Optimal Network Routing Scenario

Important Note: While the scenario and task descriptions below use Python syntax for illustration, **please solve this challenge in the language specified by the recruiter or as per the requirements of the coding test**.

Scenario: In the intricate web of a cyberpunk city's data network, the swift and efficient routing of data is crucial. The city's network is a vast maze of routers and data links, each with its own latency. A unique feature of this network is the ability of certain nodes to compress data, halving the latency of outgoing links from these nodes. This feature can be used only once per data path.

As a skilled netrunner, you are tasked with developing an algorithm that navigates this complex network, finding the path between two nodes that minimizes the total latency, taking into consideration the data compression capabilities of specific routers.

Task: Implement a function, `find_minimum_latency_path`, that determines the path with the lowest total latency between a source and destination router, accounting for possible data compression at certain nodes.

Input:

- `graph`: A representation of the network as a map or dictionary, where each key is a router ID and its value is a list of tuples representing connected routers and the latency of the connection.
- `compression_nodes`: A list or array of router IDs where data compression can be applied.
- `source`: The ID of the source router.
- `destination`: The ID of the destination router.

Expected Output:

- An integer representing the minimum total latency from the source router to the destination router.

Constraints:

- Solve this challenge in the specified programming language without relying on external libraries for the core logic.
- The solution should efficiently handle a complex network graph with multiple routers and connections.

Example:

```python
# your implementation of the find_minimum_latency_path function
graph = {
    'A': [('B', 10), ('C', 20)],
    'B': [('D', 15)],
    'C': [('D', 30)],
    'D': []
}
compression_nodes = ['B', 'C']
source = 'A'
target = 'D'

min_latency = find_minimum_latency_path(graph, compression_nodes, source, target)
print(f"Minimum total latency: {min_latency}")
```

Evaluation Criteria:

- **Algorithmic Efficiency**: Your algorithm should efficiently navigate the network, optimizing for minimum latency.
- **Correctness**: The calculated minimum total latency must accurately reflect the most efficient path, considering data compression capabilities.
- **Code Quality**: The code should be clear, well-organized, and adhere to best coding practices.
- **Edge Case Handling**: The solution should robustly handle various network topologies, including large graphs and edge cases like unavailable paths.