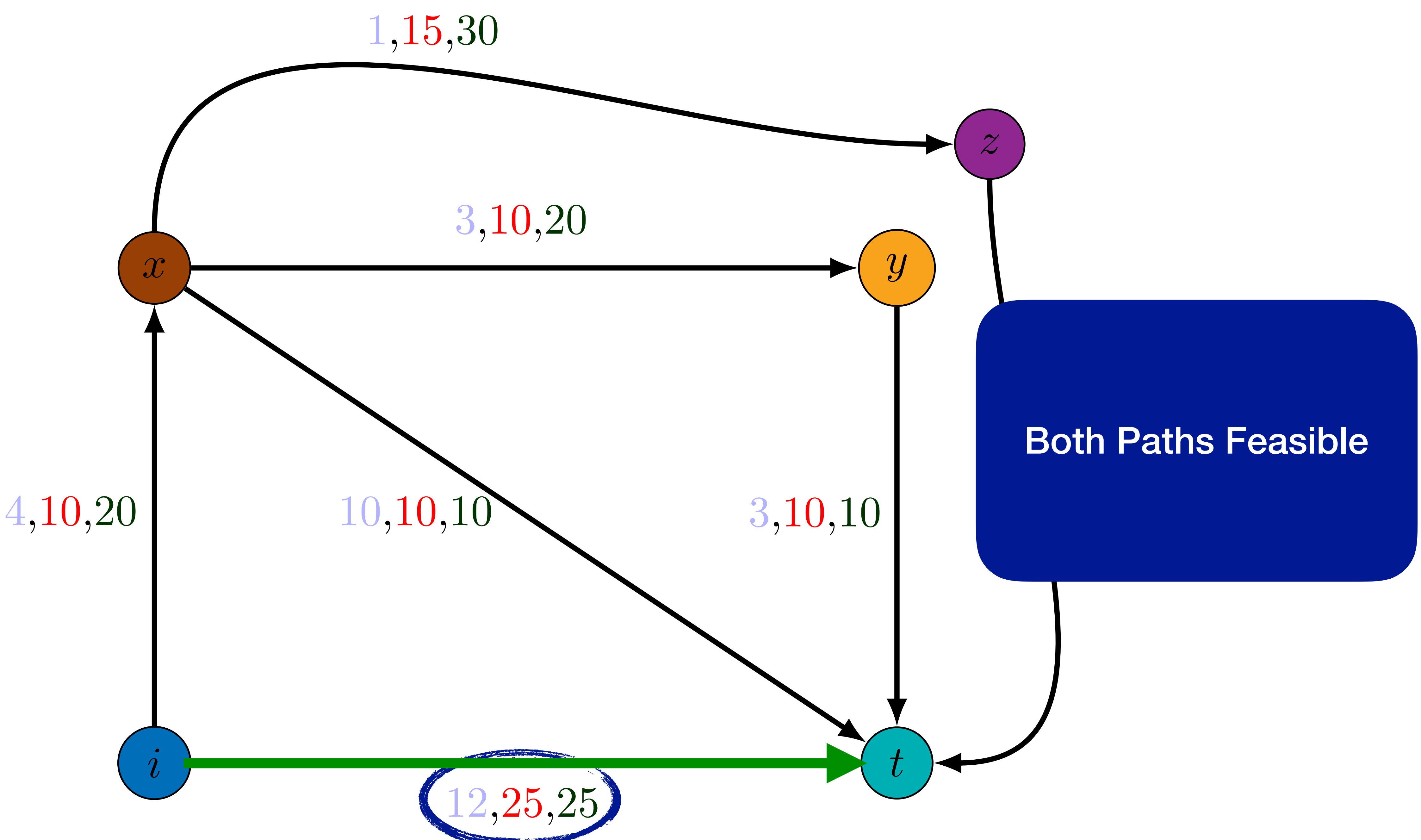
An Example. $D_F = 25$

Episode	Path	Transmission Time



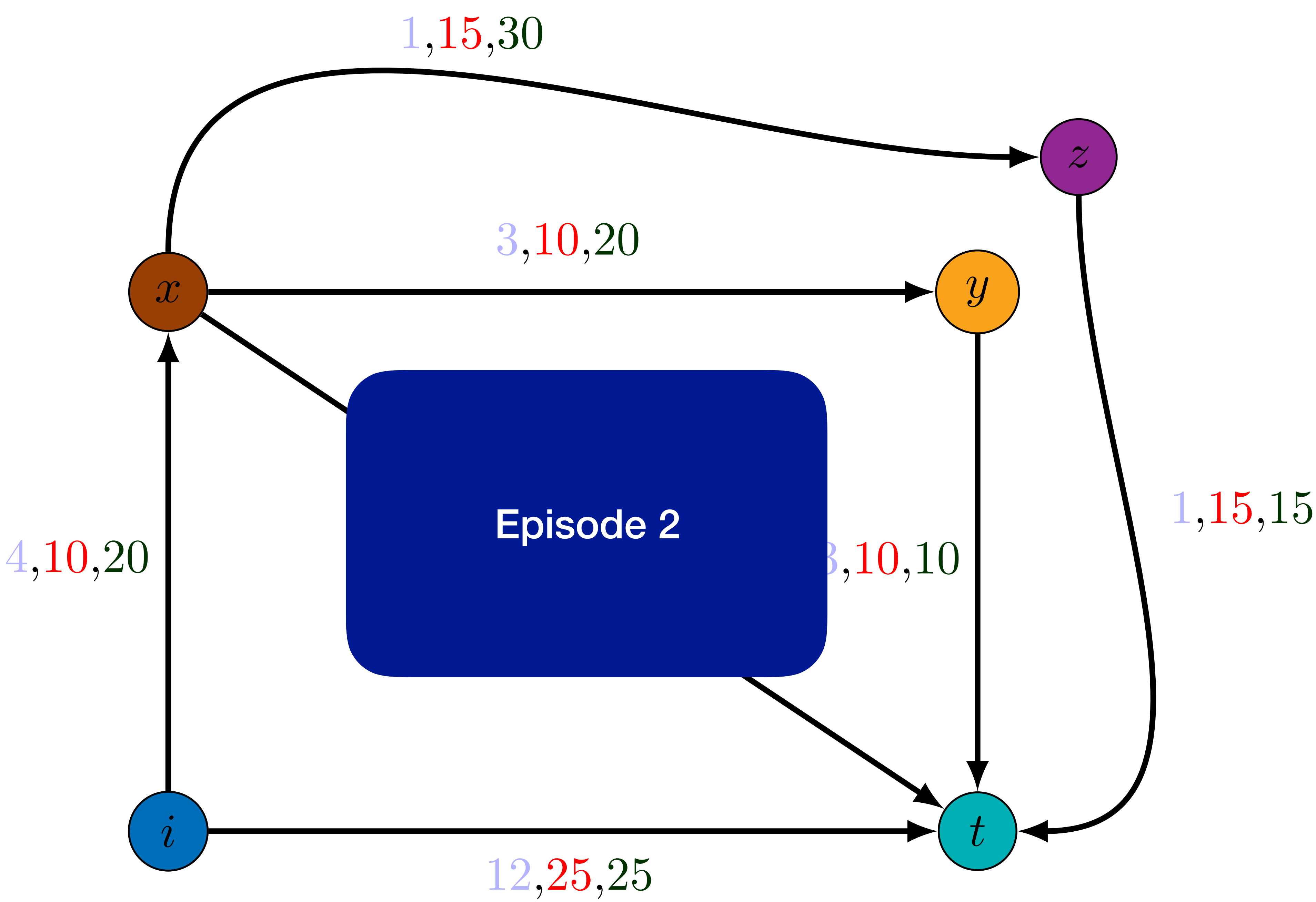
An Example. $D_F = 25$

Episode	Path	Transmission Time
1	[i, t]	12



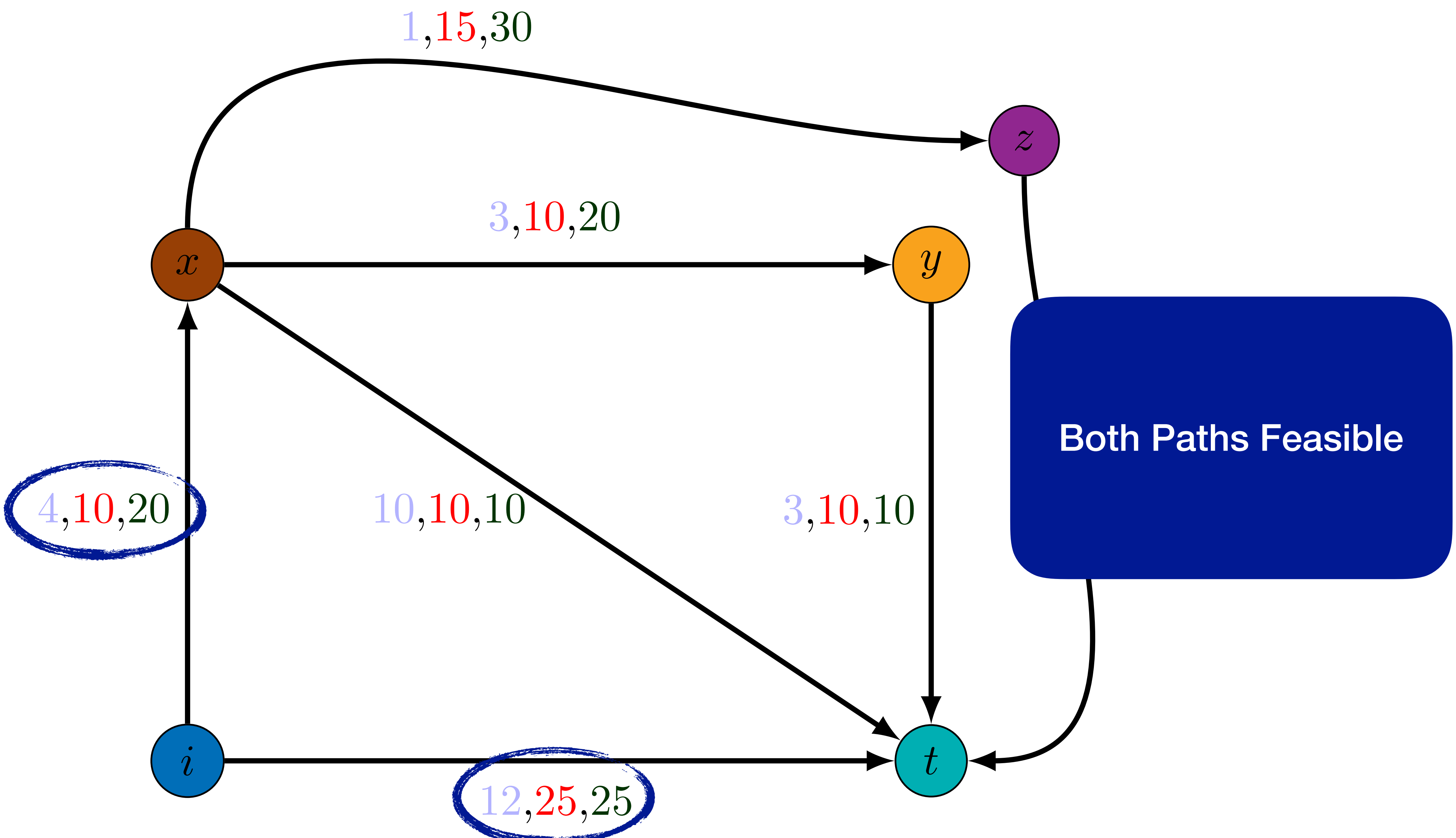
An Example. $D_F = 25$

Episode	Path	Transmission Time
1	[i, t]	12



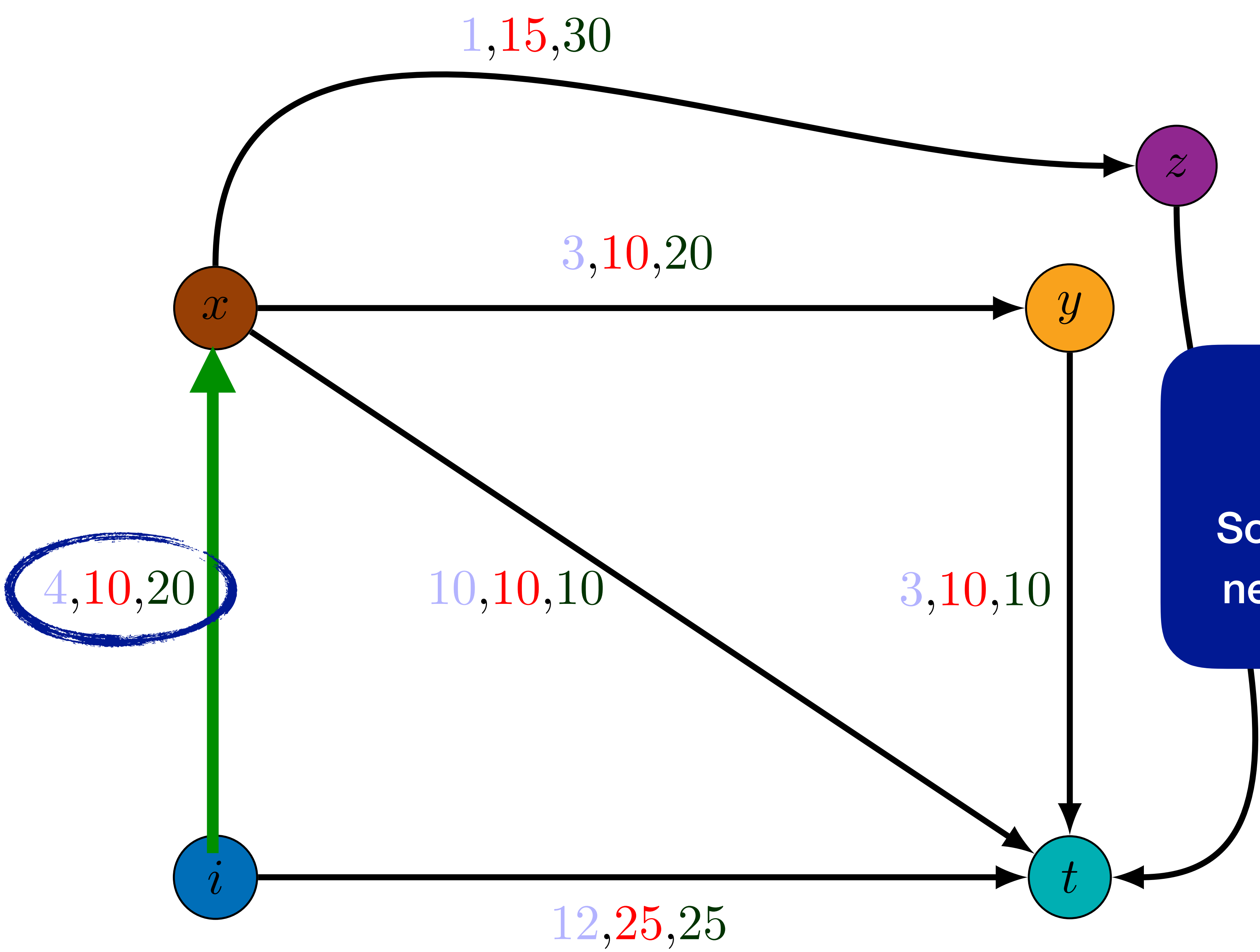
An Example. $D_F = 25$

Episode	Path	Transmission Time
1	[i, t]	12



An Example. $D_F = 25$

Episode	Path	Transmission Time
1	[i, t]	12
2	[i, x, -]	4

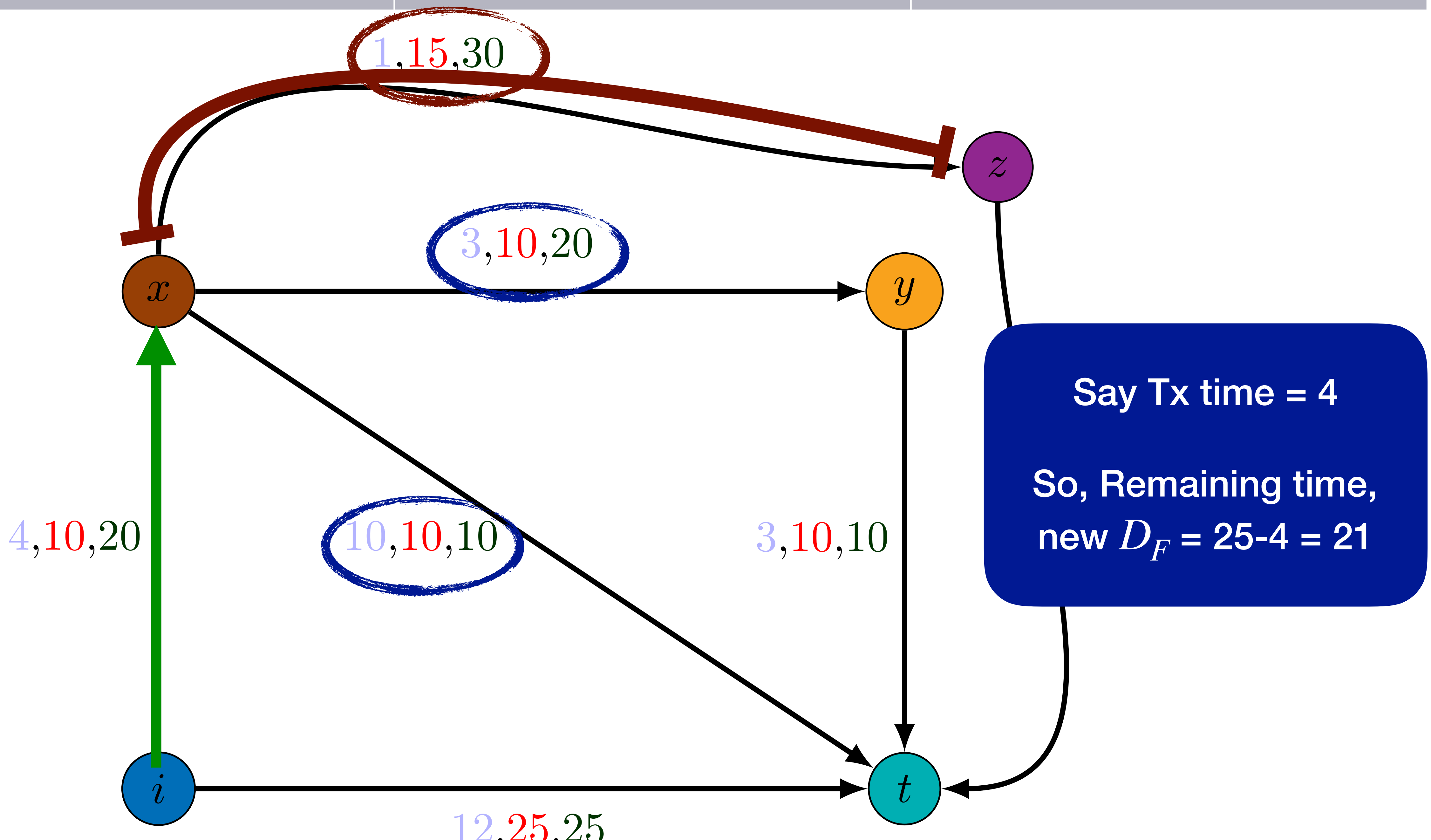


Say Tx time = 4

So, Remaining time, new D_F = 25-4 = 21

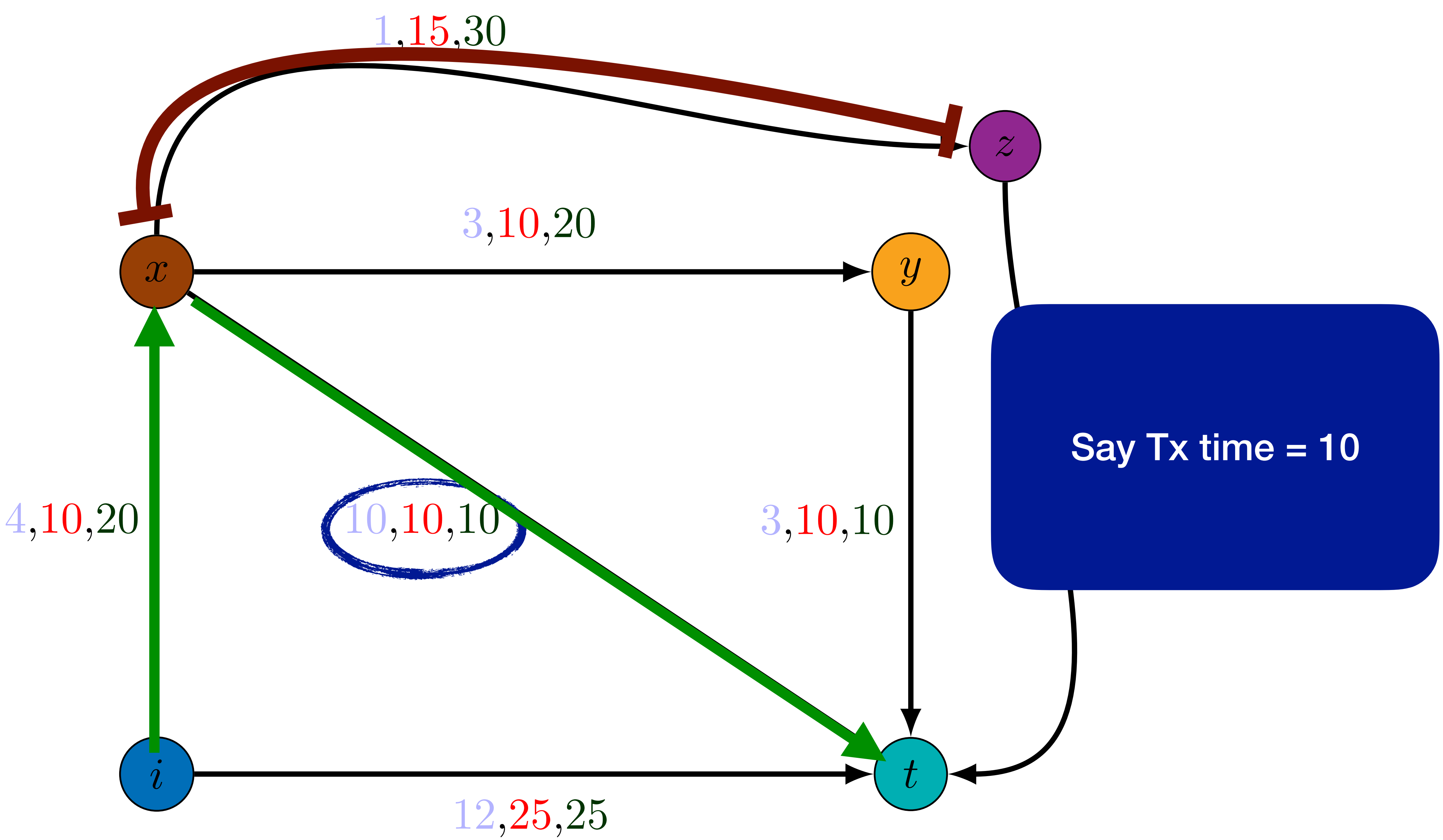
An Example. D_F = 25

Episode	Path	Transmission Time
1	[i, t]	12
2	[i, x, -]	4



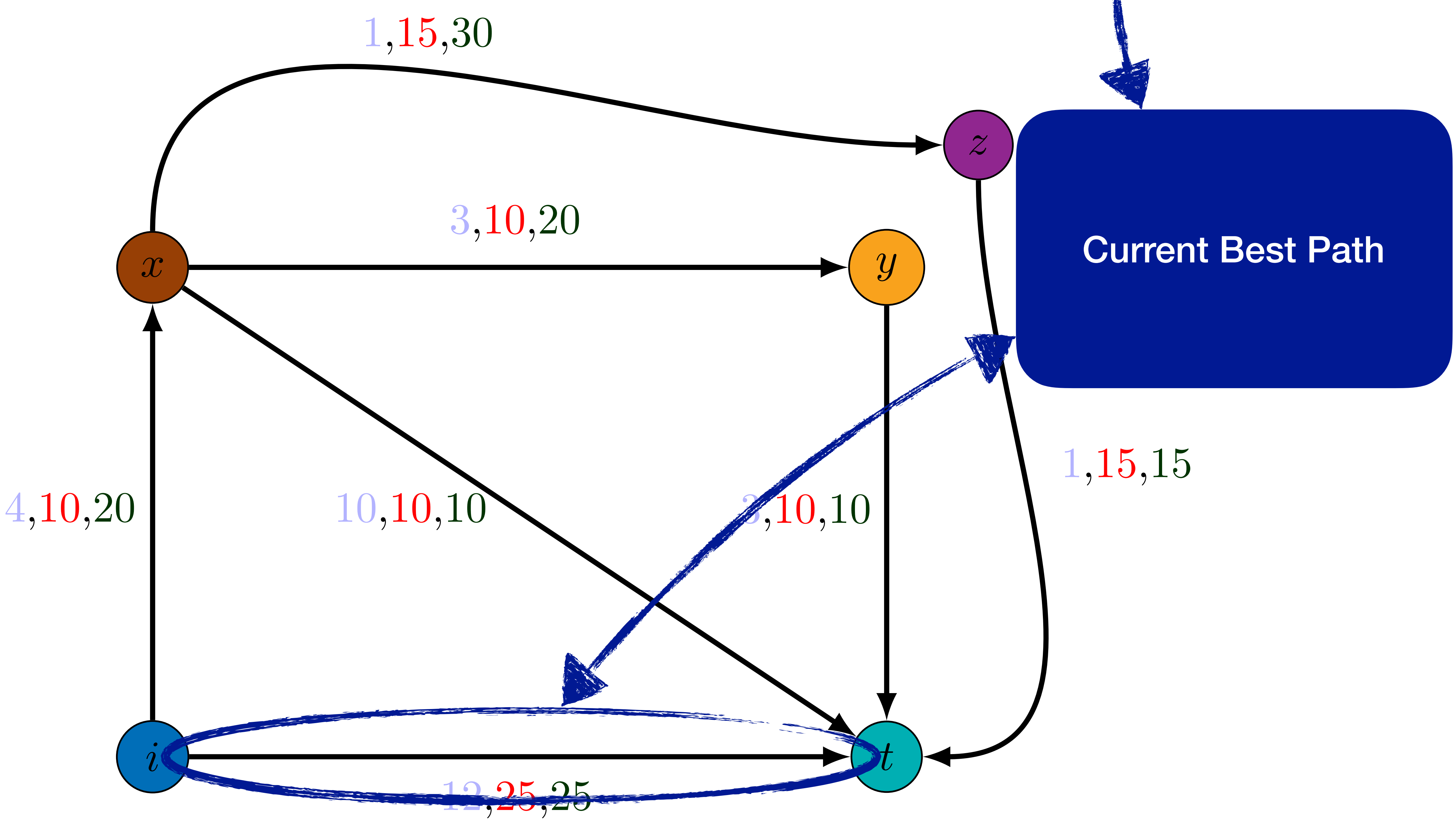
An Example. $D_F = 25$

Episode	Path	Transmission Time
1	[i, t]	12
2	[i, x, t]	14



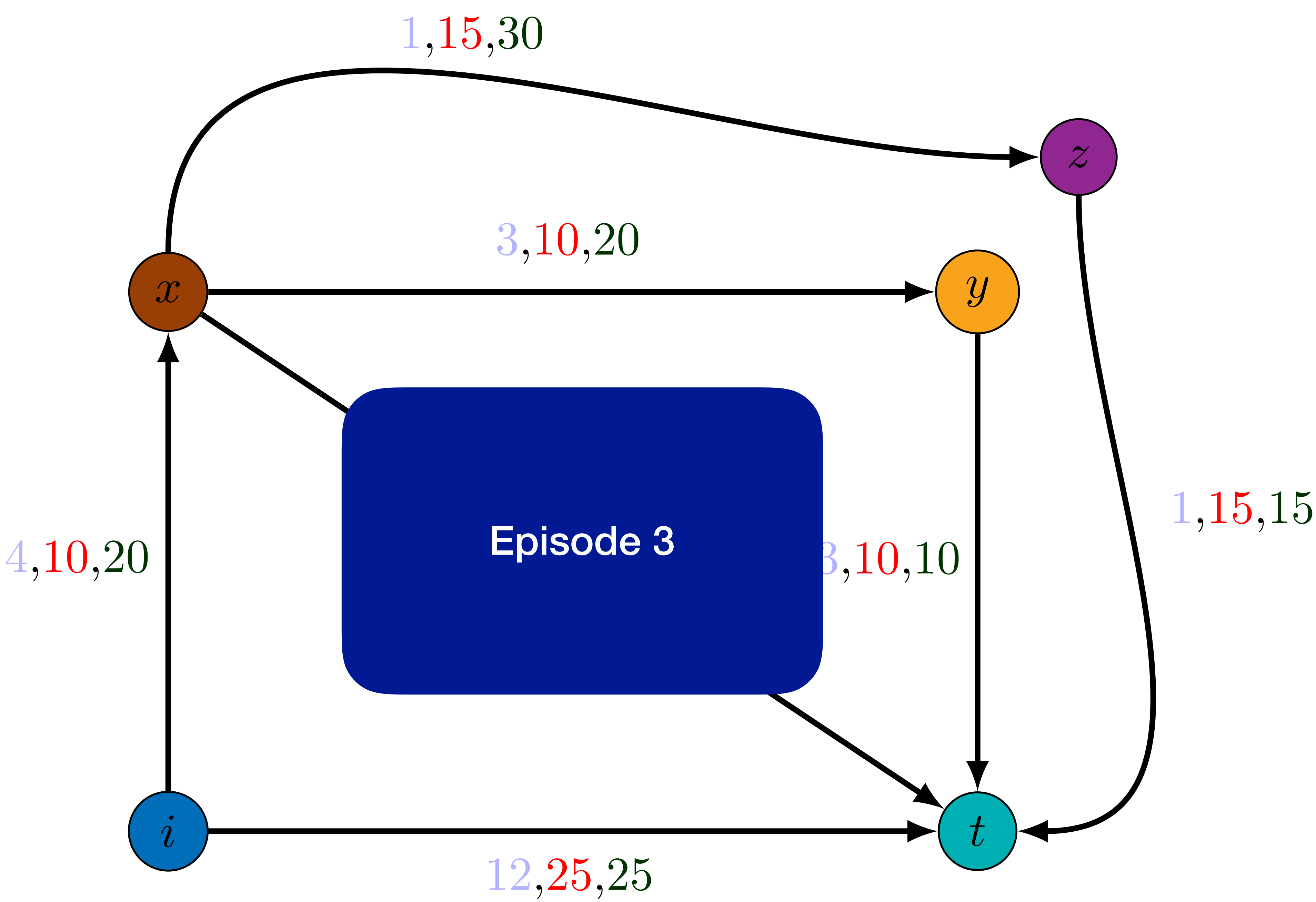
An Example. $D_F = 25$

Episode	Path	Transmission Time
1	[i, t]	12
2	[i, x, t]	14



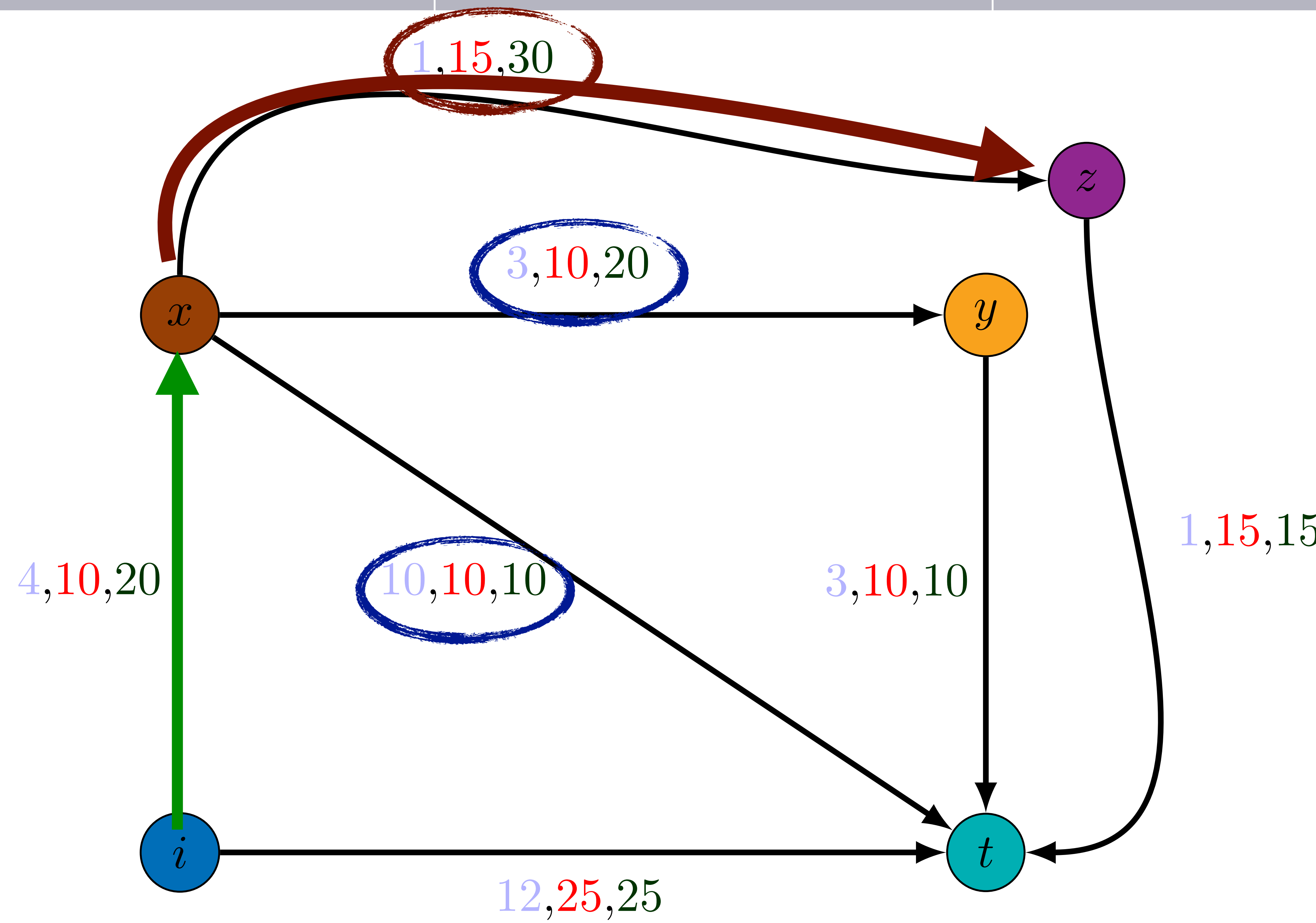
An Example. $D_F = 25$

Episode	Path	Transmission Time
1	[i, t]	12
2	[i, x, t]	14



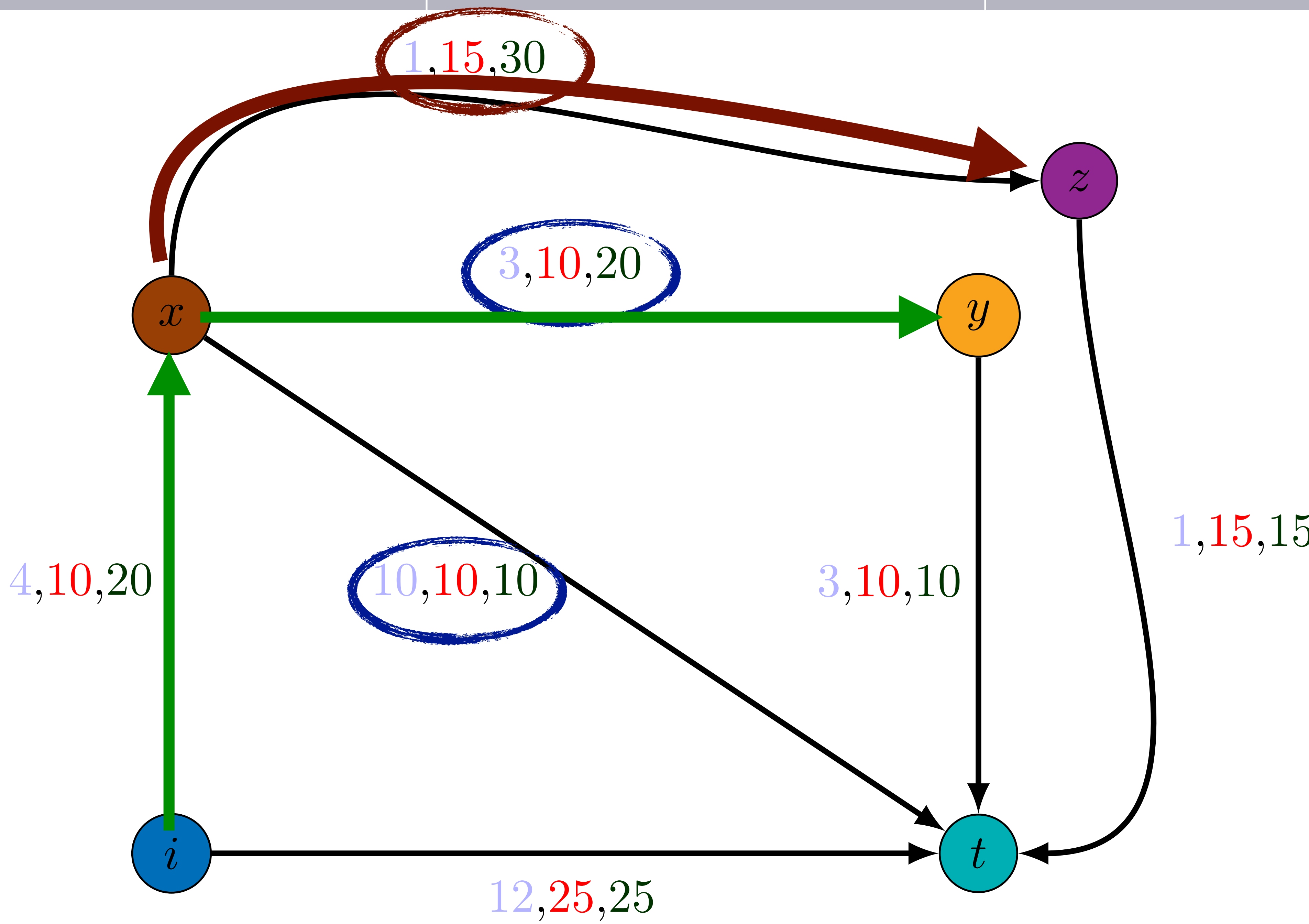
An Example. $D_F = 25$

Episode	Path	Transmission Time
1	[i, t]	12
2	[i, x, t]	14
3	[i, x, -]	4



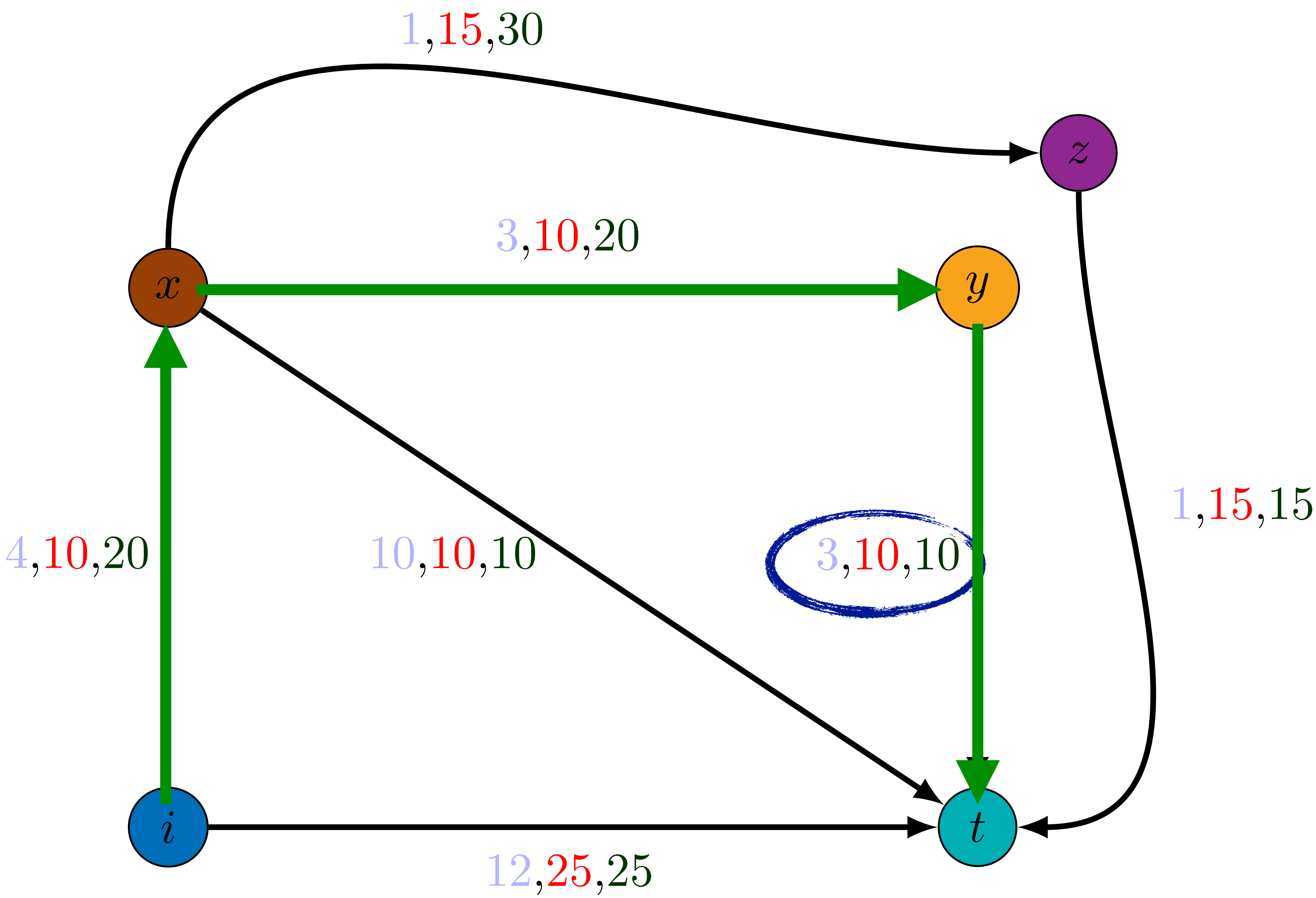
An Example. $D_F = 25$

Episode	Path	Transmission Time
1	[i, t]	12
2	[i, x, t]	14
3	[i, x, y]	7



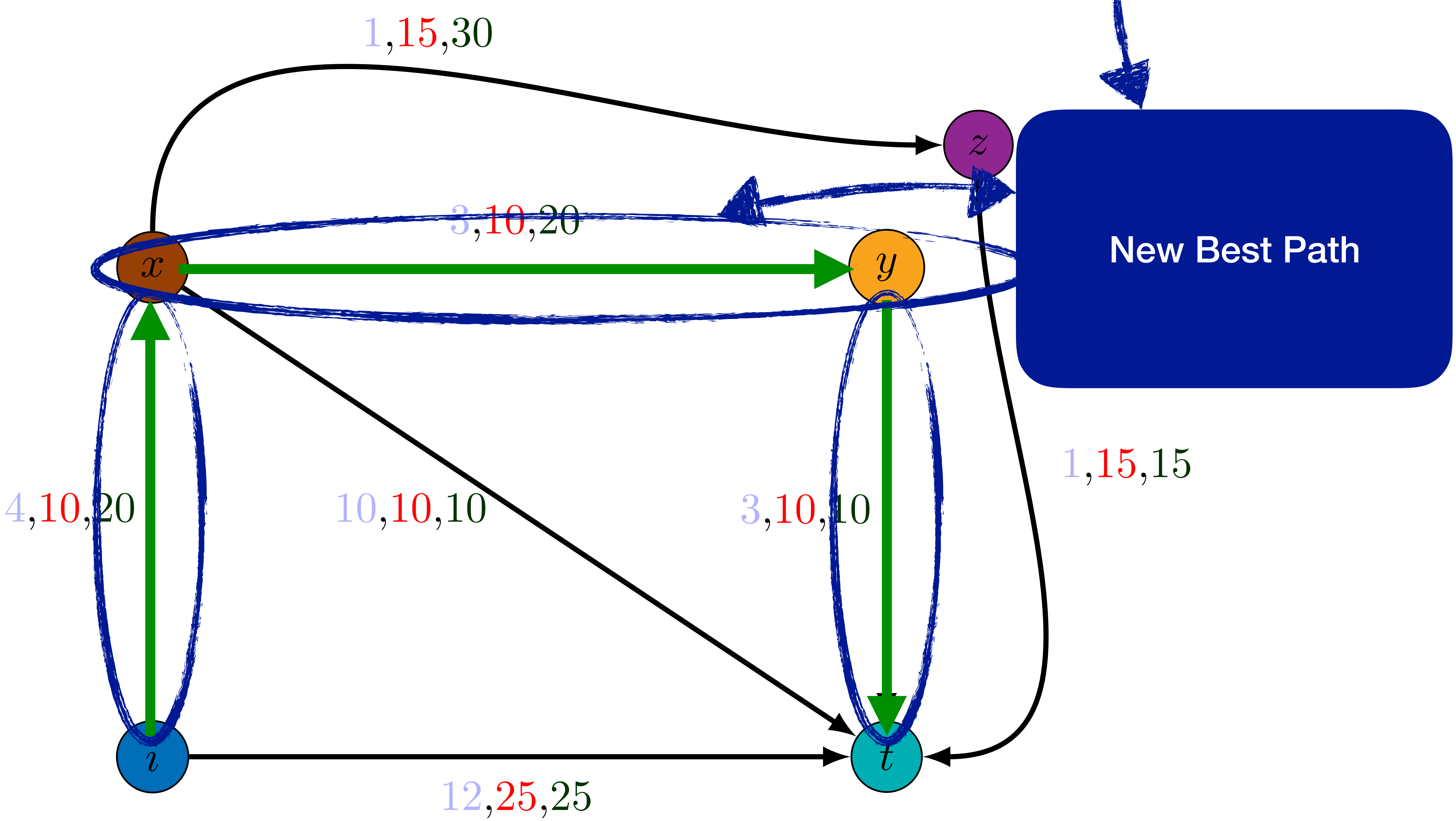
An Example. $D_F = 25$

Episode	Path	Transmission Time
1	[i, t]	12
2	[i, x, t]	14
3	[i, x, y, t]	10



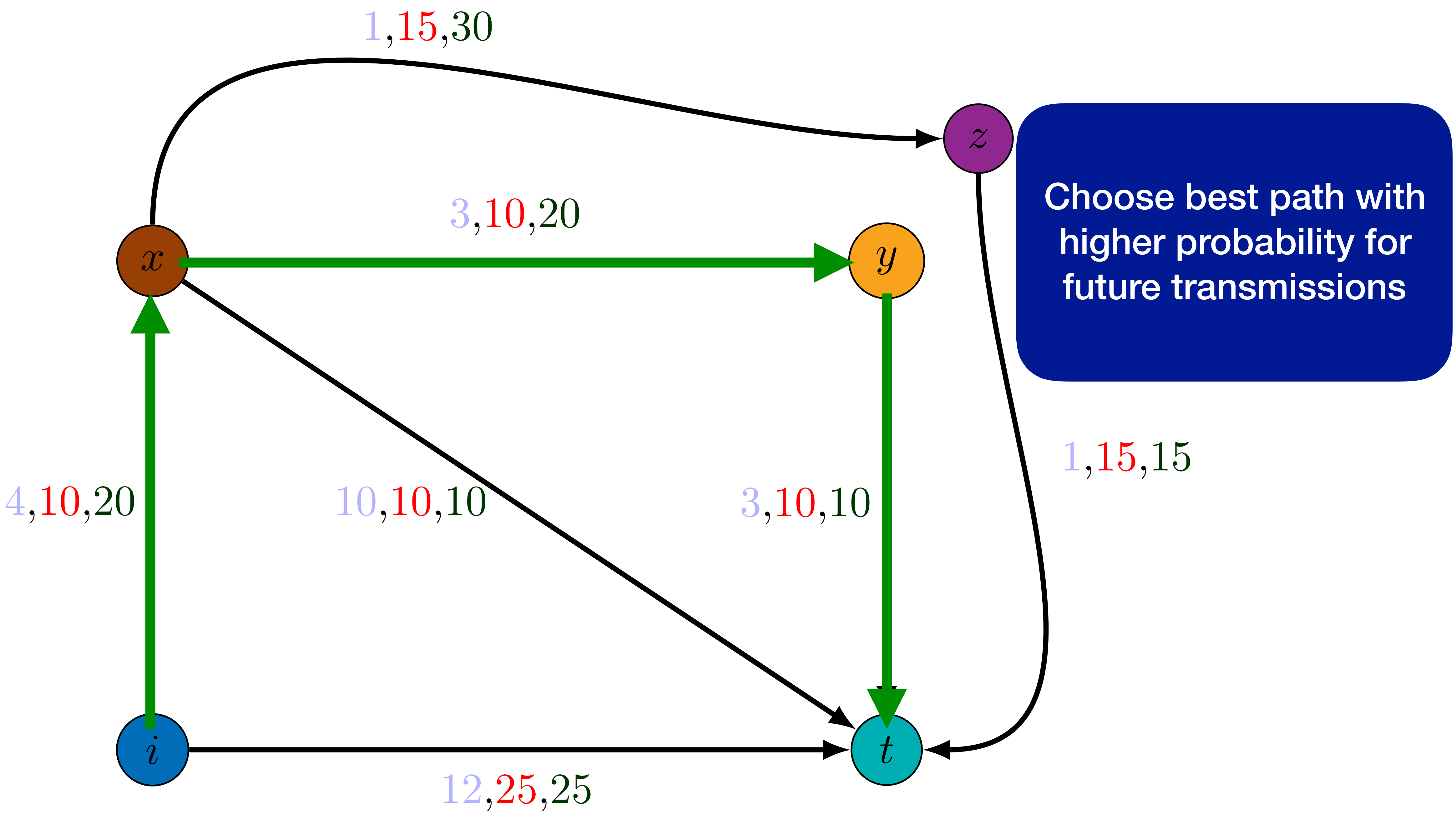
An Example. $D_F = 25$

Episode	Path	Transmission Time
1	[i, t]	12
2	[i, x, t]	14
3	[i, x, y, t]	10



An Example. $D_F = 25$

Episode	Path	Transmission Time
1	[i, t]	12
2	[i, x, t]	14
3	[i, x, y, t]	10
4	[i, x, y, t]	10



Evaluation

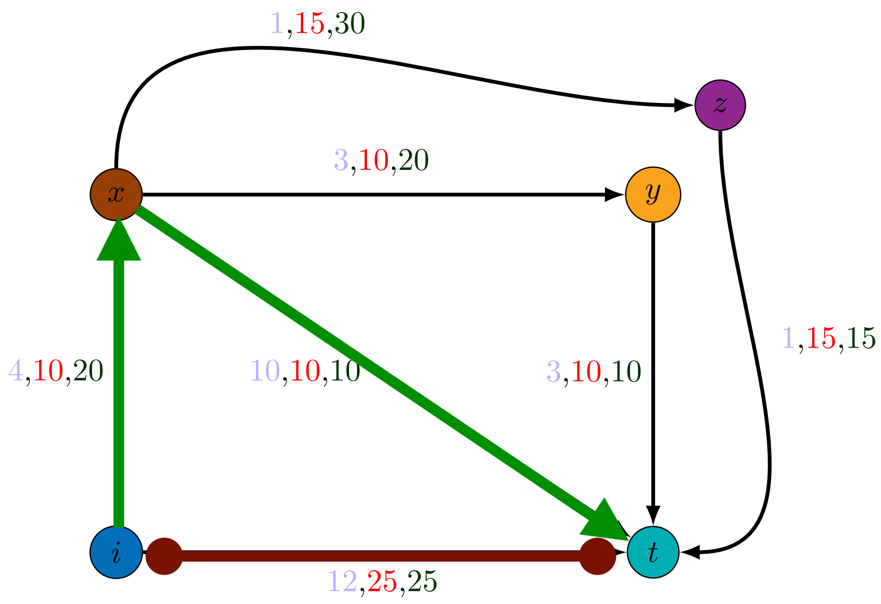
- Implemented using NetworkX[1]
- Python graph generator package
- Compare our algorithm to Rapid Routing[2] and classical RL

[1] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, "Exploring network structure, dynamics, and function using NetworkX"

[2] S. Baruah, "Rapid routing with guaranteed delay bounds," in 2018 IEEE Real-Time Systems Symposium (RTSS) , December 2018

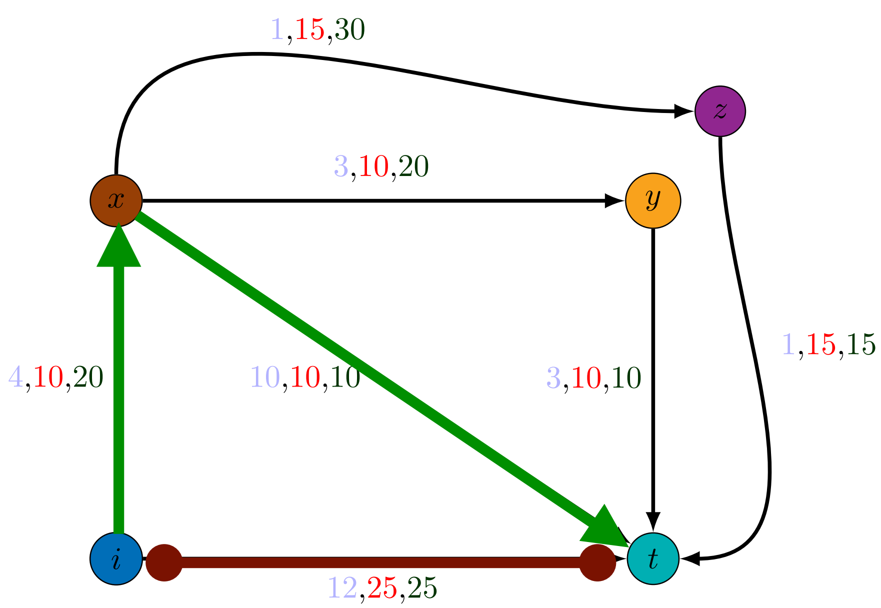
Deadline	Path	No Variance Tx Time
20	[i, x, t]	14

D_F = **20**

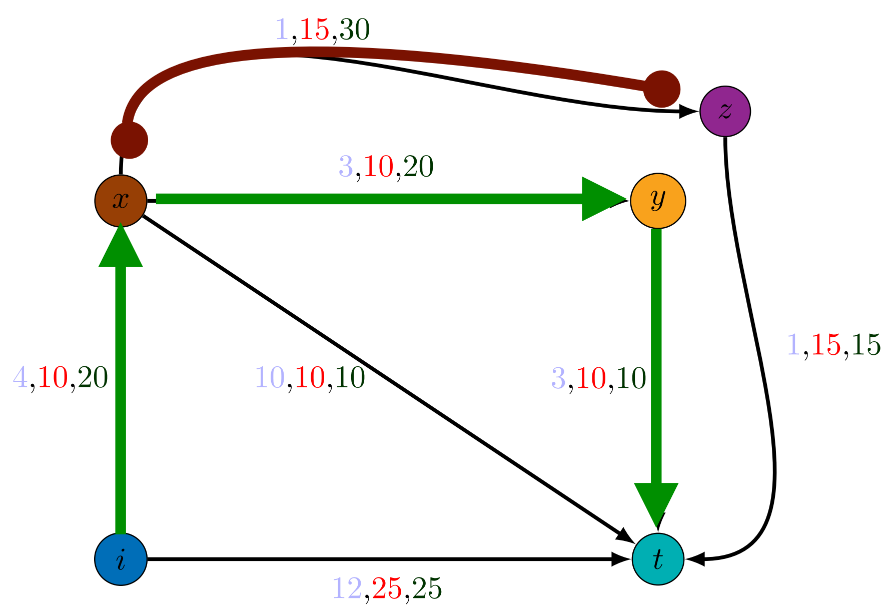


Deadline	Path	No Variance Tx Time
20	[i, x, t]	14
25	[i, x, y, t]	10

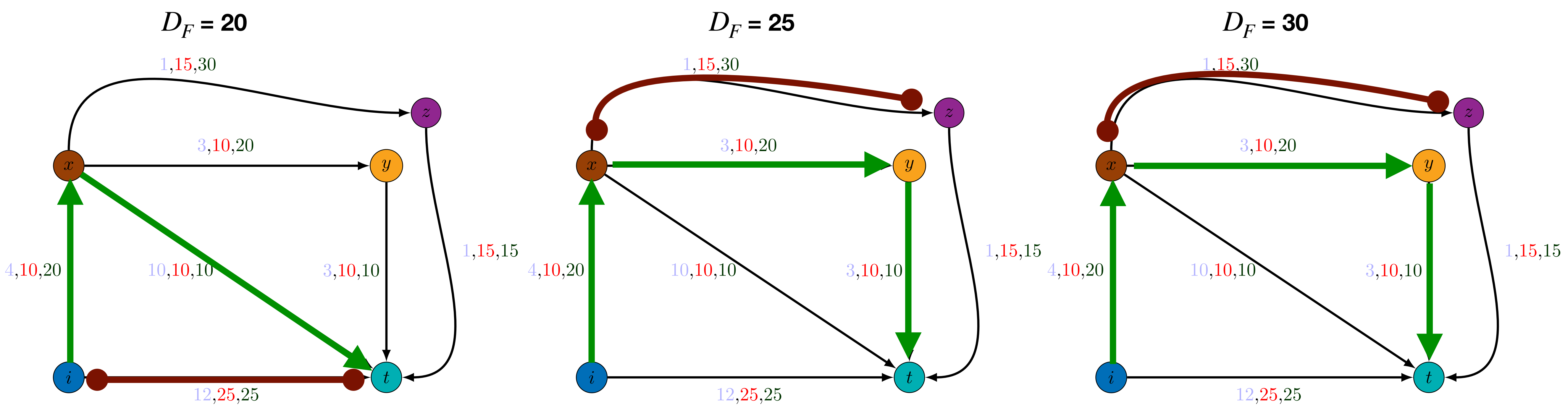
D_F = **20**



D_F = **25**

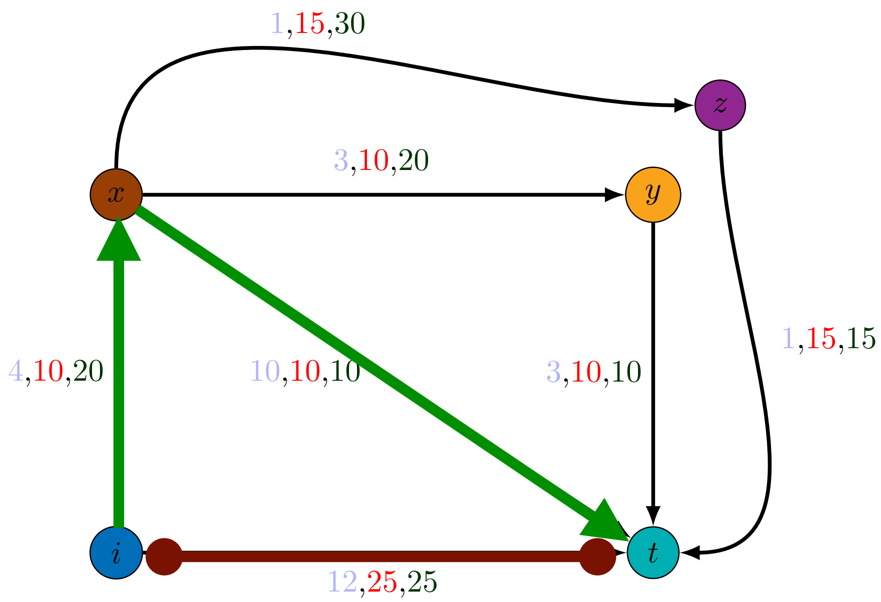


Deadline	Path	No Variance Tx Time
20	[i, x, t]	14
25	[i, x, y, t]	10
30	[i, x, y, t]	10

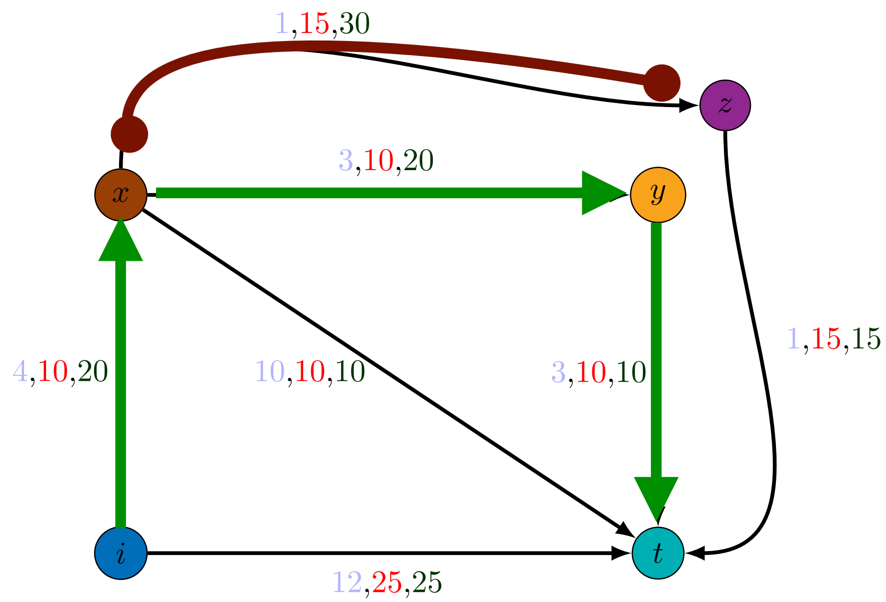


Deadline	Path	No Variance Tx Time
20	[i, x, t]	14
25	[i, x, y, t]	10
30	[i, x, y, t]	10
35	[i, x, z, t]	6

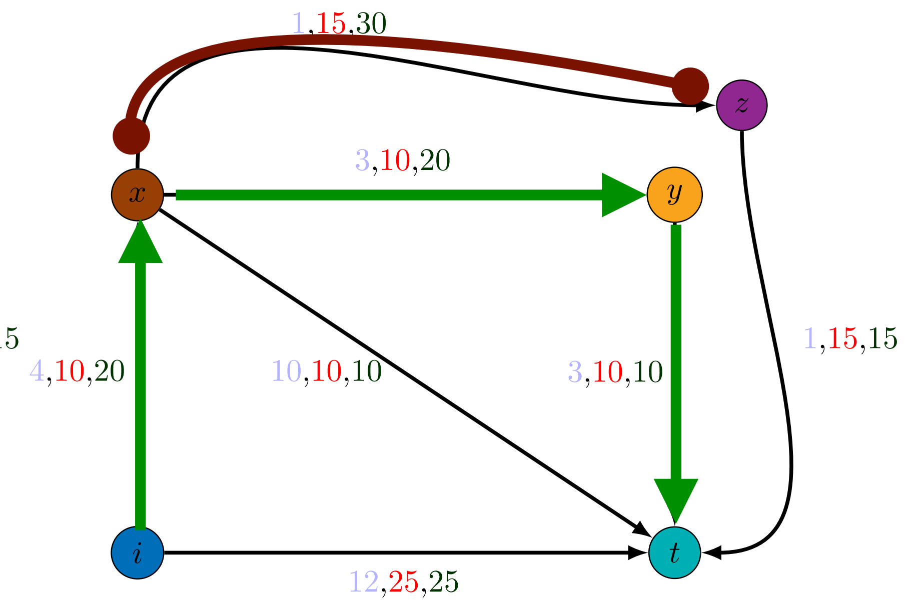
D_F = **20**



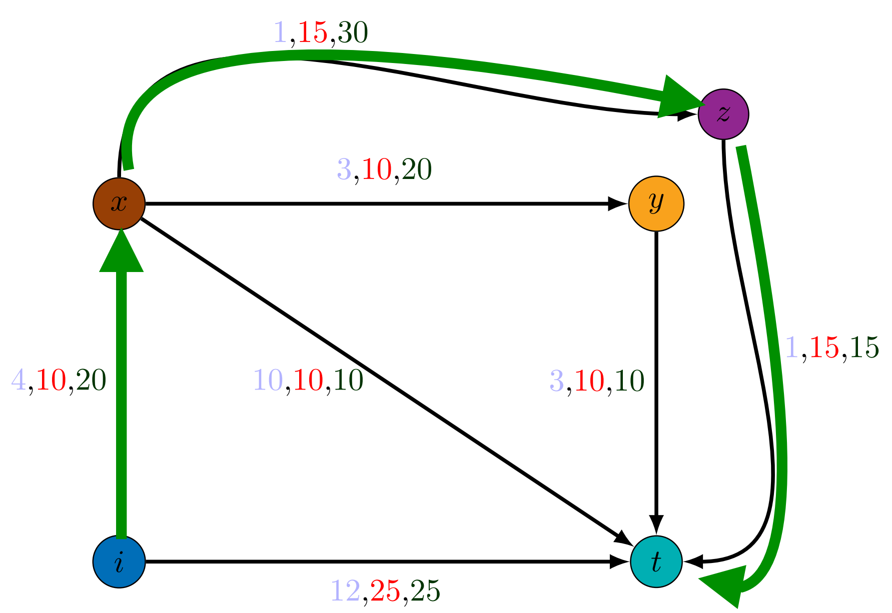
D_F = **25**



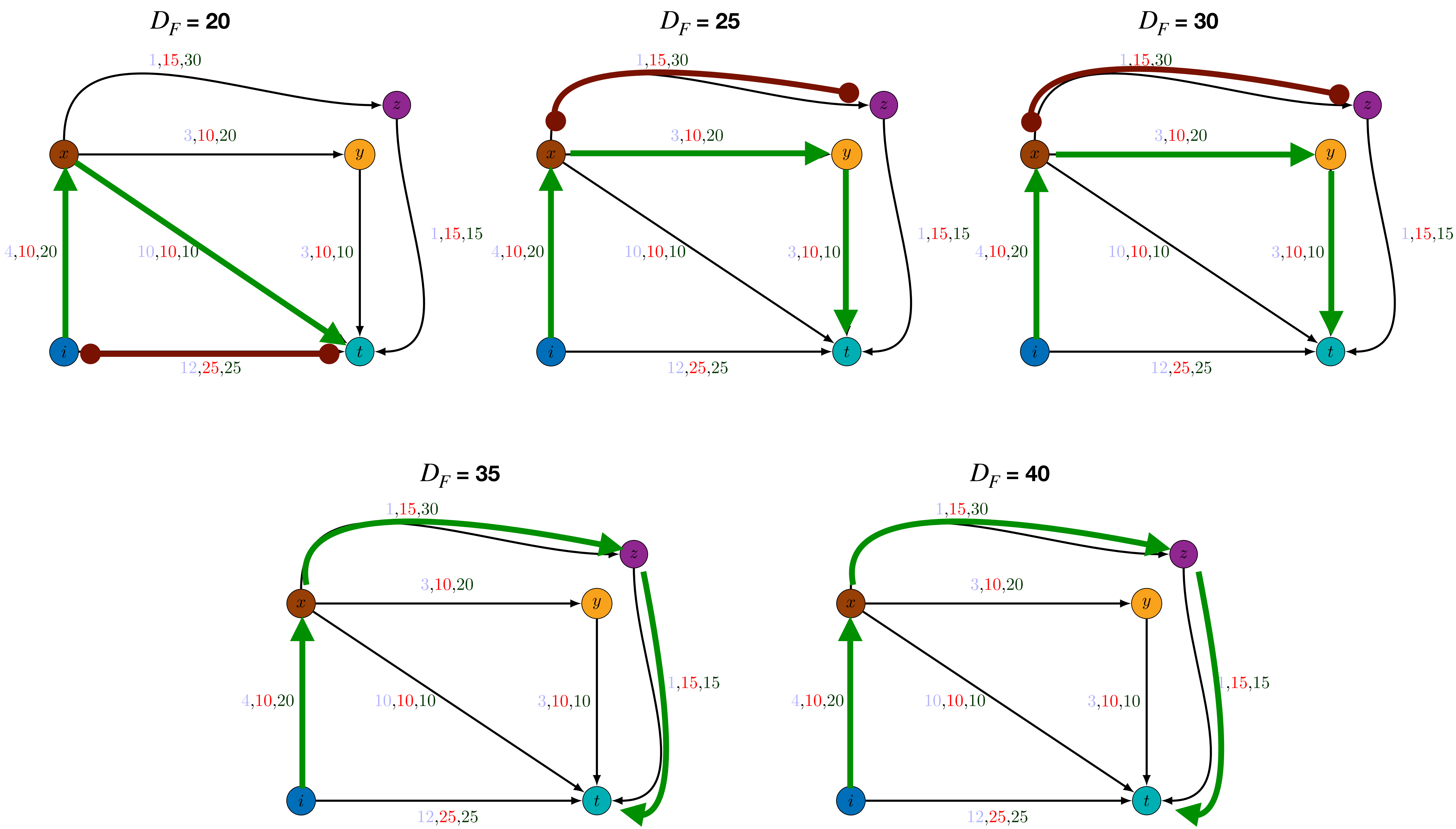
D_F = **30**



D_F = **35**



Deadline	Path	No Variance Tx Time
20	[i, x, t]	14
25	[i, x, y, t]	10
30	[i, x, y, t]	10
35	[i, x, z, t]	6
40	[i, x, z, t]	6



Experiment 1: No Variance. $\delta = c_{xy}^{T}$

- No Variance in the typical transmission times

Experiment 1: No Variance

- No Variance in the typical transmission times
- Paths taken by both Rapid Routing and Safe RL converge to the same

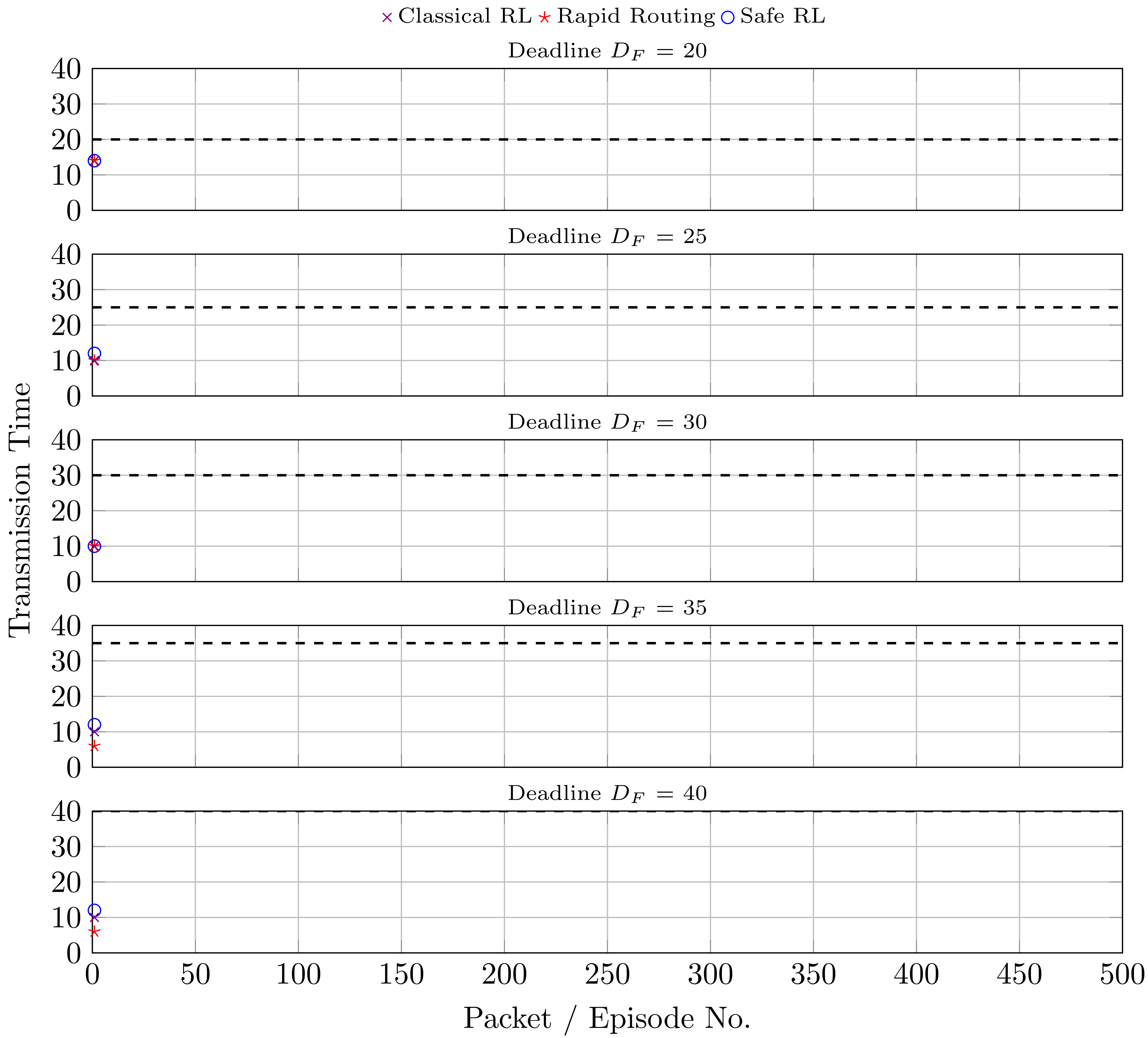
Experiment 1: No Variance

- No Variance in the typical transmission times
- Paths taken by both Rapid Routing and Safe RL converge to the same
- Average delays are slightly higher for Safe RL due to exploration

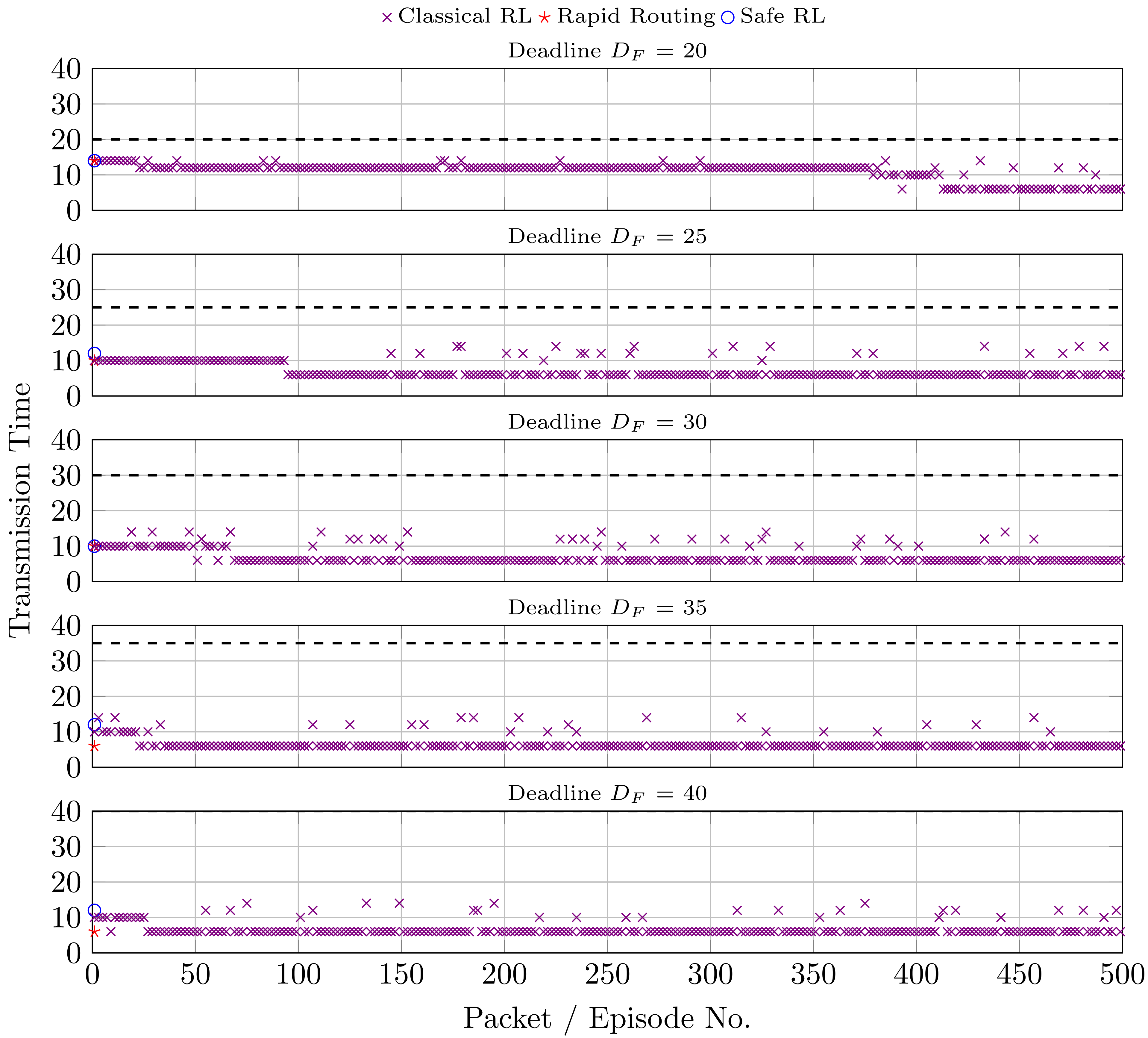
Table 1: Optimal Path for Different Deadlines

D_F	Optimal Path	Delays, Rapid Routing	Average Delays (1000 episodes)
15	Infeasible	-	-
20	{i,x,t}	14	14
25	{i,x,y,t}	10	10.24
30	{i,x,y,t}	10	10.22
35	{i,x,z,t}	6	6.64
40	{i,x,z,t}	6	6.55

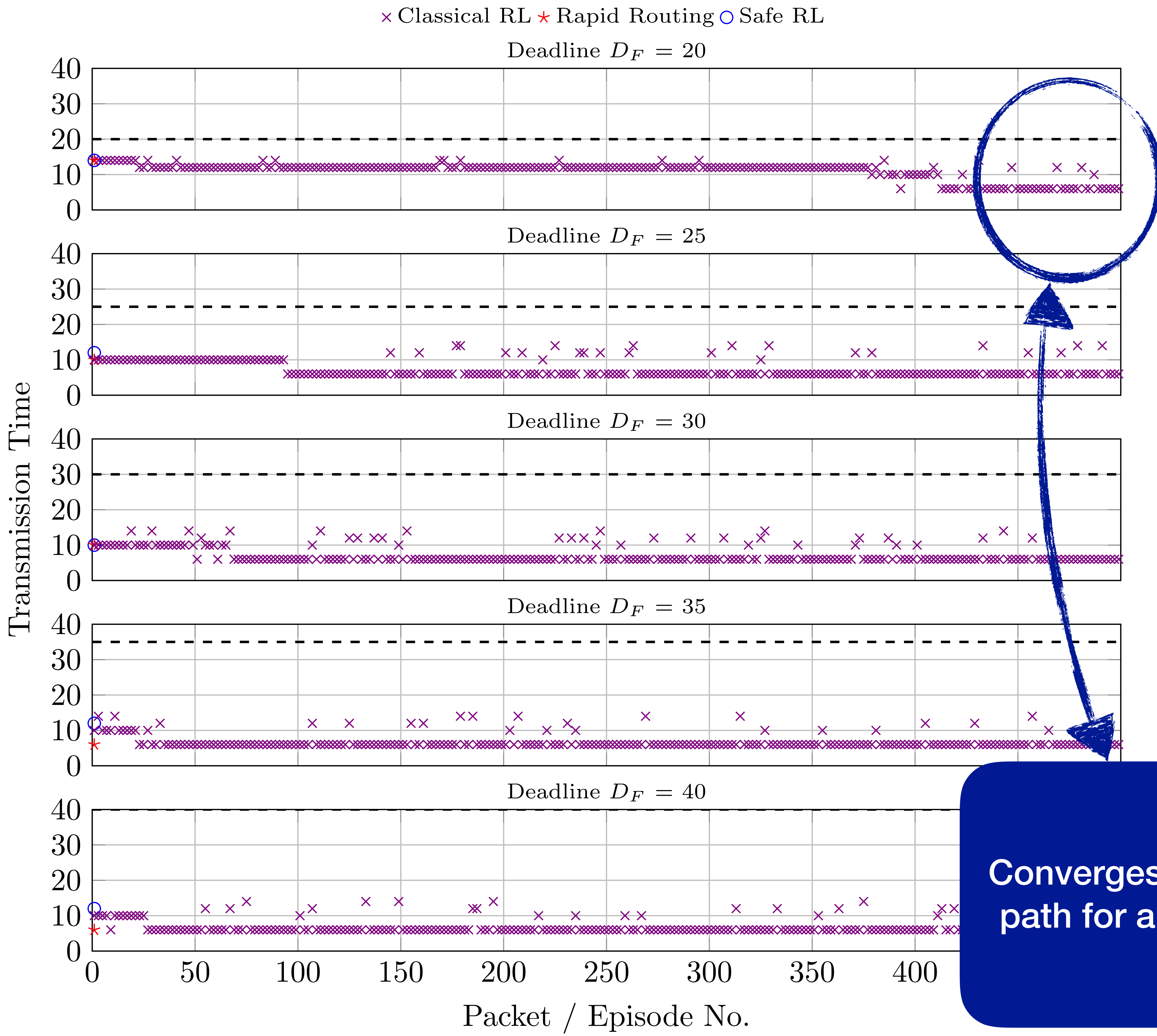
Experiment 1: No Variance. $\delta = c_{xy}^T$



Experiment 1: No Variance. $\delta = c_{xy}^T$

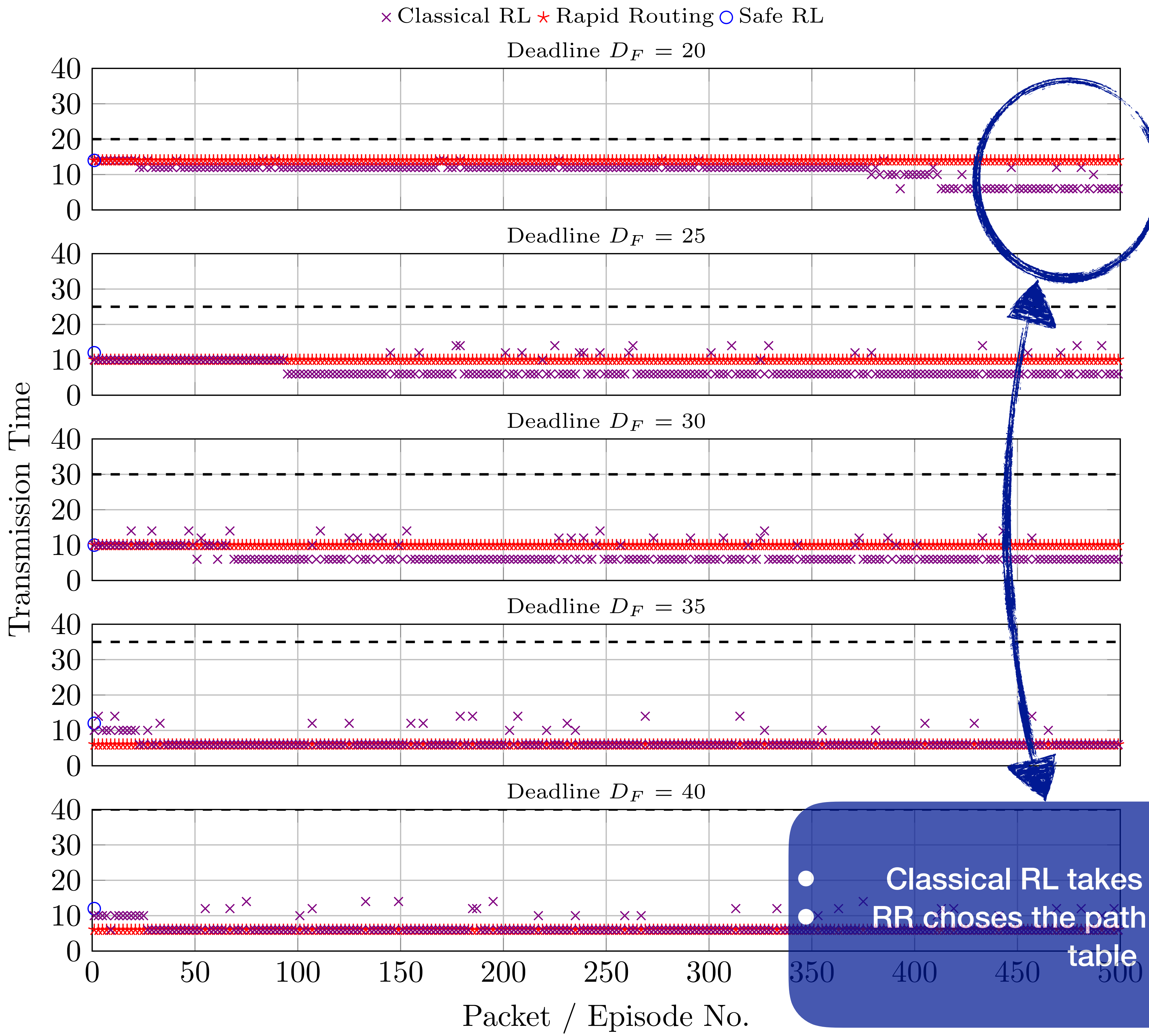


Experiment 1: No Variance. $\delta = c_{xy}^{T}$



Converges to the best path for all deadlines

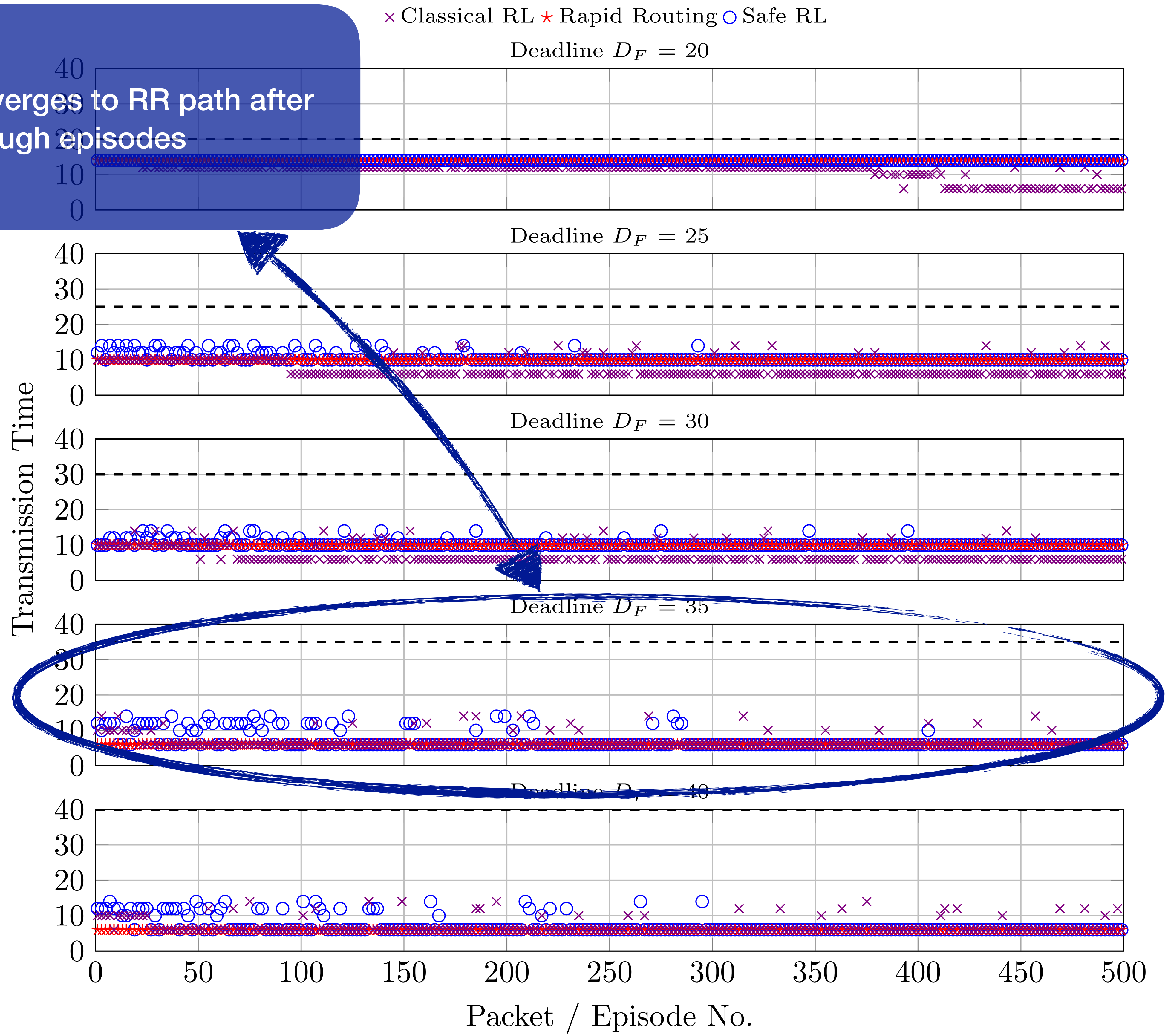
Experiment 1: No Variance. $\delta = c_{xy}^T$



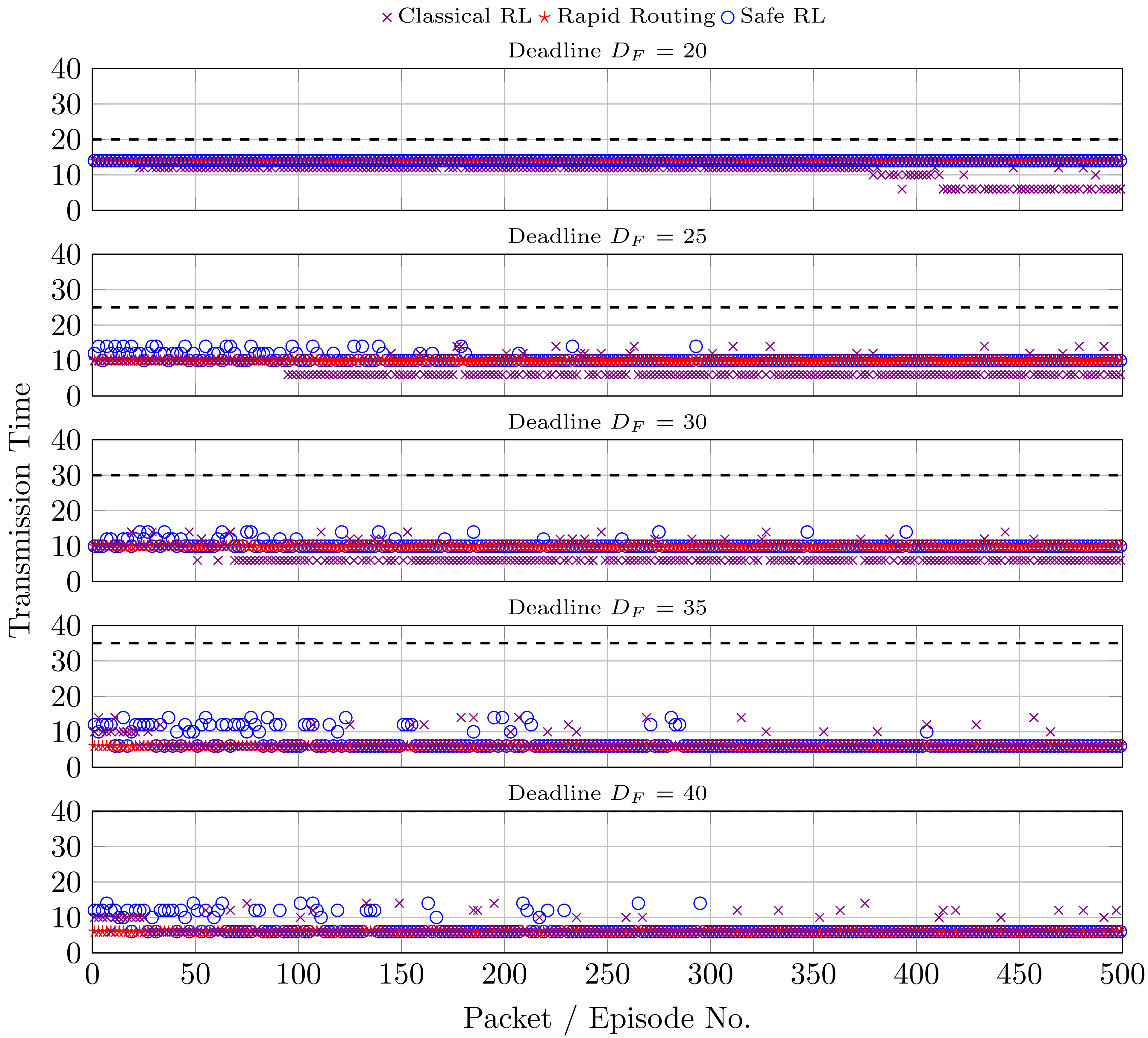
- Classical RL takes unsafe path
- RR choses the path according to table

Experiment 1: No Variance. $\delta = c_{xy}^T$

Safe RL converges to RR path after enough episodes



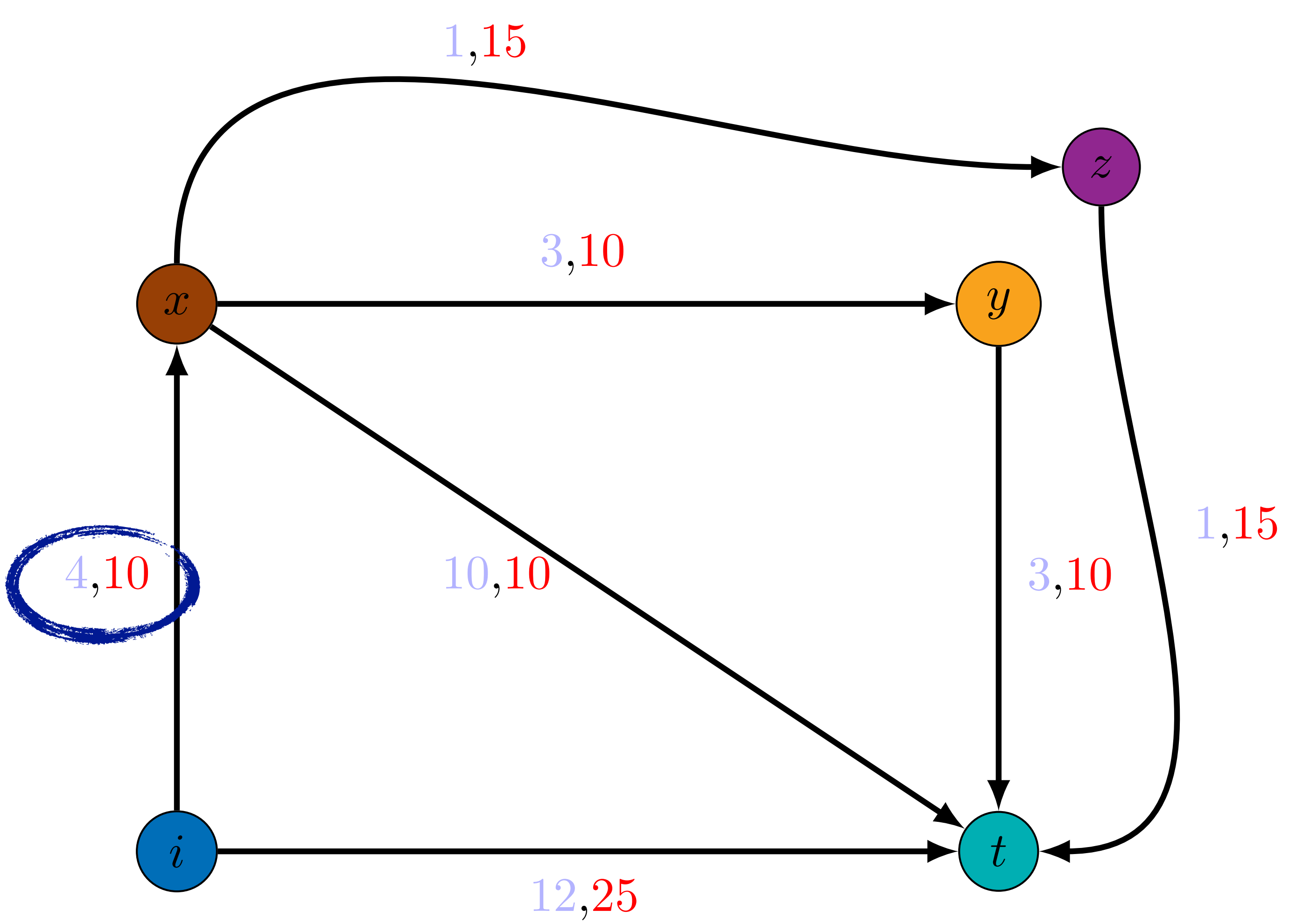
Experiment 1: No Variance. $\delta = c_{xy}^T$



Videos 1 and 2

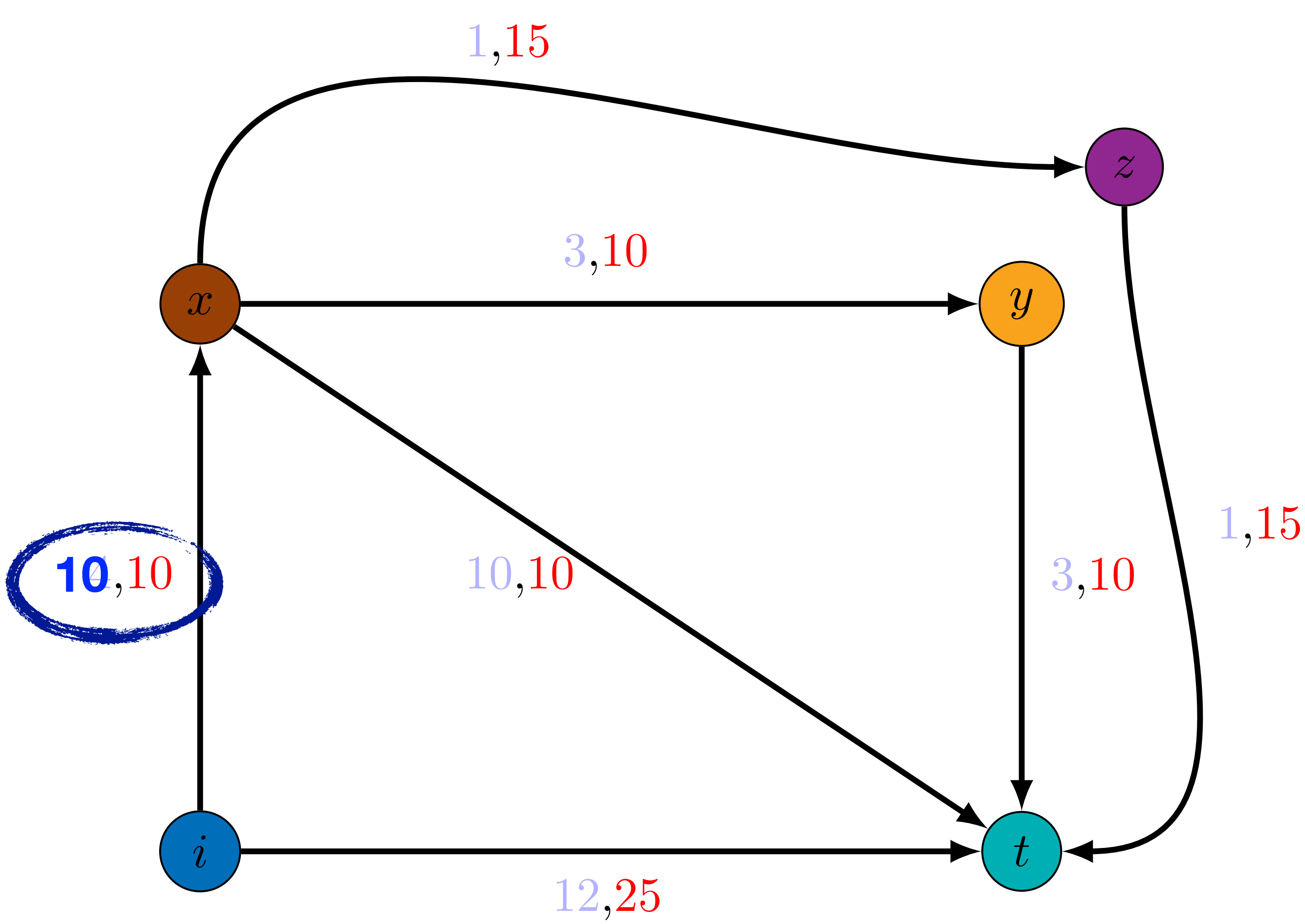
Experiment 2: Congestion on link

- Tests the adaptability of our algorithm
- Simulate congestion of network
- At episode **40**, c_{ix}^{T} = **10** instead of c_{ix}^{T} = **4**

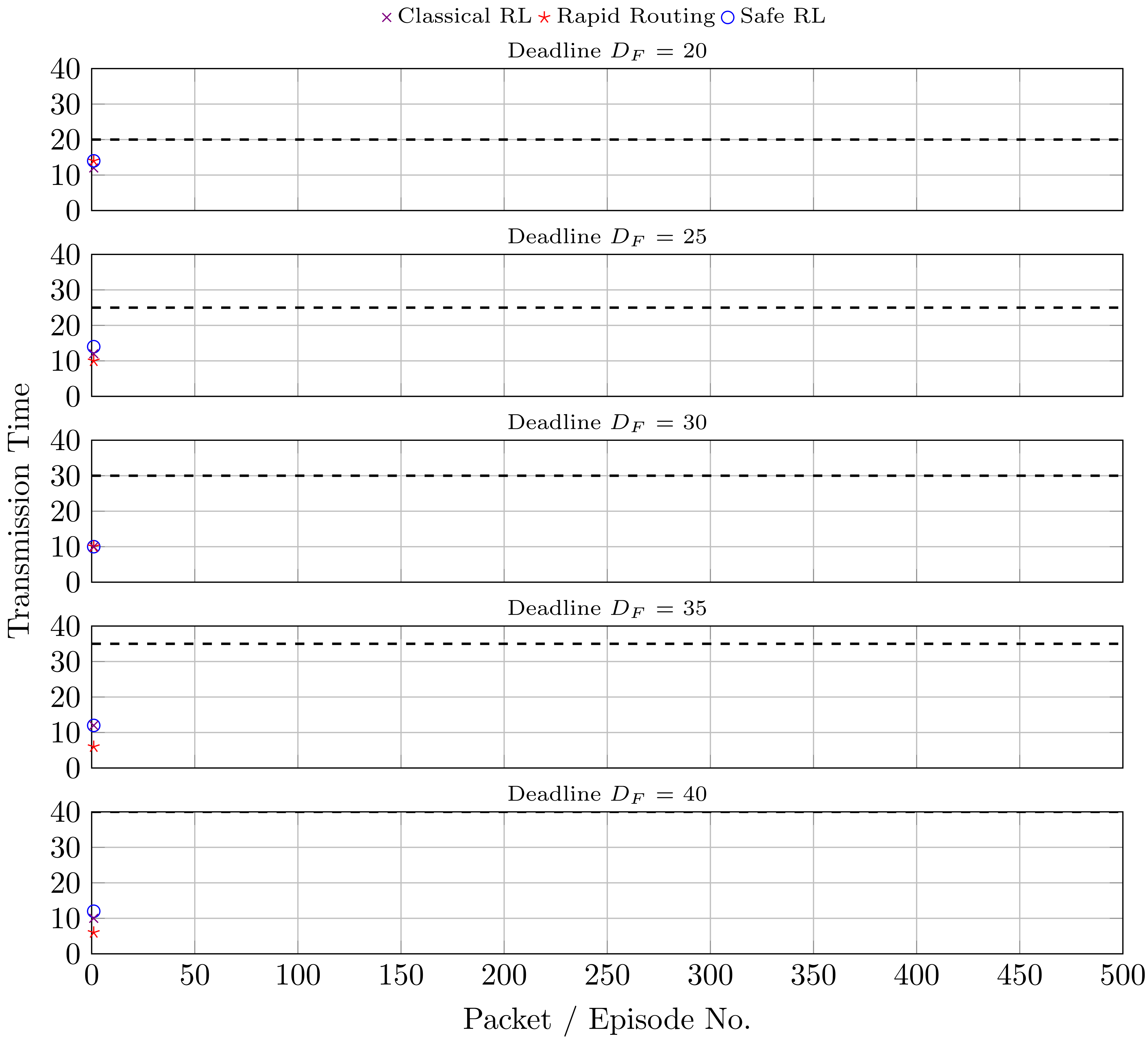


Experiment 2: Congestion on link

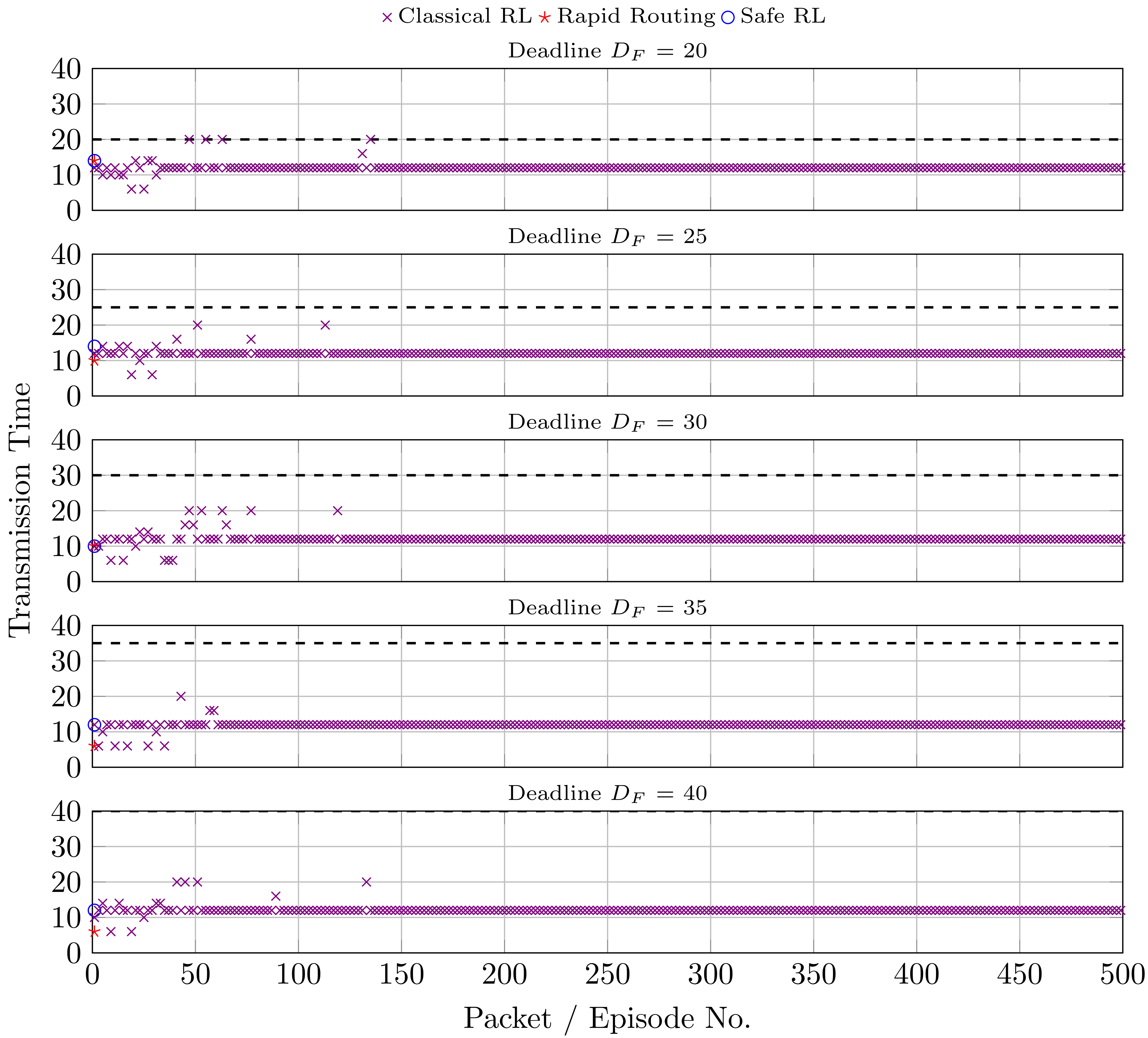
- Tests the adaptability of our algorithm
- Simulate congestion of network
- At episode **40**, c_{ix}^{T} = **10** instead of c_{ix}^{T} = **4**



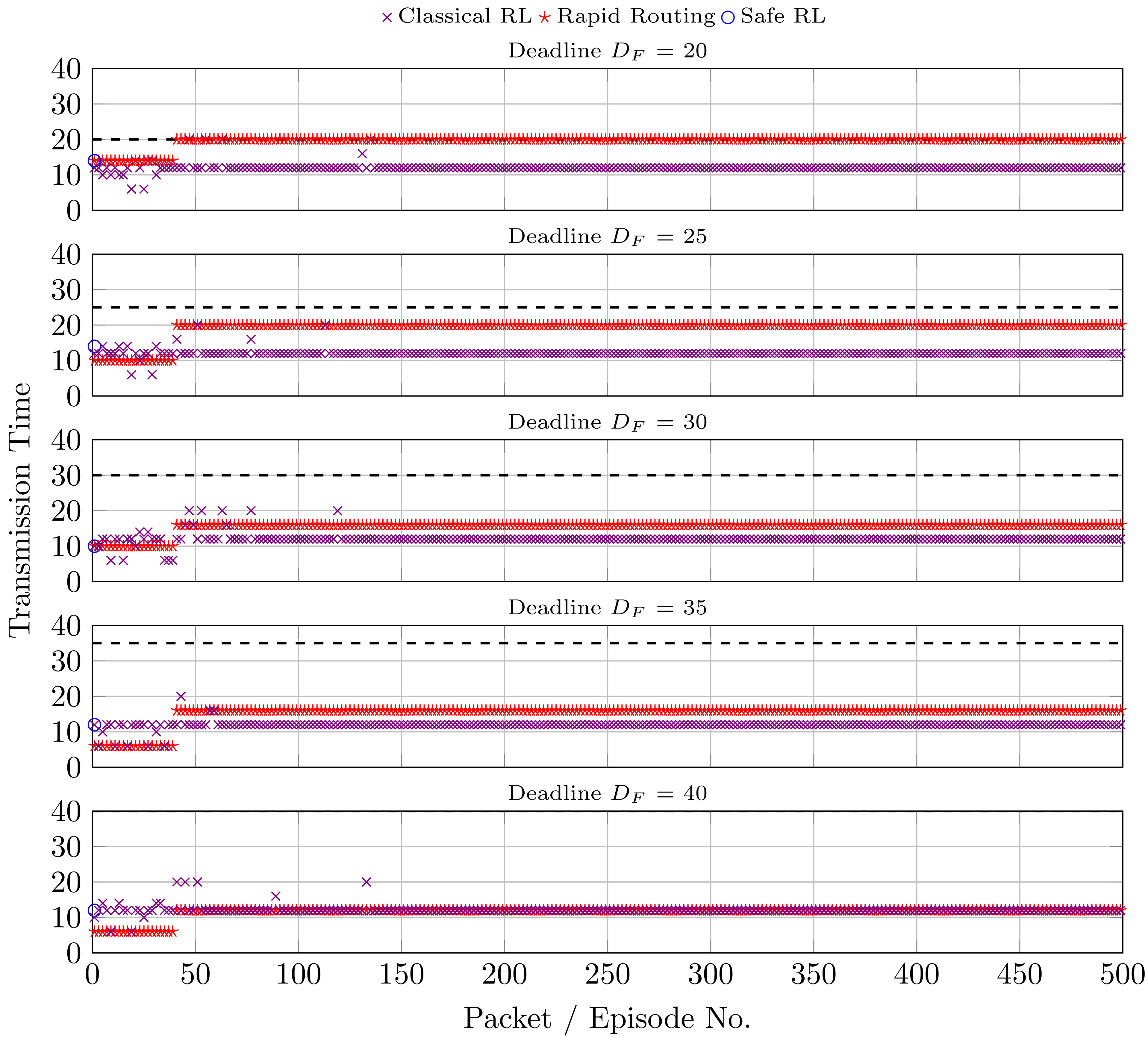
Experiment 2: Congestion on link.



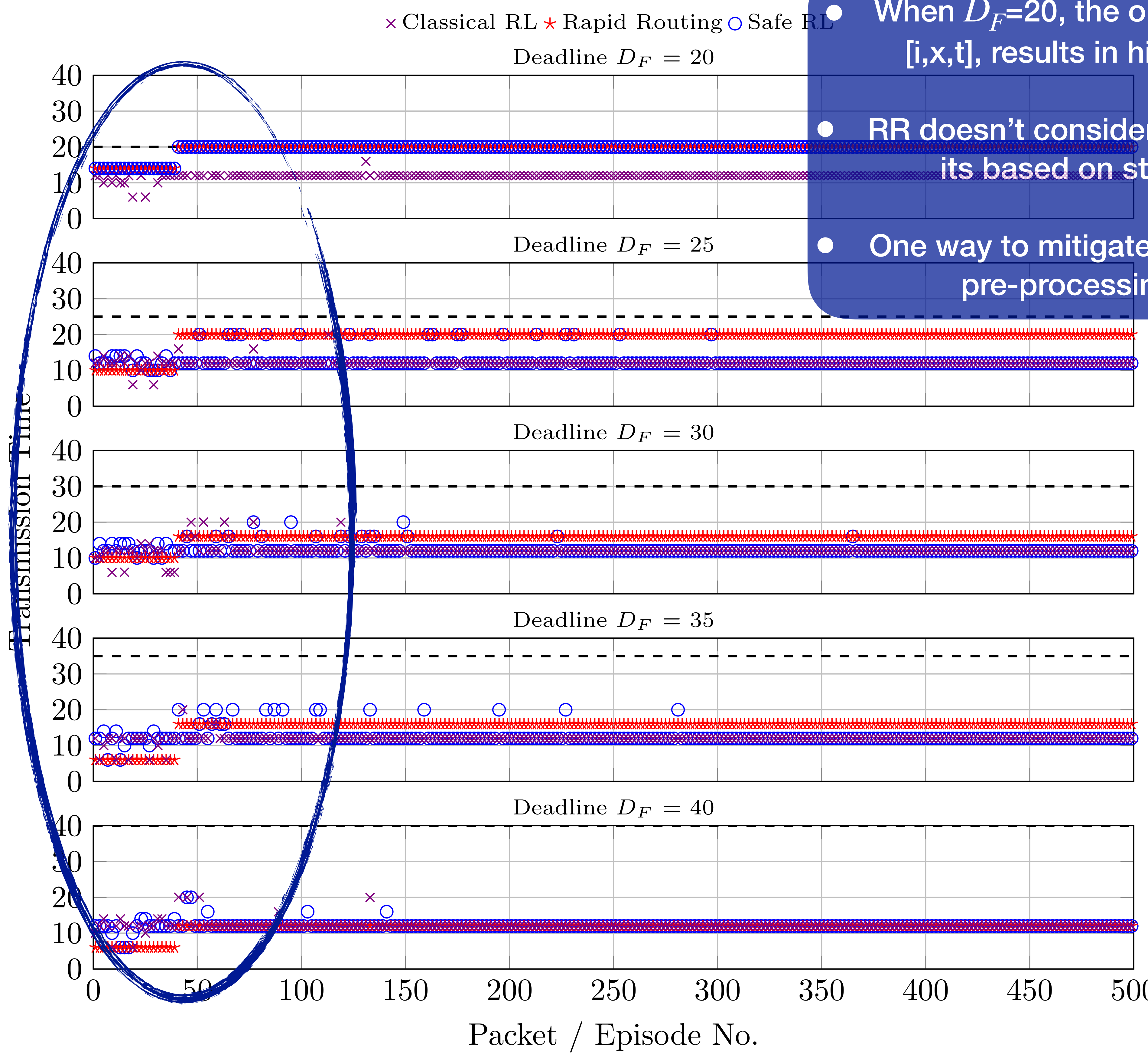
Experiment 2: Congestion on link



Experiment 2: Congestion on link

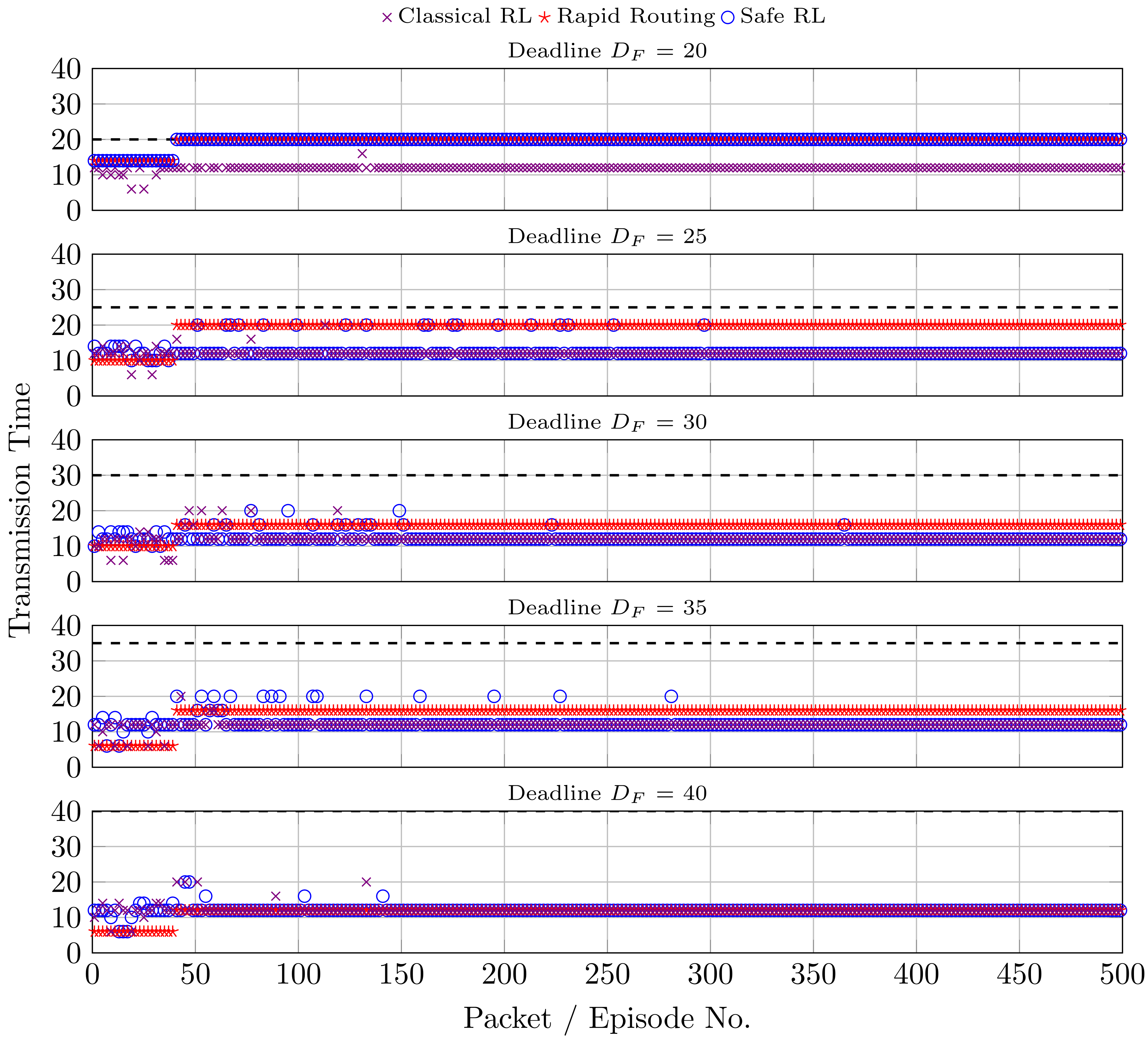


Experiment 2: Congestion on link



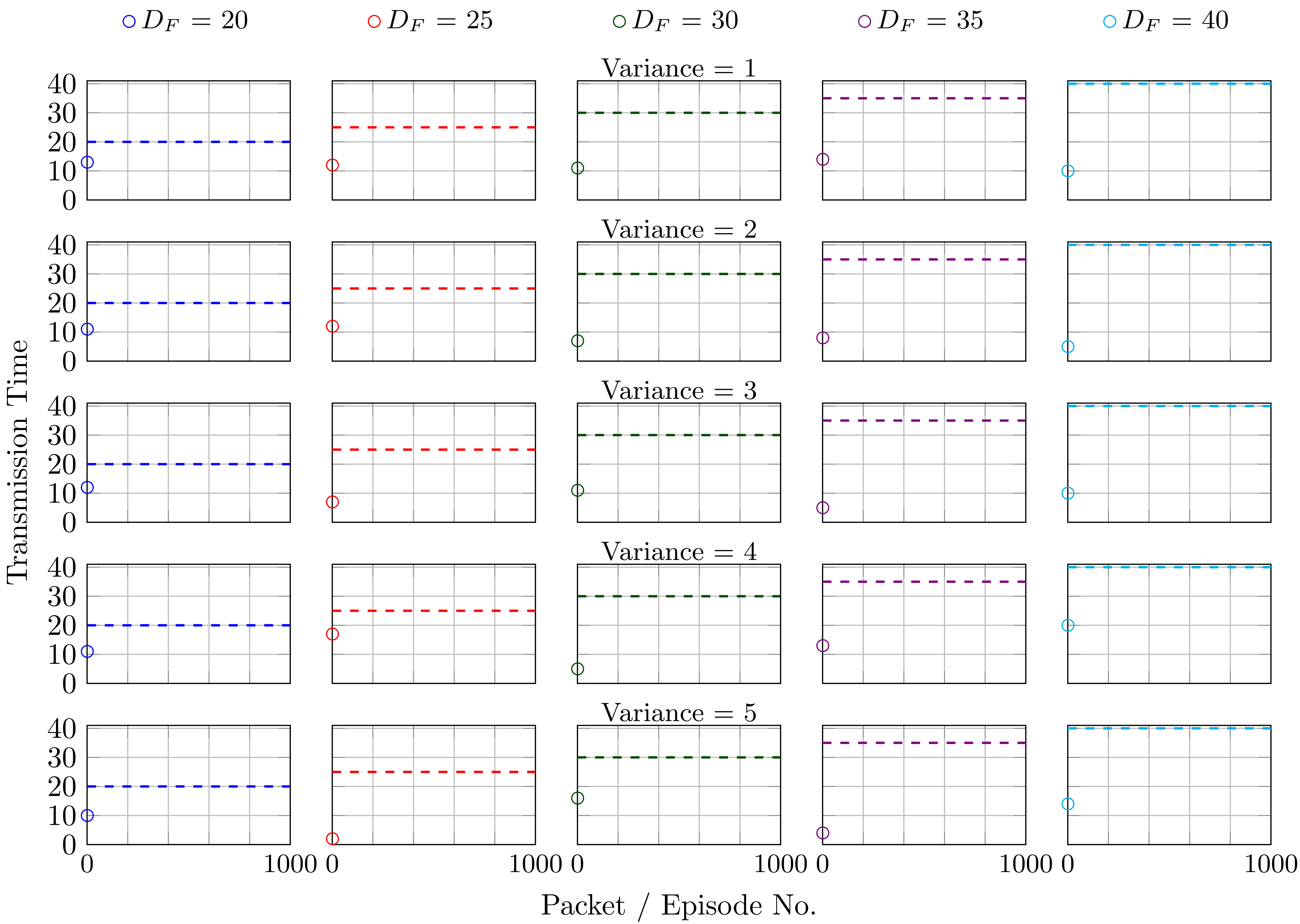
- When D_F=20, the only safe path is [i,x,t], results in higher tx times
- RR doesn't consider disturbance as its based on static tables.
- One way to mitigate it is to redo the pre-processing/tables

Experiment 2: Congestion on link

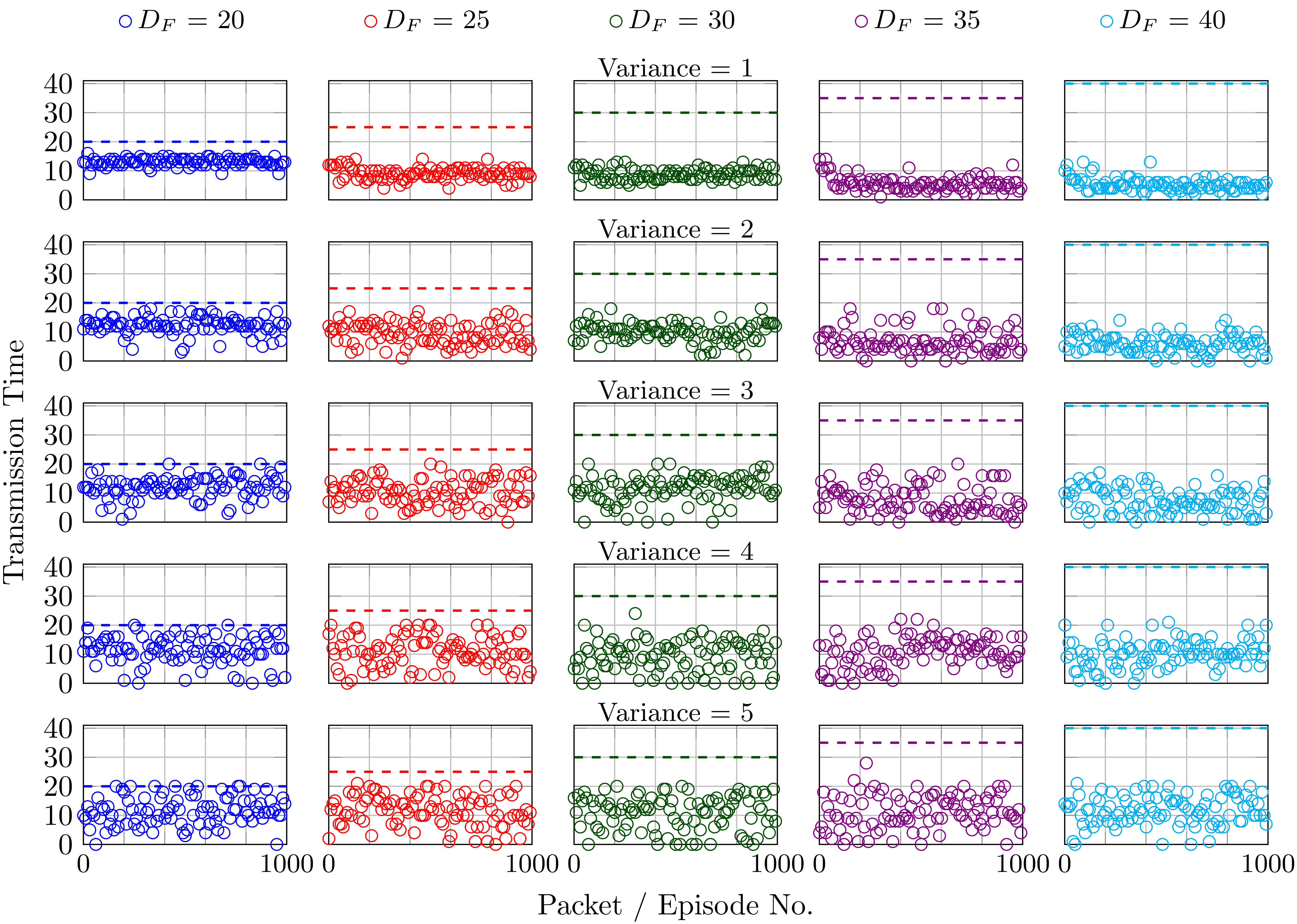


Video 3

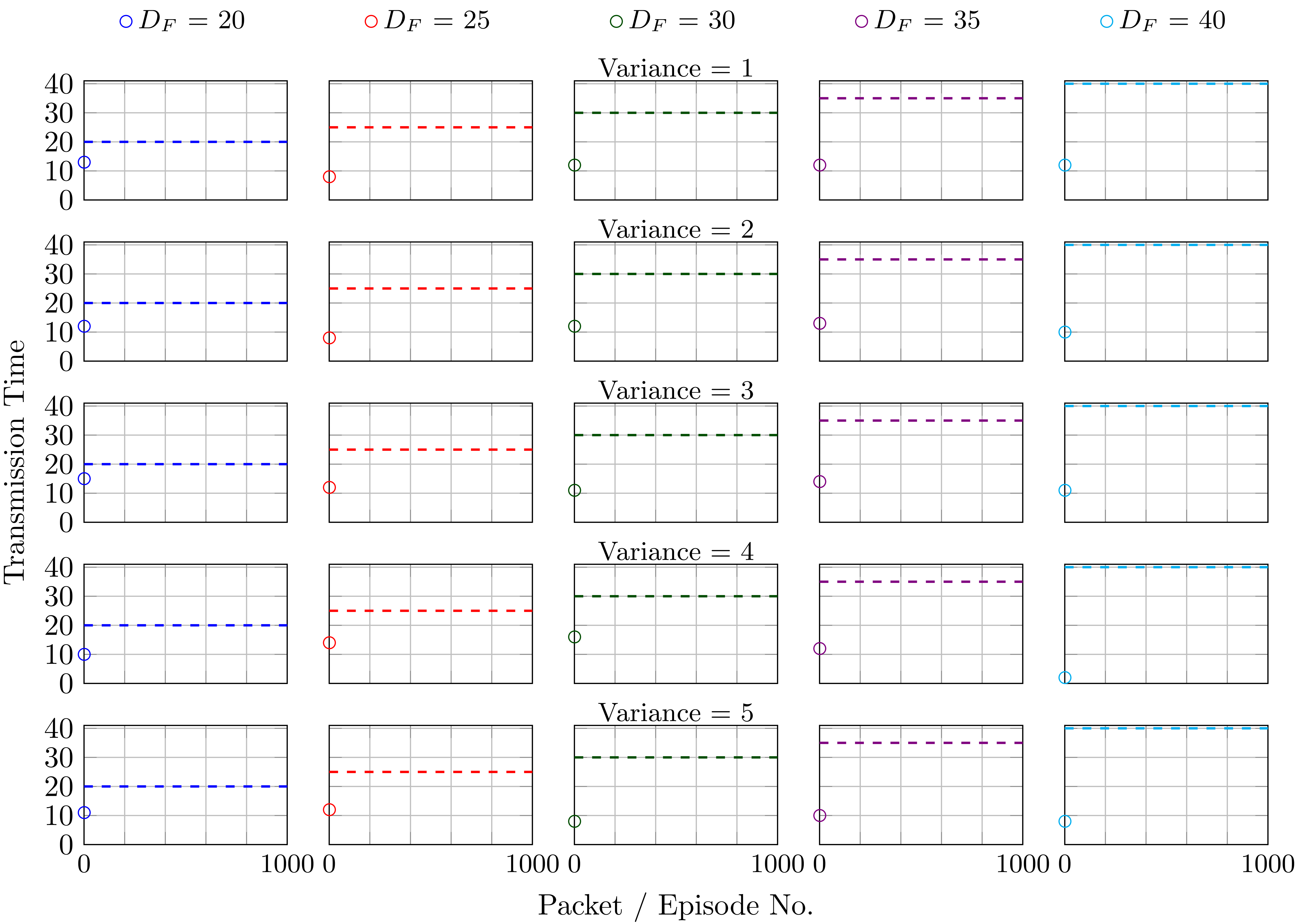
Experiment 3a: Truncated Normal Distribution



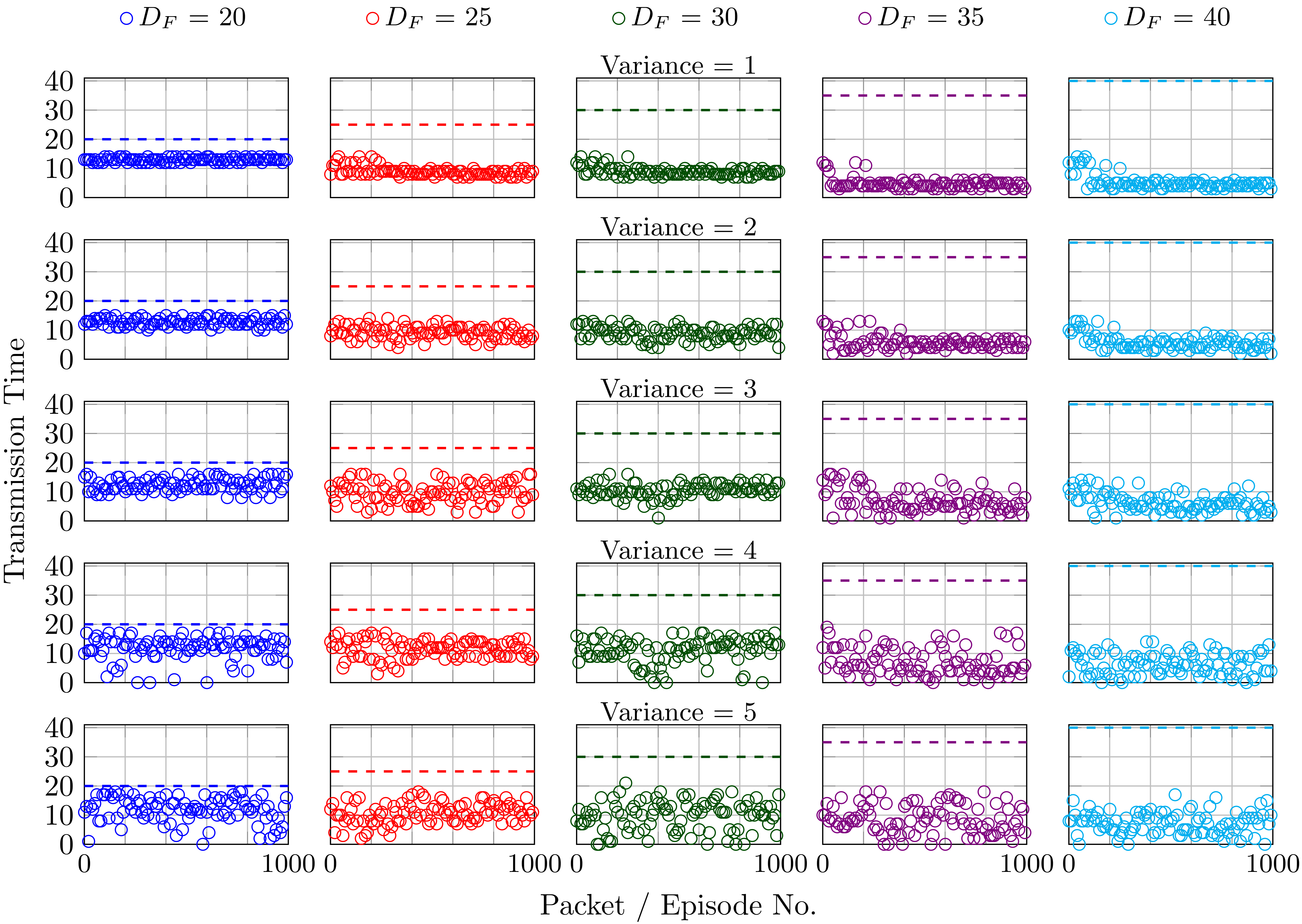
Experiment 3a: Truncated Normal Distribution



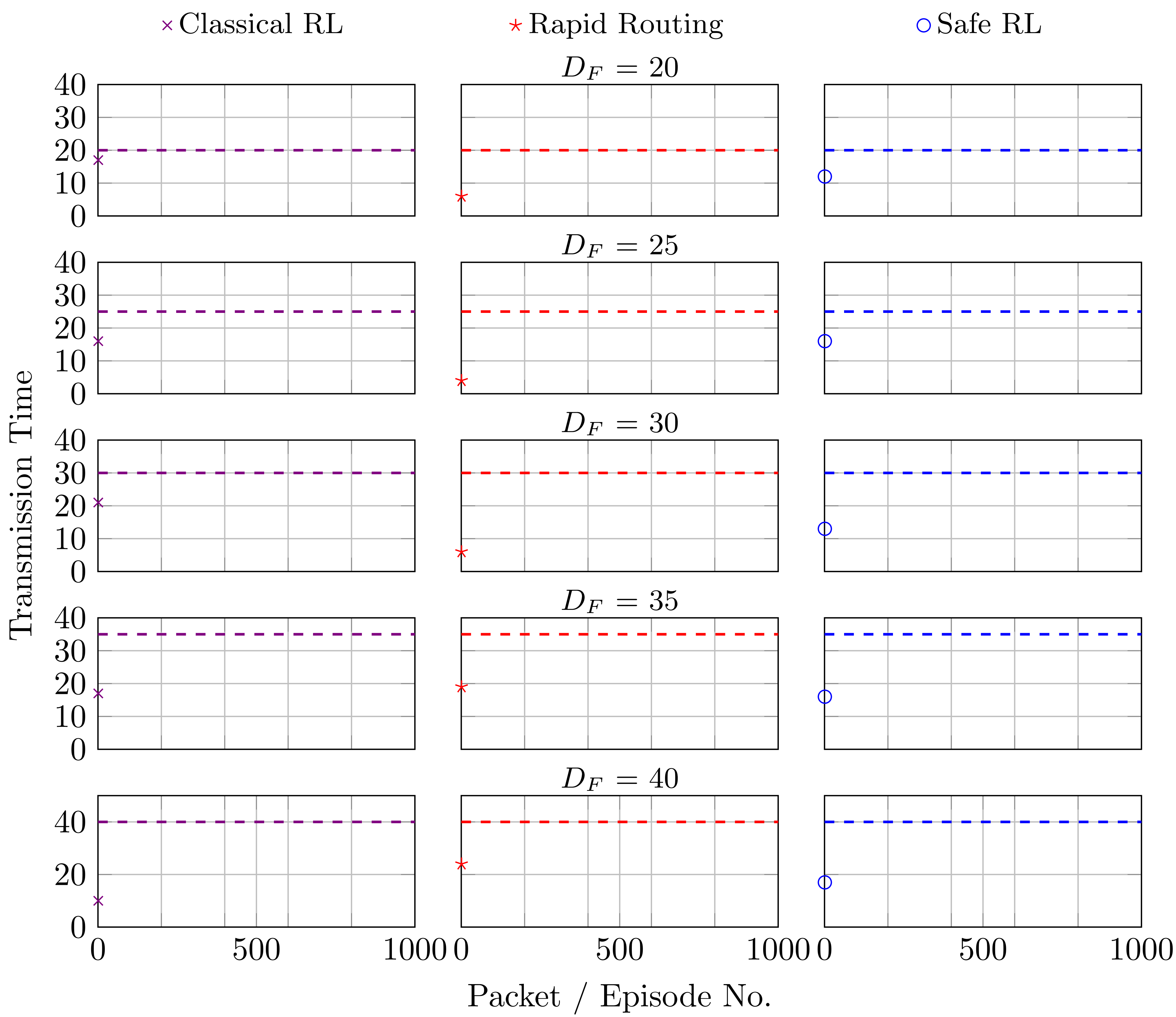
Experiment 3b: Uniform Distribution



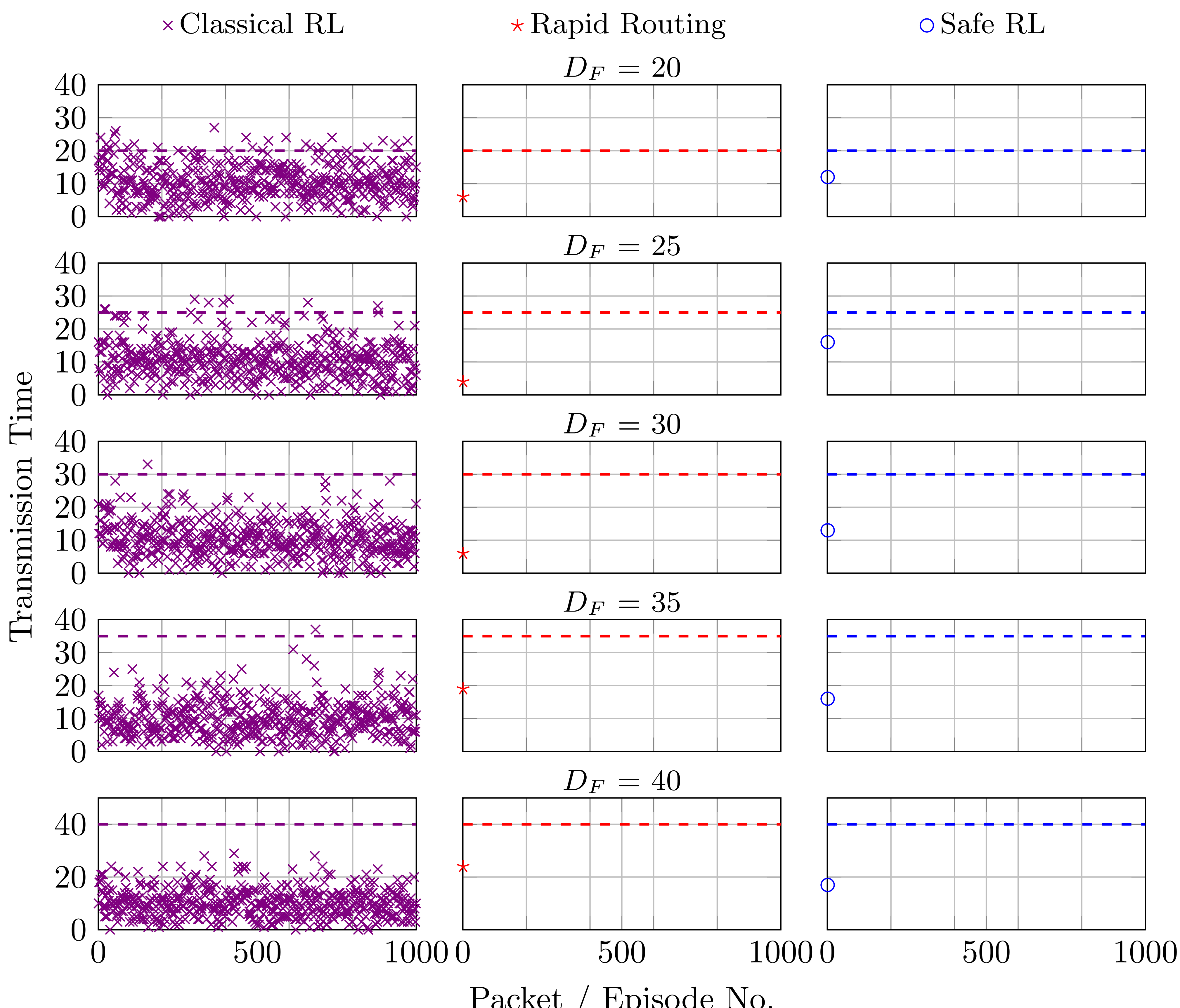
Experiment 3b: Uniform Distribution



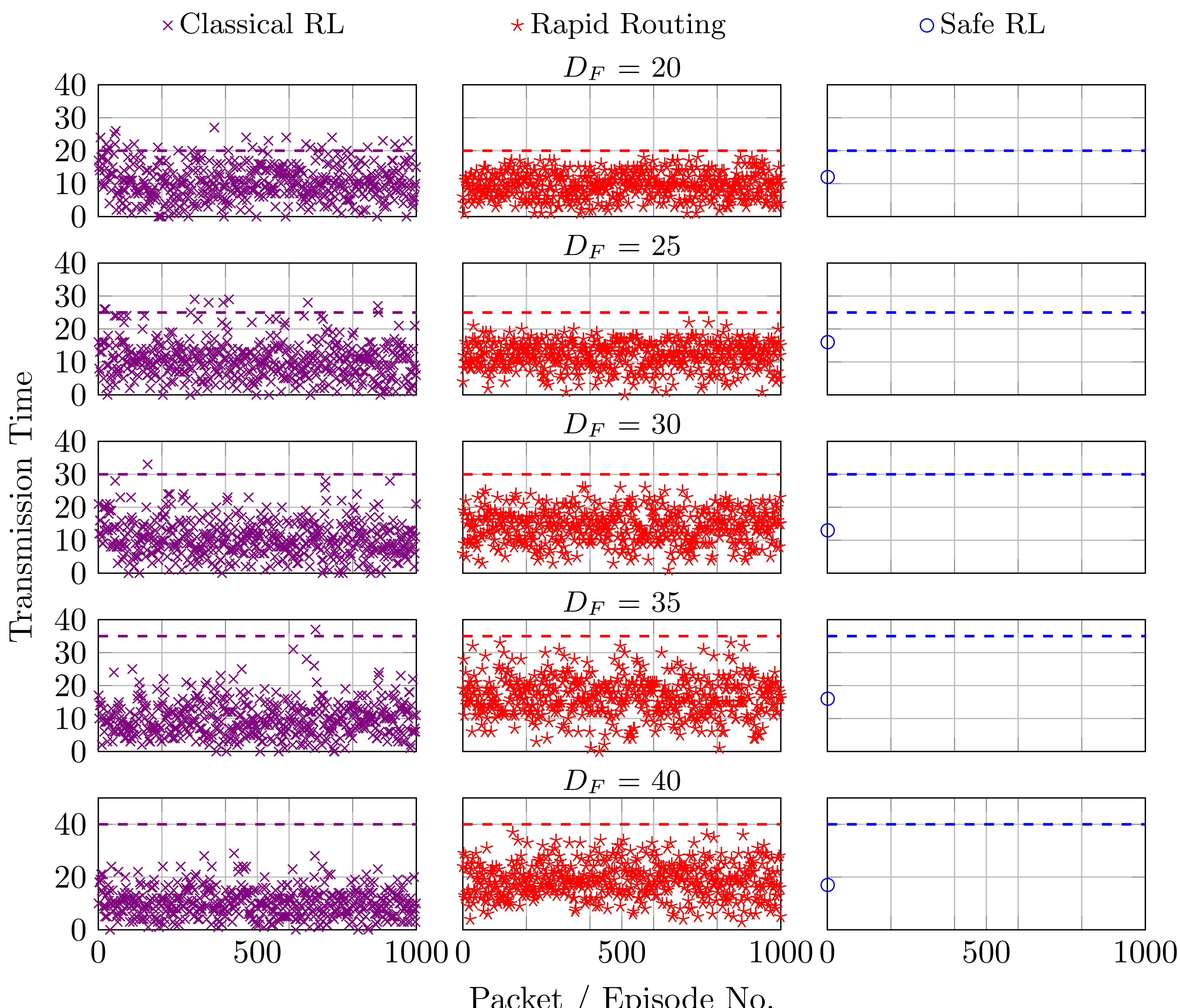
Experiment 4: WC-Uniform Distribution. $0 < \delta \leq c_{xy}^{W}$



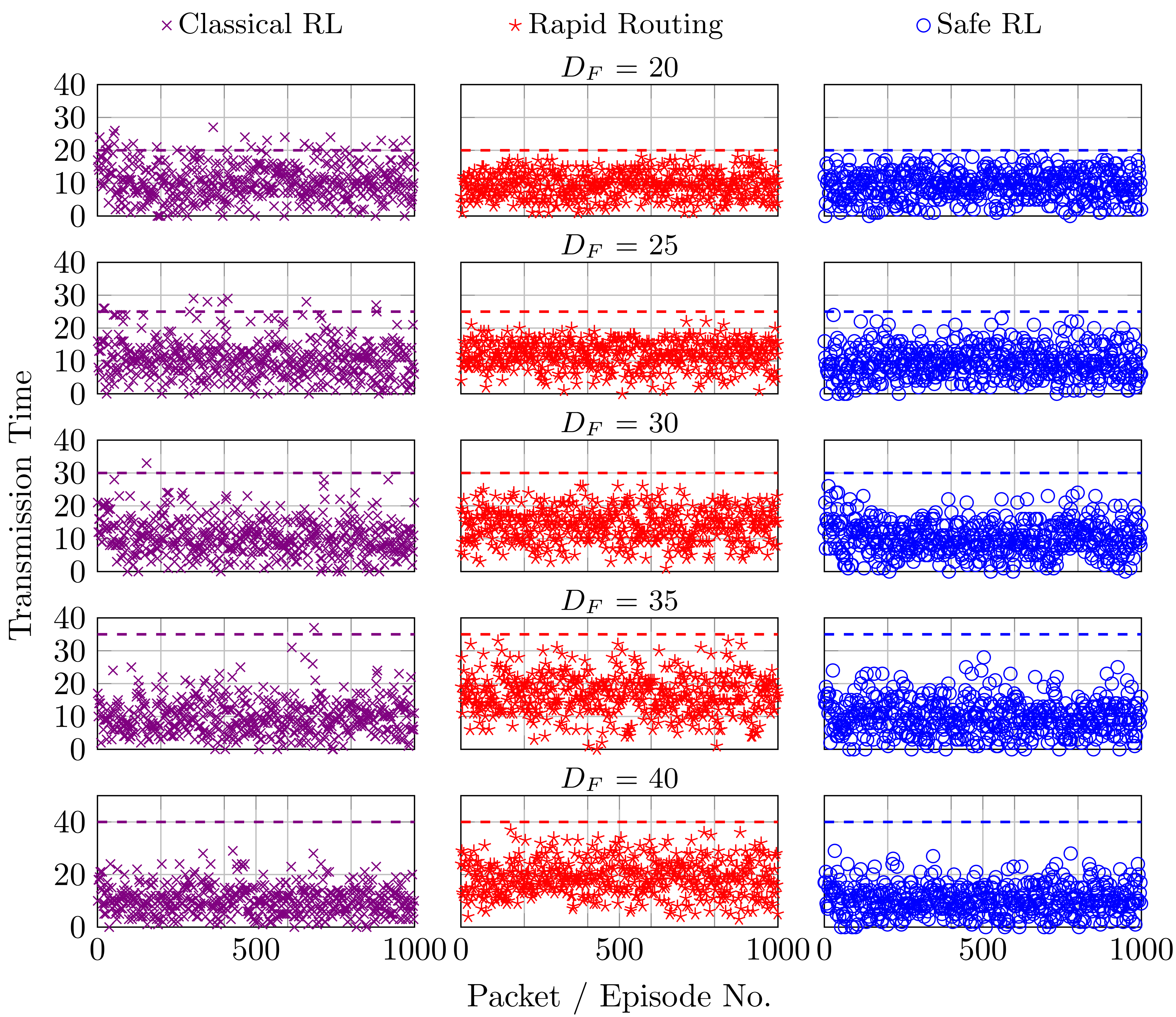
Experiment 4: WC-Uniform Distribution. $0 < \delta \leq c_{xy}^{W}$



Experiment 4: WC-Uniform Distribution. $0 < \delta \leq c_{xy}^{W}$

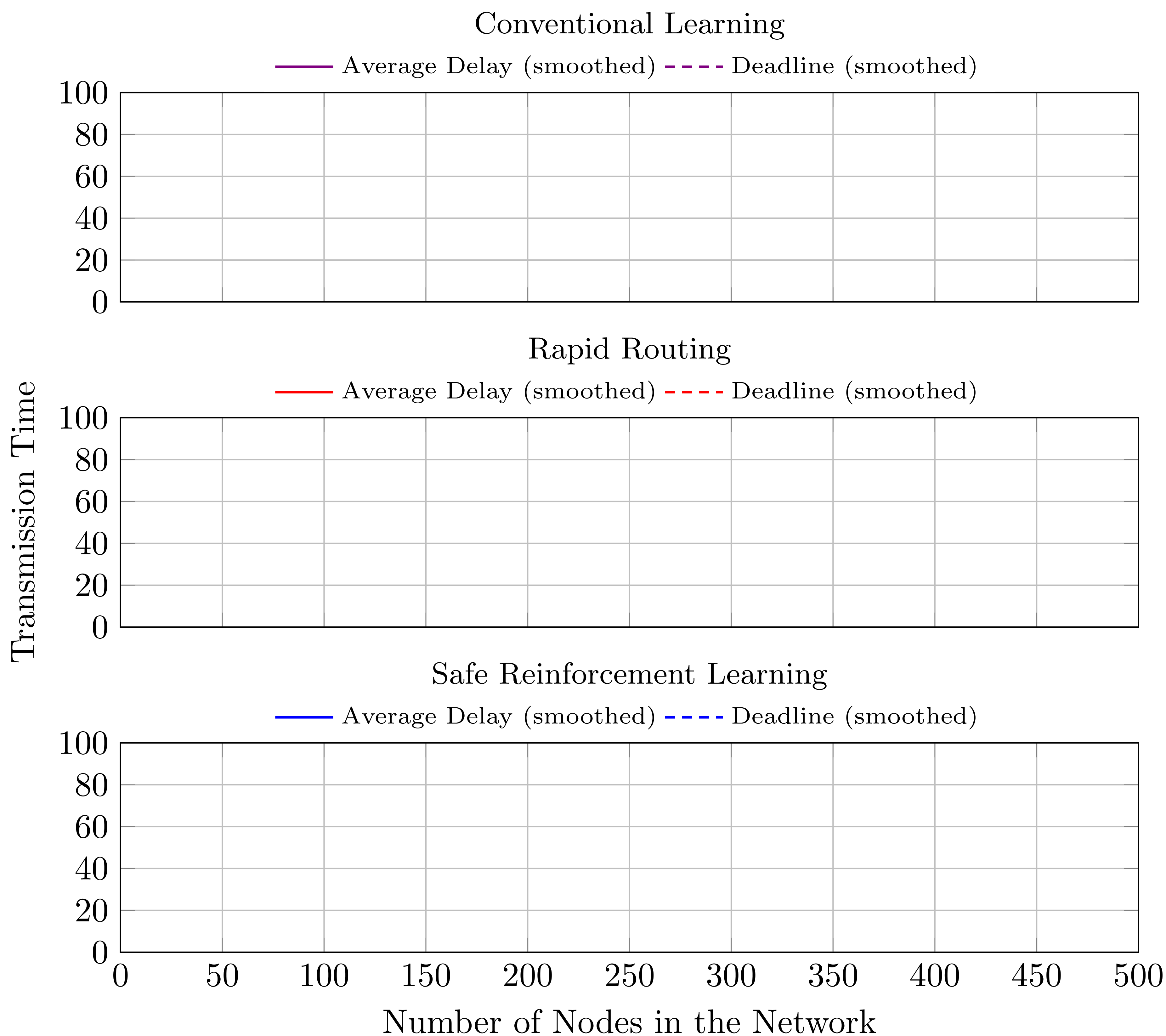


Experiment 4: WC-Uniform Distribution. $0 < \delta \leq c_{xy}^W$

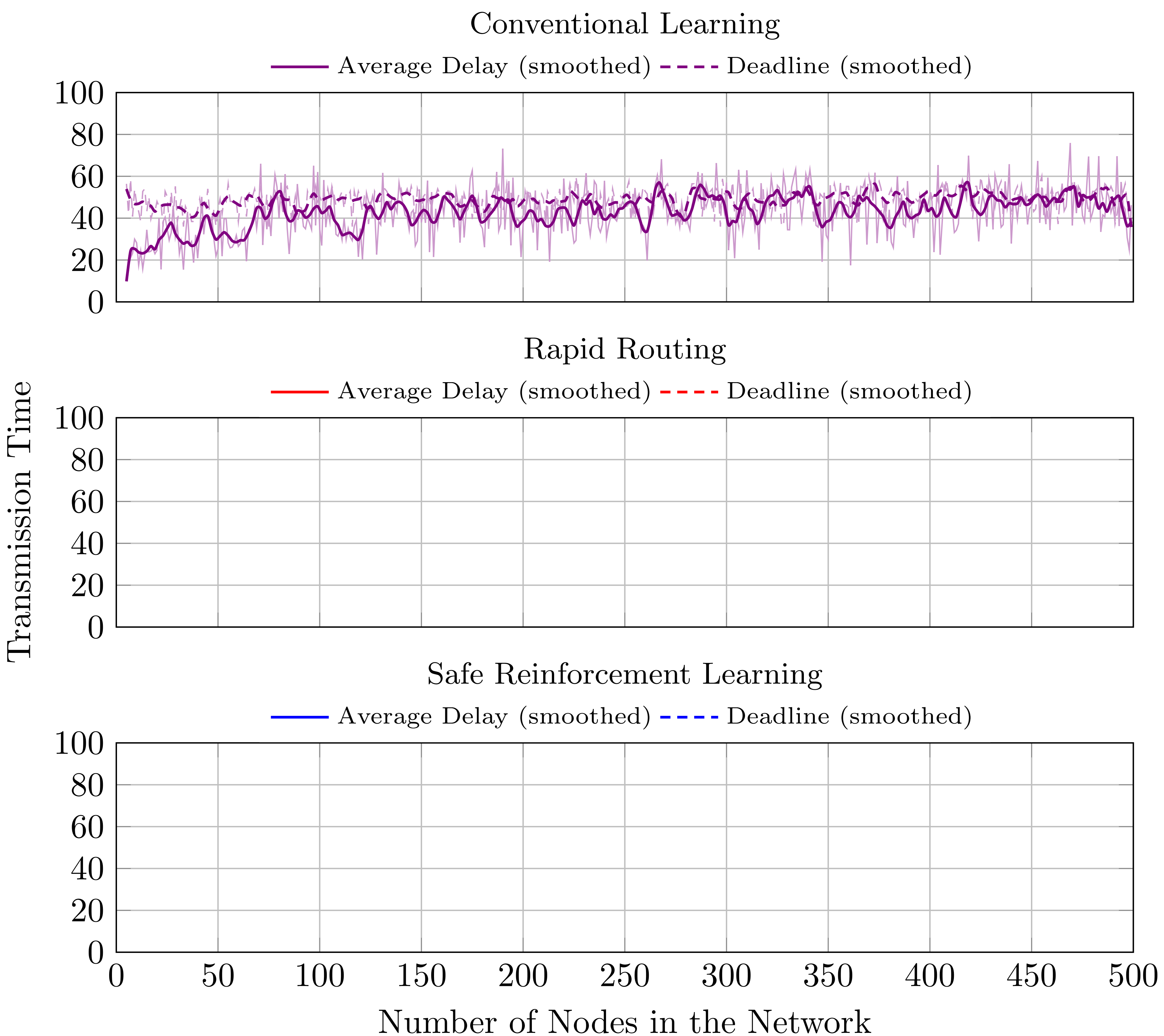


Video 4

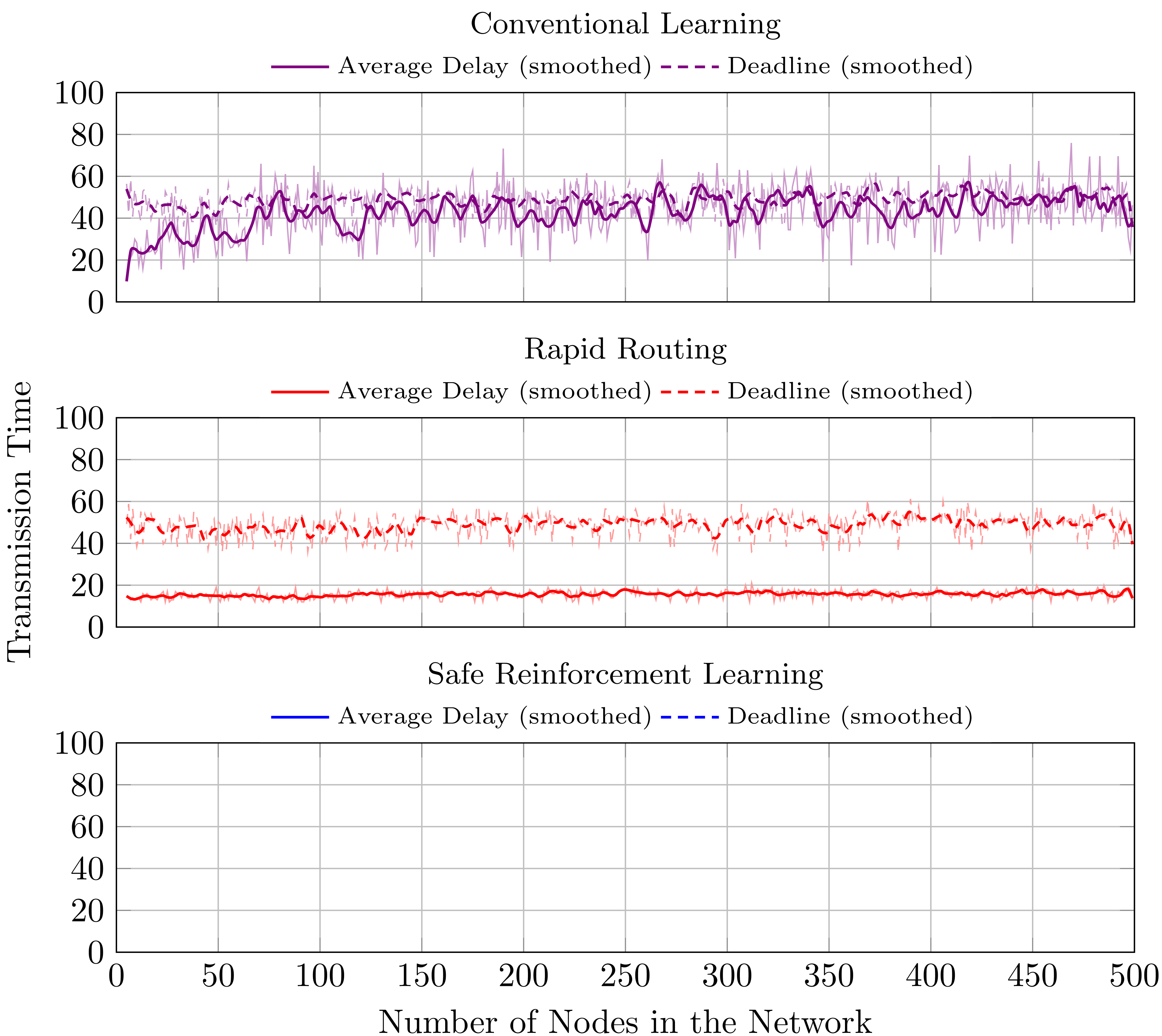
Experiment 5: Large Networks



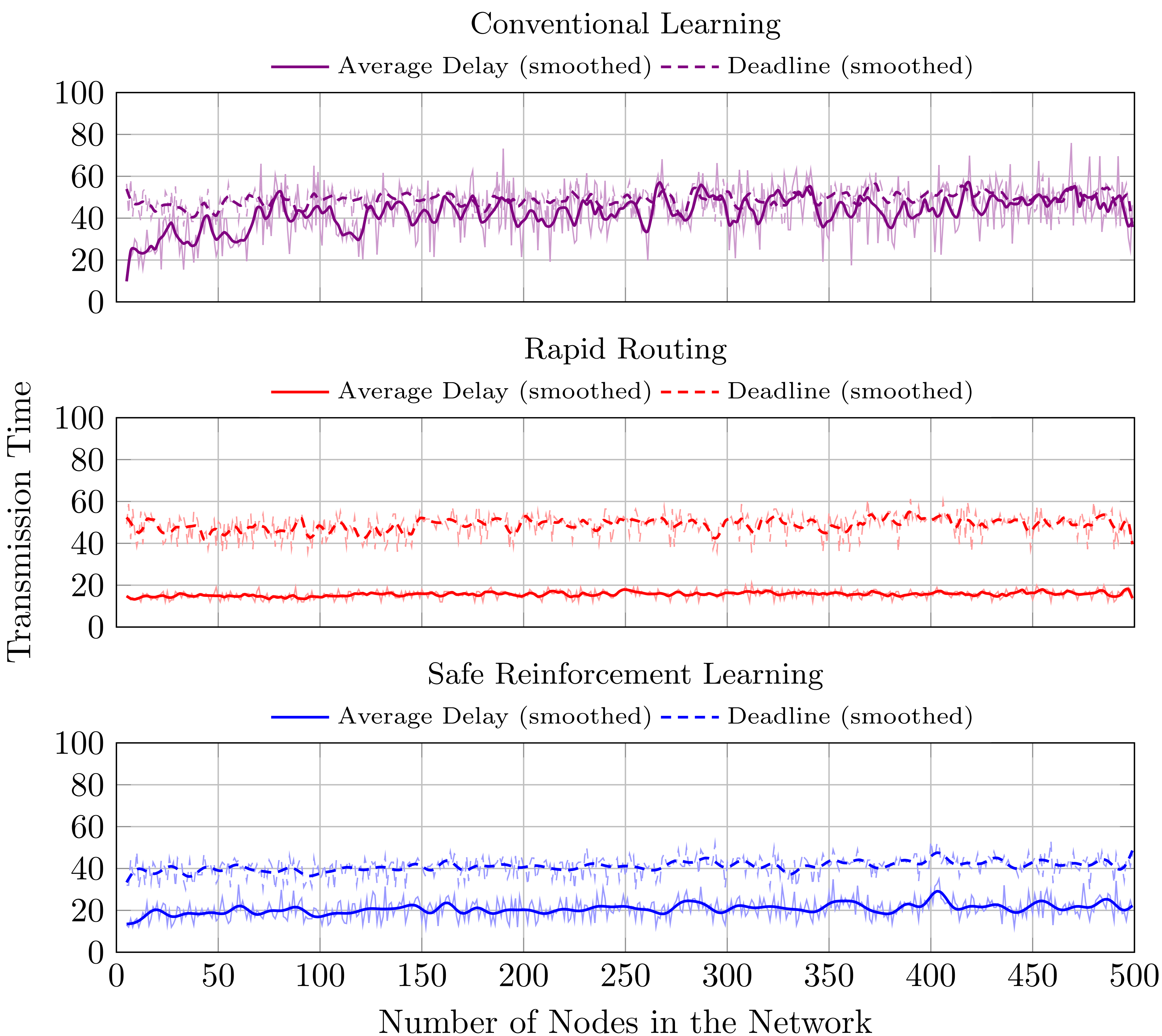
Experiment 5: Large Networks



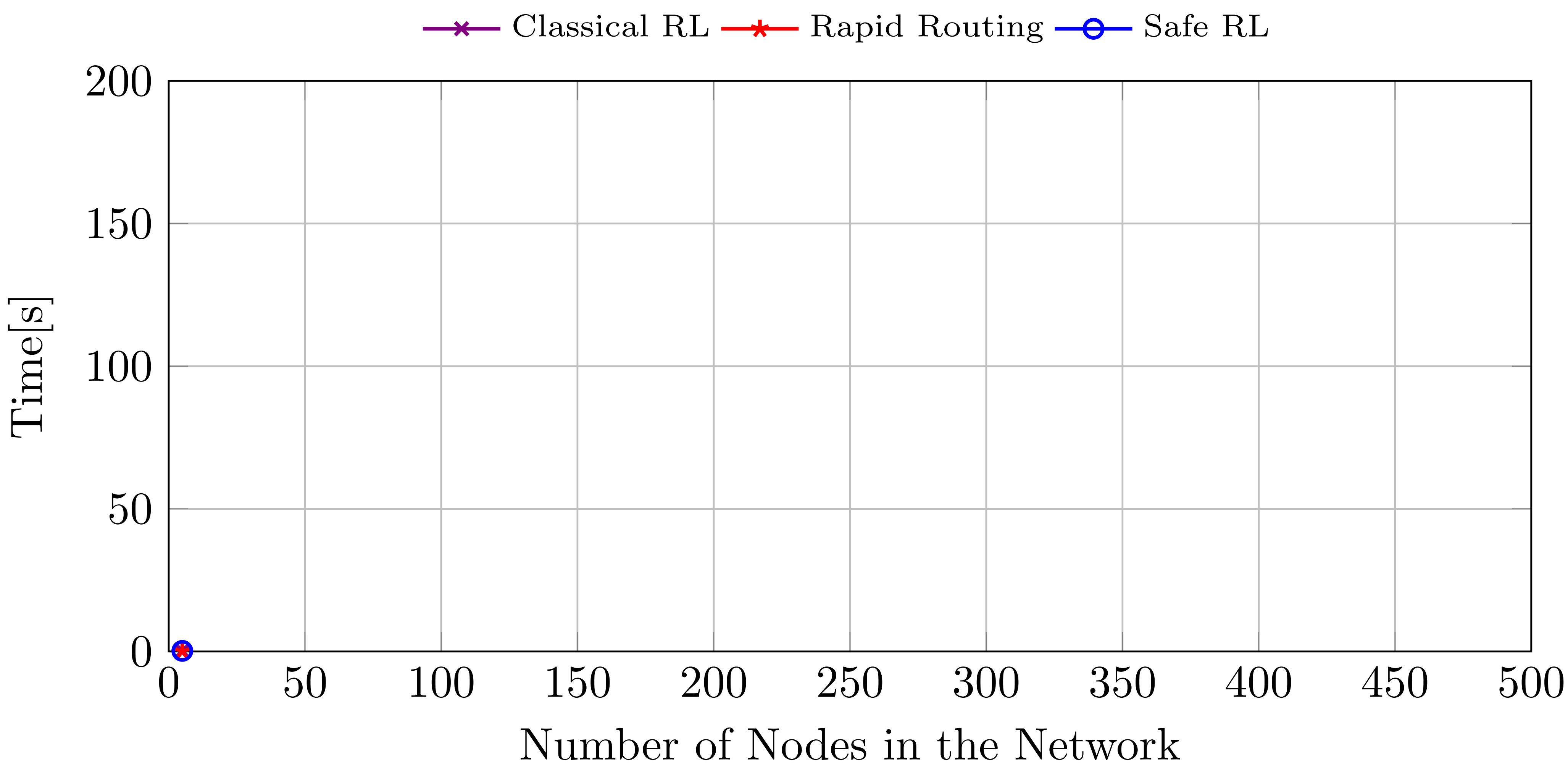
Experiment 5: Large Networks



Experiment 5: Large Networks

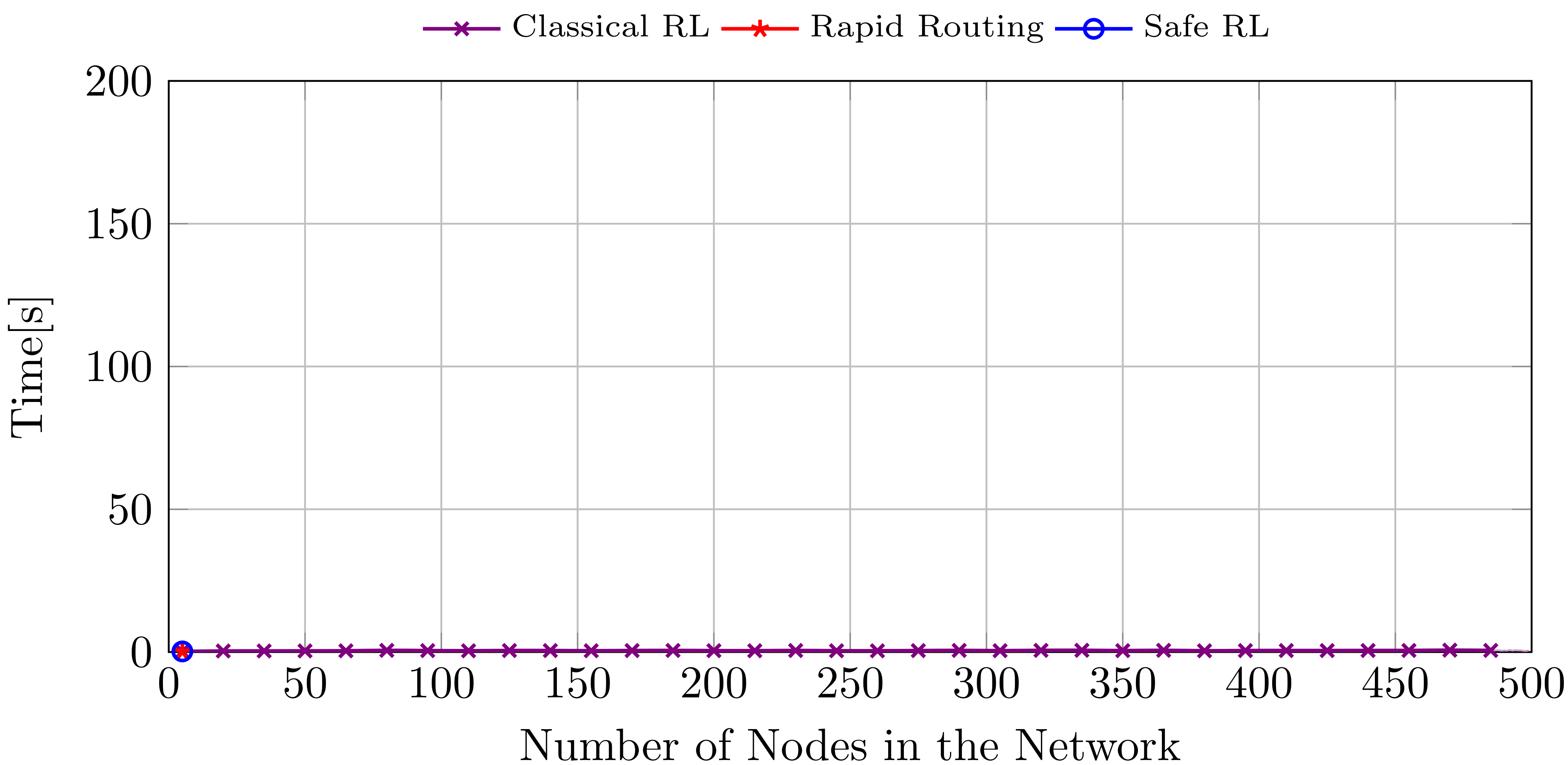


Experiment 5: Computational time



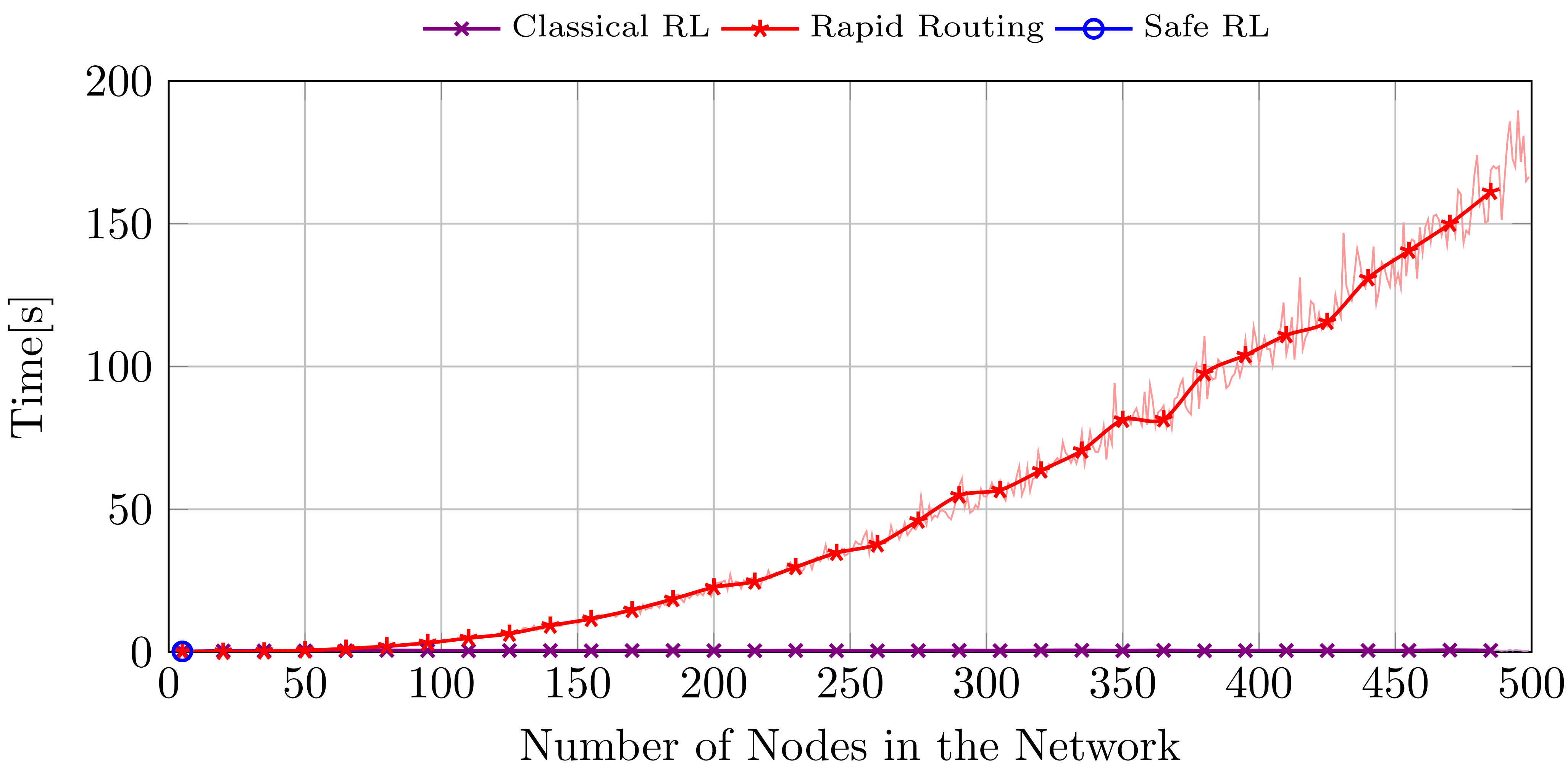
Experiment 5: Computational time

- Classical RL has low complexity (ms), but doesn't provide guarantees



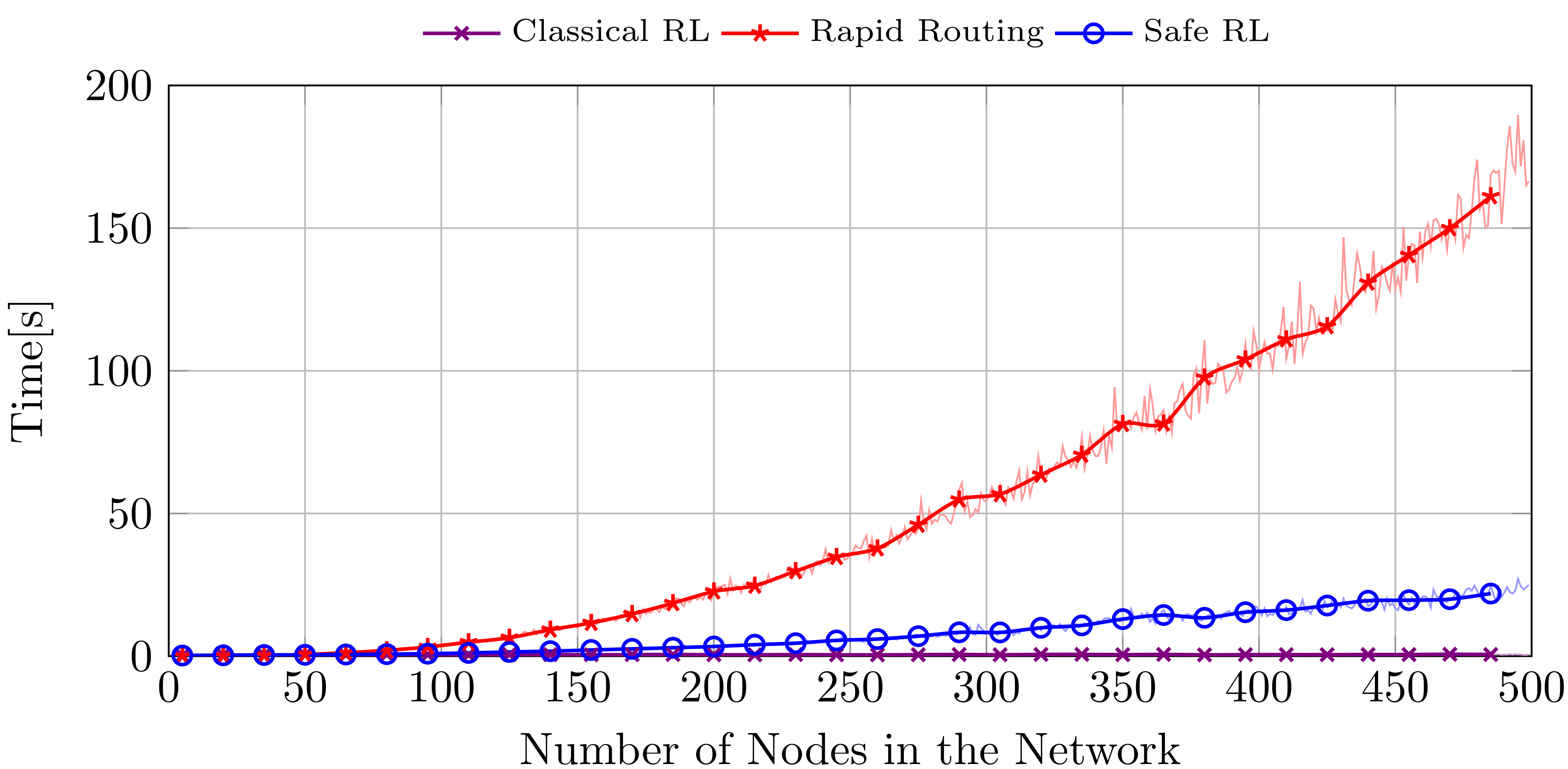
Experiment 5: Computational time

- Classical RL has low complexity (ms), but doesn't provide guarantees
- Rapid Routing needs to be rerun when typical tx time changes



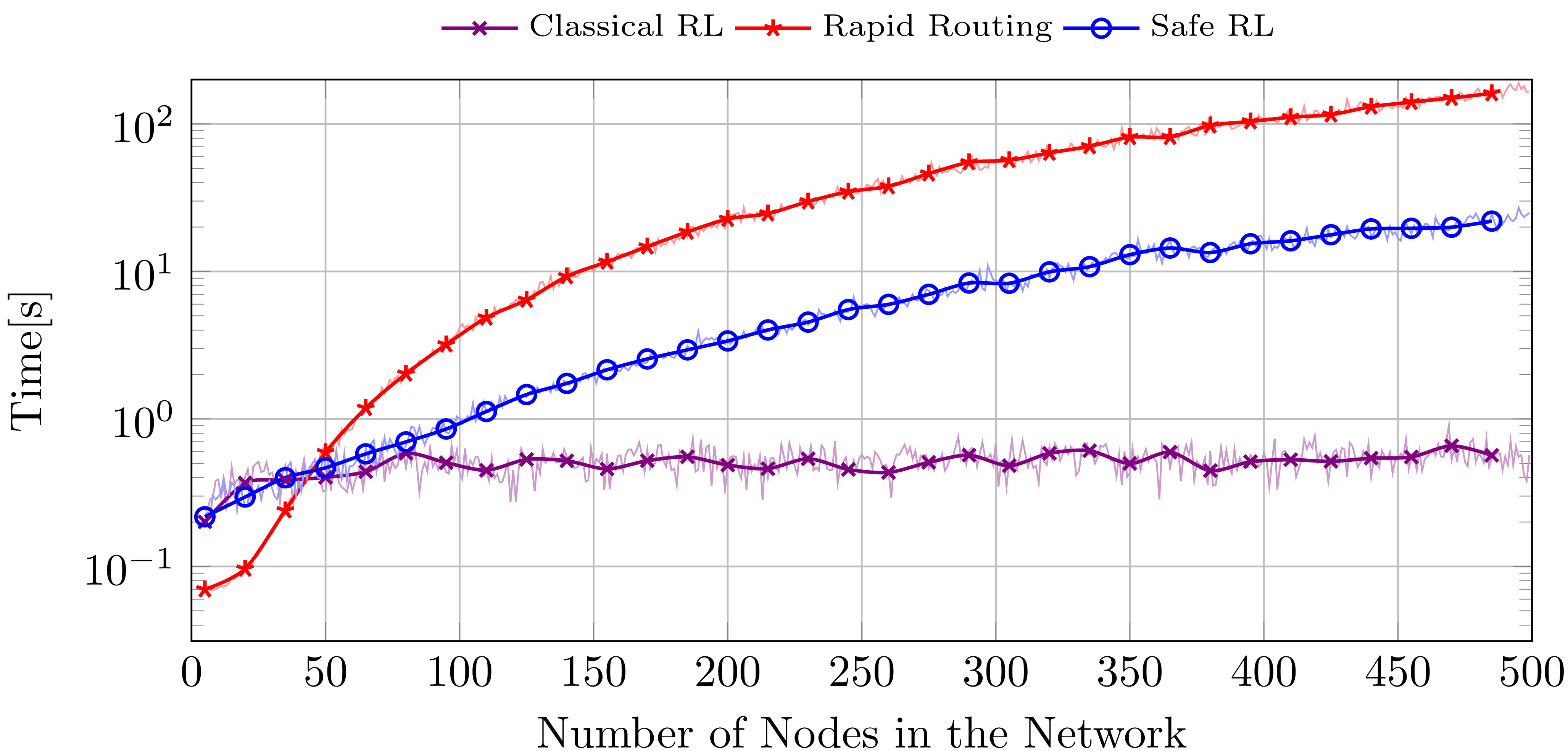
Experiment 5: Computational time

- Classical RL has low complexity (ms), but doesn't provide guarantees
- Rapid Routing needs to be rerun when typical tx time changes
- Most complexity of safe RL comes from pre-processing stage. Run only once during network creation



Experiment 5: Computational time

- Classical RL has low complexity (ms), but doesn't provide guarantees
- Rapid Routing needs to be rerun when typical tx time changes
- Most complexity of safe RL comes from pre-processing stage. Run only once during network creation



Conclusion

- Applied reinforcement learning to routing over real-time networks

Conclusion

- Applied reinforcement learning to routing over real-time networks
- Augmented state-space allows safe exploration

Conclusion

- Applied reinforcement learning to routing over real-time networks
- Augmented state-space allows safe exploration
- Constant adaptation to changes in typical transmission time

Conclusion

- Applied reinforcement learning to routing over real-time networks
- Augmented state-space allows safe exploration
- Constant adaptation to changes in typical transmission time
- Compared to classical RL, our algorithm is robust and does not violate any deadlines

Conclusion

- Applied reinforcement learning to routing over real-time networks
- Augmented state-space allows safe exploration
- Constant adaptation to changes in typical transmission time
- Compared to classical RL, our algorithm is robust and does not violate any deadlines
- Compared to previous work, our algorithm
  - Adapts online to changes in typical transmission time
  - Is less computationally intensive

Future Work

Future Work

- Implement on a network emulator
    - Thank you Alex for pointers.

Future Work

- Implement on a network emulator
  - Thank you Alex for pointers.
- Investigate probability propagation through network

Future Work

- Implement on a network emulator
    - Thank you Alex for pointers.
- Investigate probability propagation through network
- Is there anyway to guarantee safety if loops are present in the network?

gautham@control.lth.se