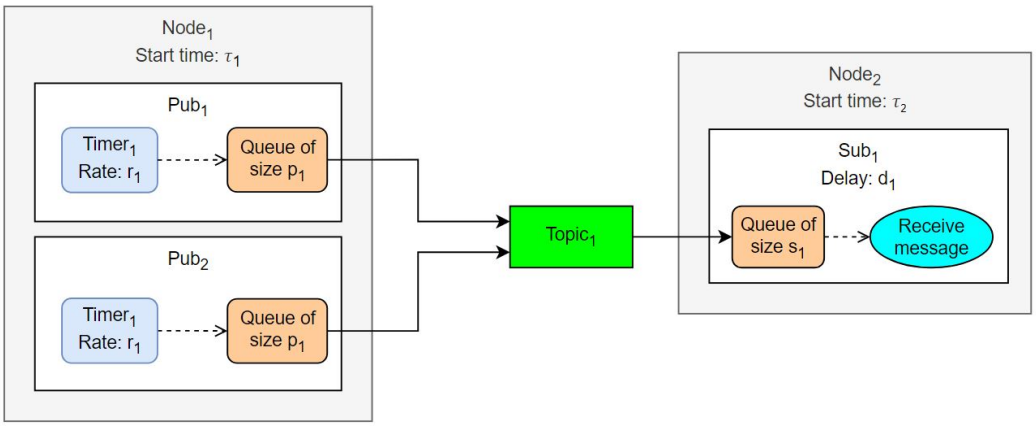

Figure 1: Overview of a ROS program

1 Formal Modeling of ROS Program

A ROS program is a tuple $P = (Pub, Sub, Timer, Topic, Node, T_{pub}, Q_{size}, T_{rate}, E, D, T_0)$ where,

- Pub is a set of publishers
- Sub is a set of subscribers
- $Timer$ is a set of timers
- $Topic$ is a set of topics
- $Node = \{node \mid node \subseteq Pub \cup Sub\}$ is a set of nodes
- $T_{pub} : Timer \mapsto Pub$ is the timer assignment for each publisher
- $Q_{size} : Pub \cup Sub \mapsto \mathbb{N}$ denotes the queue sizes of each publisher and subscriber
- $T_{rate} : Timer \mapsto \mathbb{R}_{\geq 0}$ is the rate of publishing for a timer
- $E : Pub \cup Sub \mapsto Topic$ is a transition function denoting the topic for each publisher to publish and each subscriber subscribed
- $D : Sub \mapsto \mathbb{R}_{\geq 0}$ is the processing time for each subscriber to transfer from the publisher
- $T_0 : Node \mapsto \mathbb{R}_{\geq 0}$ is the start time of each node

2 Example

We are looking at a simplified modeling of a Kobuki robot illustrated in Fig. 2. Kobuki is integrated with 3 different types of sensors, velocity controllers, and high-precision motors. The ROS-based architecture has 5 different nodes: Sensor_Publisher_Node, Safety_Controller_Node, Random_Walker_Node, Multiplexer_Node, and Base_Motor_Node.

The Sensor_Publisher_Node has two wheel drop sensors, three bumper sensors, and three cliff sensors. Both the wheel drop sensors are publishing the sensor values at a fixed rate to a topic named *events/wheel_drop* using the associated publisher queues. All the bumper sensors are publishing into topic *events/bumper* and all the cliff sensors are publishing into *events/cliff* topic.

The Safety_Controller_Node receives the sensor values from three subscribers $QWheel, QBumper$, and $QCliff$ respectively for the wheel, bumper, and cliff sensors. The node processes all the sensor values and calculates the velocity command of the robots. If either of the wheel drop sensors, detects any wheel drop event, then the safety controller node will send a command to stop the robot. If any of the bumper or cliff sensors detect any bumping or cliff events, then the node will send a command to move the robot backward and turn the robot a little bit on the opposite side of the detected event. If there are no events detected by any sensors, the node will not send any commands. The node will send the velocity command by publishing it into a topic named *multiplexer/safety_controller*.

The Random_Walker_Node will send velocity commands to randomly move the robot. If the wheel drop sensors detect any wheel drop event, then the robot will stop. If the bumper or cliff sensor detects any corresponding events, then the robot will randomly turn in any direction, and start moving forward if the sensors do not detect anything. The node will send the velocity command by publishing it into a topic named *multiplexer/random_walker*.

The Multiplexer_Node will receive the velocity commands from both the Safety_Controller_Node and the Random_Walker_Node by using two subscribers $MUXSafetyQ$ and $MUXRandomQ$. The multiplexer will give higher priority to the velocity commands sent by the safety controllers and will execute the random walker commands if it is 'safe' to do so. The Base_Motor_Node will get the actual velocity commands from the Multiplexer_Node through a topic *commands/velocity* and run the motors of the robot.

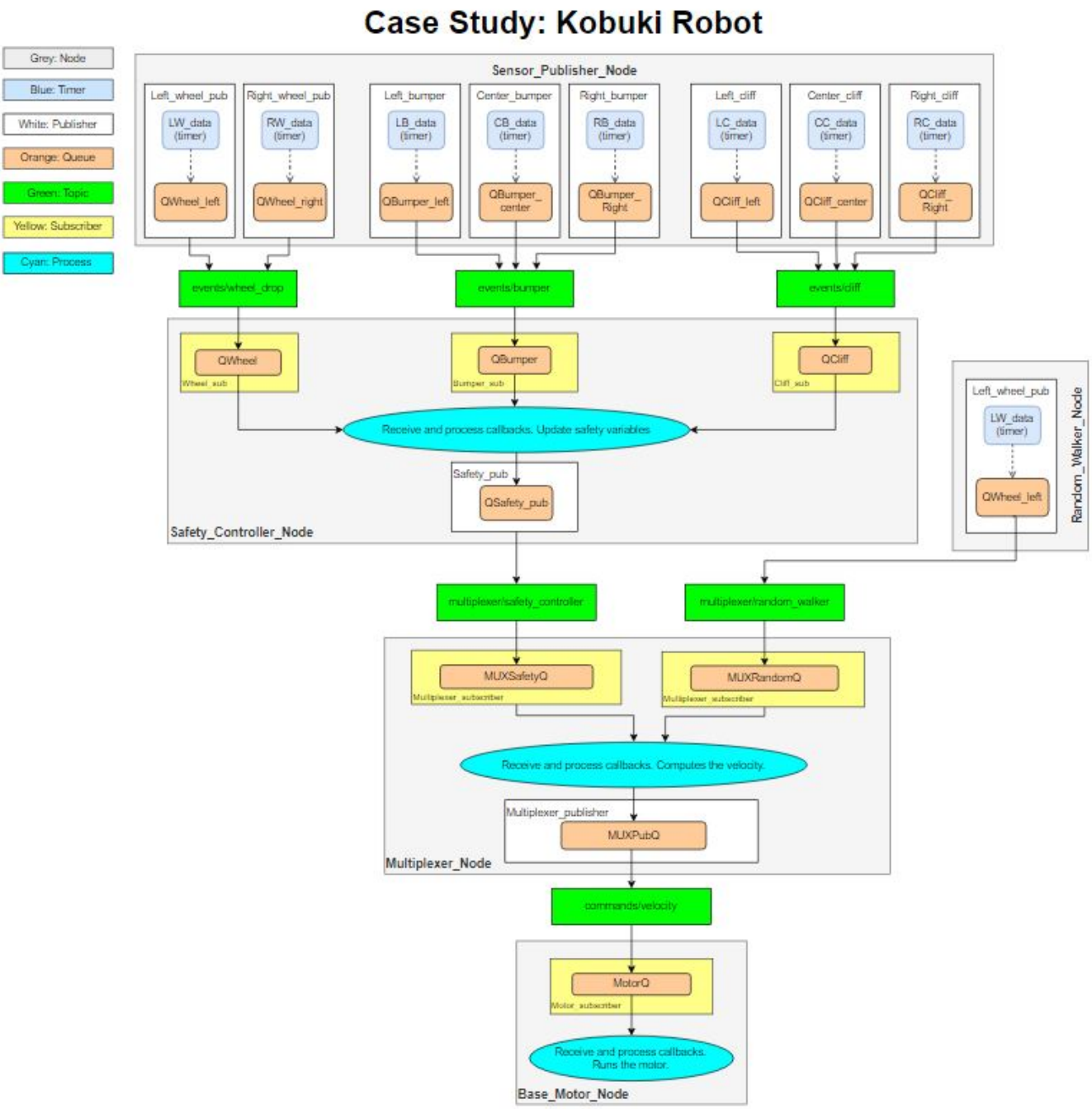

Figure 2: Modeling Kobuki Robot

We want to verify the following properties:

1. Bounded time properties (Lost messages from publishers): Every message from a publisher i is delivered to subscriber j within T_B time units.
2. Liveliness property: Every subscriber j must receive any message within every $< j_{min}$ time units (the subscriber queue changes within every j_{min} time units)
3. Active within deadline: Every subscriber j must contain any message within deadline j_d (subscriber queue not empty after j_d time units)
4. Publisher queue should not be full: If a publisher has queue with size n, the publisher queue should always have $< n$ messages.
5. Subscriber queue should be saturated: If a subscriber has queue with size n, the subscriber should always have n messages after t_s time units.
6. Maximum delay between messages: if a subscriber has message 'a' at timestamp t_a, and message 'b' at timestamp t_b, then $|t_a - t_b| \leq t_{max}$
7. Range of valuations (safety property): If a subscriber has message 'a', then $v_{min} < Val(a) <= v_{max}$