MARVELL™

Graph feature arc proposal

DPDK 25.03

Nitin Saxena
Senior Principal Engineer

3 Jan 2025

Agenda

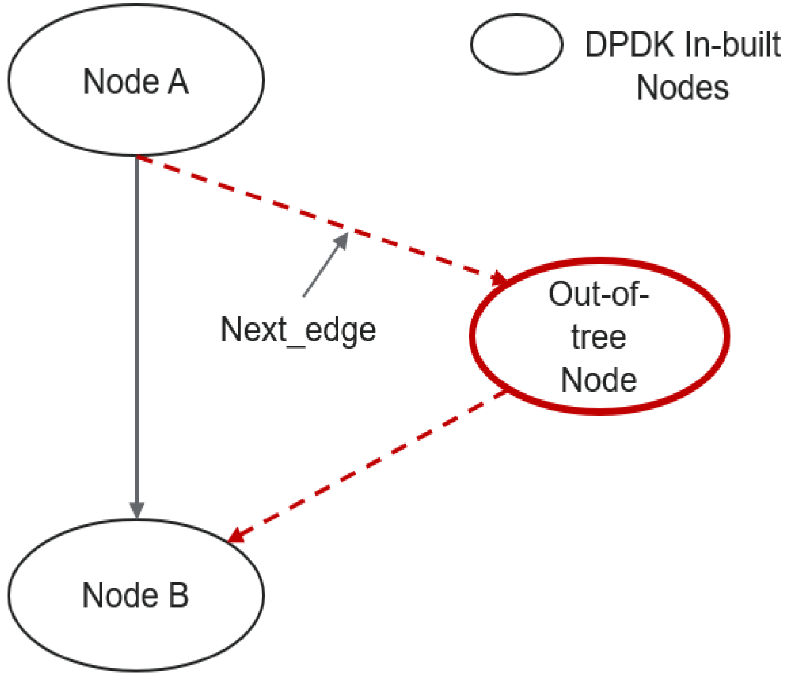
- Objectives of Feature arc
- Introduction to Feature arc
- How to use Feature arc
- Control plane and Fast path APIs

© 2024 Marvell confidential. All rights reserved.

Objectives of feature arc

1. Allow out-of-tree nodes hooked to DPDK in-built nodes and provide mechanism to steer packets toward it

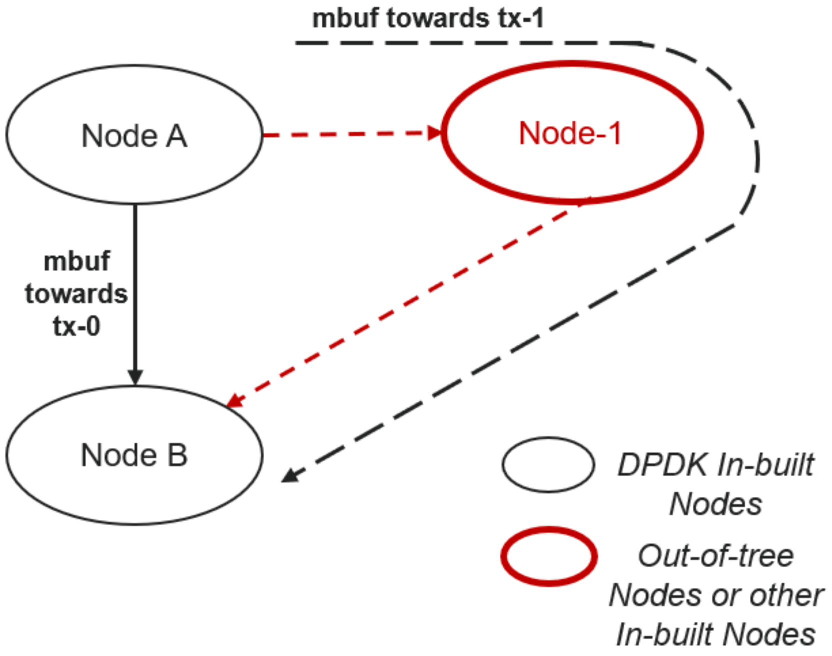
    - Provide hook points in sub-graphs created by in-built DPDK nodes
    - Packets should be steered to hooked nodes in a generic manner
    - Like out-of-tree nodes, *other in-built nodes* can also be hooked



Objectives of feature arc…

2. Provide mechanism to enable feature nodes per interface/ethdev

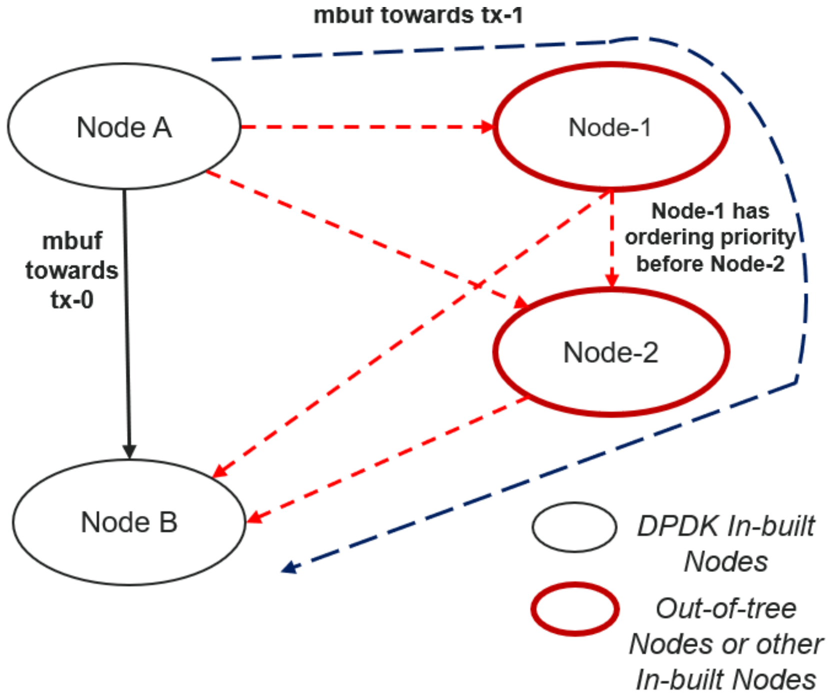
- Enable/disable of feature nodes on any interface should be allowed at runtime and not during graph creation
- Packets corresponding to an interface *“tx-1”* are steered to a feature node *“Node-1”* only when *“Node-1”* is enabled on *“tx-1”*.



Objectives of feature arc…

3. There can be more than one feature nodes enabled on an interface at runtime

   - Should have ordering sequence of packet traversal across multiple feature nodes


mbuf towards tx-1
Node A
Node-1
Node-1 has ordering priority before Node-2
mbuf towards tx-0
Node-2
Node B
DPDK In-built Nodes
Out-of-tree Nodes or other In-built Nodes

Objectives of feature arc…

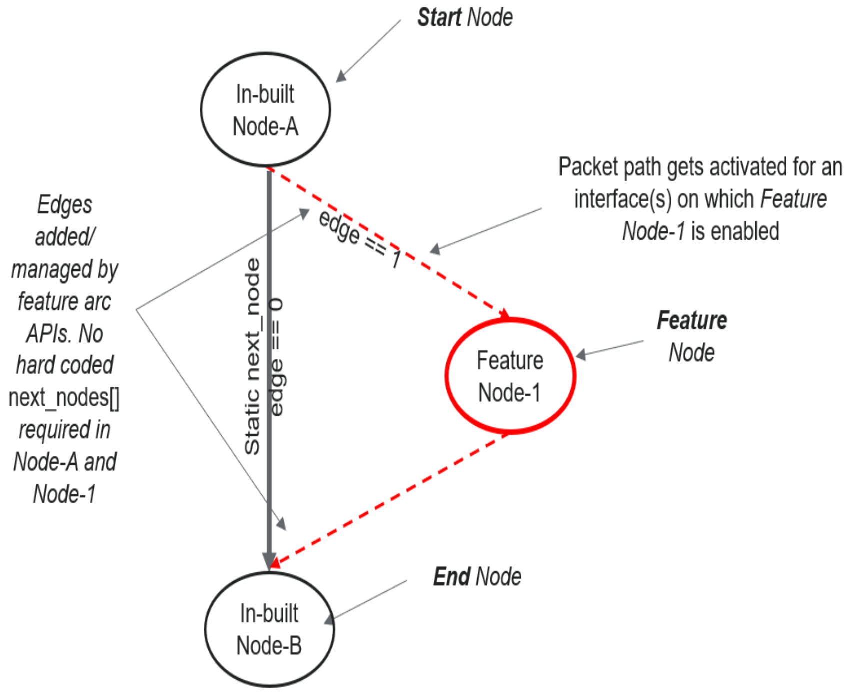
4. Faster data and control plane synchronization

   - Any feature disable/enable in control plane should not block worker cores
   - Feature disable should allow to destroy resources allocated by application during enabling a feature

© 2024 Marvell confidential. All rights reserved.

Introduction to Feature arc

1. An abstraction defining nodes packet path based on interface
2. Feature arc represents an ordered list of *feature nodes* with
   - A *start* node where:
     - Packets enters feature arc
   - An *end node* where:
     - Last *feature node* to create a *default* exit path for packets
   - One or more *feature nodes*
     - Added between start and end nodes
     - Ordered priority among feature nodes


Start Node
In-built Node-A
Packet path gets activated for an interface(s) on which Feature Node-1 is enabled
edge == 1
Edges added/ managed by feature arc APIs. No hard coded next_nodes[] required in Node-A and Node-1
Static next_node
edge == 0
Feature Node
Feature Node-1
End Node
In-built Node-B

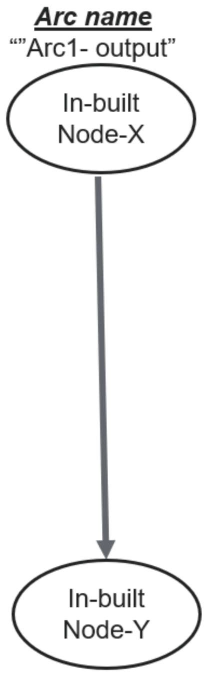
How to use Feature arc

- Feature arc and feature nodes registration
- Feature arc initialization
- Feature enable/disable in control plane
- Feature nodes fast path processing

© 2024 Marvell confidential. All rights reserved.

Feature arc registration

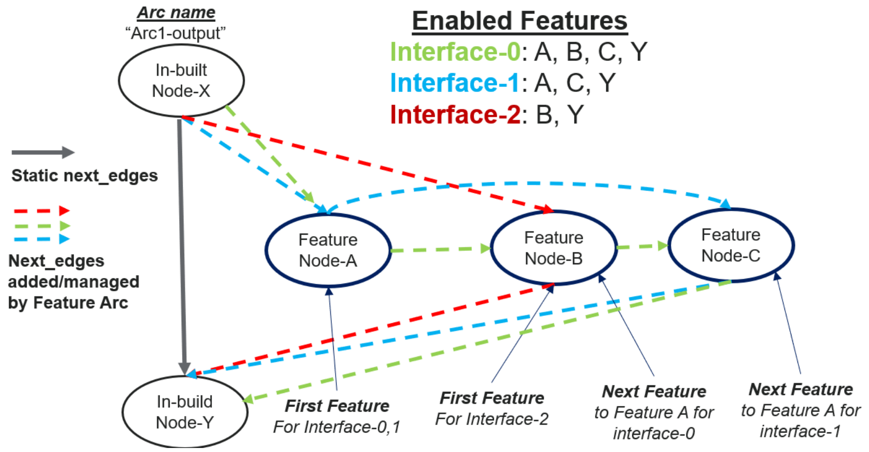
```
/* Node-X registration */
RTE_NODE_REGISTER (Node-X);

/* Node-Y registration */
RTE_NODE_REGISTER (Node-Y);

/* Node-Y feature initialization */
struct rte_graph_feature_register Node-Y-feature = {
    .feature_name = "Node-Y-feature",
    .arc_name = "Arc1-output",
     /* process() function called for node-Y" */
    .feature_process_fn = node_y_feat_process_fn(),
    .feature_node = &Node-Y,
};

/* Arc1 initialization */
struct rte_graph_feature_arc_register arc1 = {
    arc_name = "Arc1-output",

    /* Max number of interfaces supported */
    max_indexes = RTE_MAX_ETHPORTS,

    /* (struct rte_node_register *) */
    .start_node = &Node-X,
    /* process() function called for Node-X */
    .start_node_feature_process_fn = node_x_feature_process_fn(),

    /* end feature */
    .end_feature_node = &Node-Y-feature,
};

/* Feature arc registration */
RTE_GRAPH_FEATURE_ARC_REGISTER(arc1);
```



© 2024 Marvell confidential. All rights reserved.

Feature registration

```
RTE_NODE_REGISTER (node-A);
struct rte_graph_feature_register nodeA-feature = {
    .feature_name = "Node-A-feature",
    .arc_name = "Arc1-output",
    .feature_process_fn = nodeA_feature_process_fn(),
    .feature_node = &node-A,
};
/* Node-A feature registration */
RTE_GRAPH_FEATURE_REGISTER(nodeA-feature);

RTE_NODE_REGISTER (node-C);
struct rte_graph_feature_register nodeC-feature = {
    .feature_name = "Node-C-feature",
    .arc_name = "Arc1-output",
    .feature_process_fn = nodeC_feature_process_fn(),
    .feature_node = &node-C,
    .runs_after = "Node-A-feature",
    .notifier_cb = nodeC_notifier_cb(),
};
RTE_GRAPH_FEATURE_REGISTER(nodeC-feature);

RTE_NODE_REGISTER (node-B);
struct rte_graph_feature_register nodeB-feature = {
    .feature_name = "Node-B-feature",
    .arc_name = "Arc1-output",
    .feature_process_fn = nodeB_feature_process_fn(),
    .feature_node = &node-B,
    .runs_after = "Node-A-feature",
    .runs_before = "Node-C-feature",
};
/* Node-B feature registration */
RTE_GRAPH_FEATURE_REGISTER(nodeB-feature);
```



Arc name
"Arc1-output"
In-built Node-X
Enabled Features
Interface-0: A, B, C, Y
Interface-1: A, C, Y
Interface-2: B, Y
Static next_edges
Next_edges added/managed by Feature Arc
Feature Node-A
Feature Node-B
Feature Node-C
In-build Node-Y
First Feature For Interface-0,1
First Feature For Interface-2
Next Feature to Feature A for interface-0
Next Feature to Feature A for interface-1

Feature arc initialization

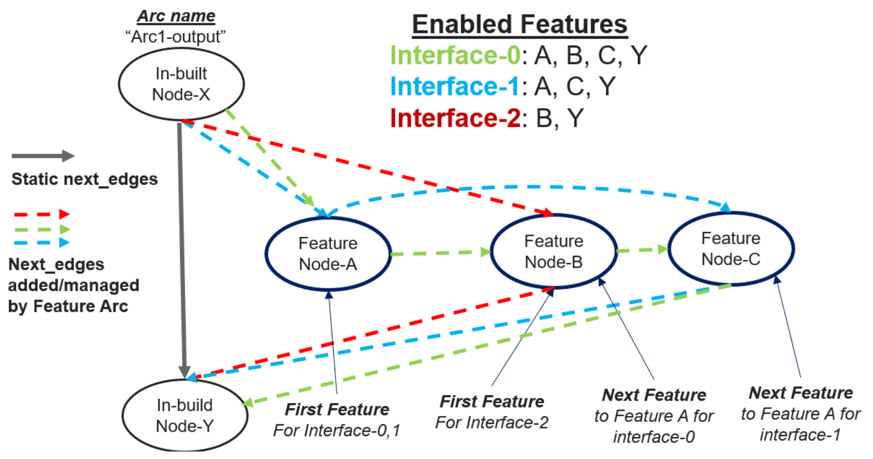
- Application should call rte_graph_feature_arc_init() before graph creation

- If not called, feature arc registrations has no effect.
- If possible, create RCU variable as well for worker core synchronization

```
static int worker_loop(void *cfg)
{
    struct rte_rcu_qsbr *qsbr = app_get_rcu_qsbr();
    struct rte_graph *graph = app_get_graph();

    rte_rcu_qsbr_thread_register(qsbr, rte_lcore_id());
    rte_rcu_qsbr_thread_online(qsbr, rte_lcore_id());

    while(1) {
        if (rte_get_main_lcore() == rte_lcore_id()) {
         /* main core calling
         * rte_graph_feature_enable()/rte_graph_feature_disable()
         */
        } else {
            rte_graph_walk(graph);
            rte_rcu_qsbr_quiescent(qsbr, rte_lcore_id());
        }
    }
}

void main()
{
    struct rte_graph_param graph_params;

    /* Initialize feature arc */
    rte_graph_feature_arc_init();

    /* Create rte_graph */
    rte_graph_create(&graph_params);

    rte_eal_mp_remote_launch(worker_loop, NULL, CALL_MAIN);
}
```

Feature enable/disable at runtime

```
{     struct rte_rcu_qsbr *qsbr = app_get_rcu_qsbr();
      rte_graph_feature_arc_t _arc;

      rte_graph_feature_arc_lookup_by_name("Arc1-output", &_arc);

     /* Enable first feature on each interface */
     rte_graph_feature_enable(_arc, 0 /* if0 */,
                                "Node-A-feature" /* feature name */,
                                100 /* cookie for (if0, Node-A) */,
                                rcu_qsbr);
     rte_graph_feature_enable(_arc, 1 /* if1 */,
                               "Node-A-feature" /* feature name */,
                               200 /* cookie for (if1, Node-A) */,
                               rcu_qsbr);
     rte_graph_feature_enable(_arc, 2 /* if2 */,
                                "Node-B-feature" /* feature name */,
                                300 /* cookie for (if2, Node-B) */,
                                rcu_qsbr);

     /* Disable feature on each interface */
     rte_graph_feature_disable(_arc, 0 /* if0 */,
                                 "Node-A-feature" /* feature name */,
                                 rcu_qsbr);
     rte_graph_feature_disable(_arc, 1 /* if1 */,
                                 "Node-A-feature" /* feature name */,
                                 rcu_qsbr);
     rte_graph_feature_disable(_arc, 2 /* if2 */,
                                 "Node-B-feature" /* feature name */,
                                 rcu_qsbr);
}
```

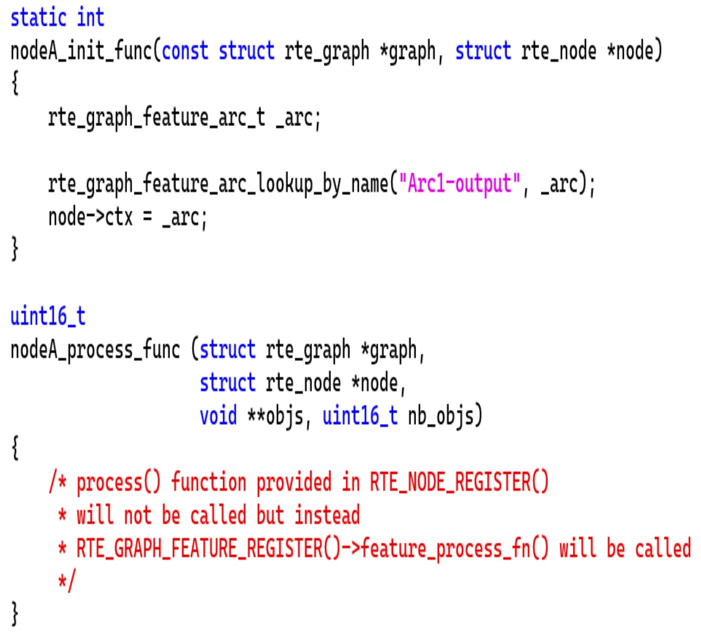


Fast path processing in *Start node* (Node-X)

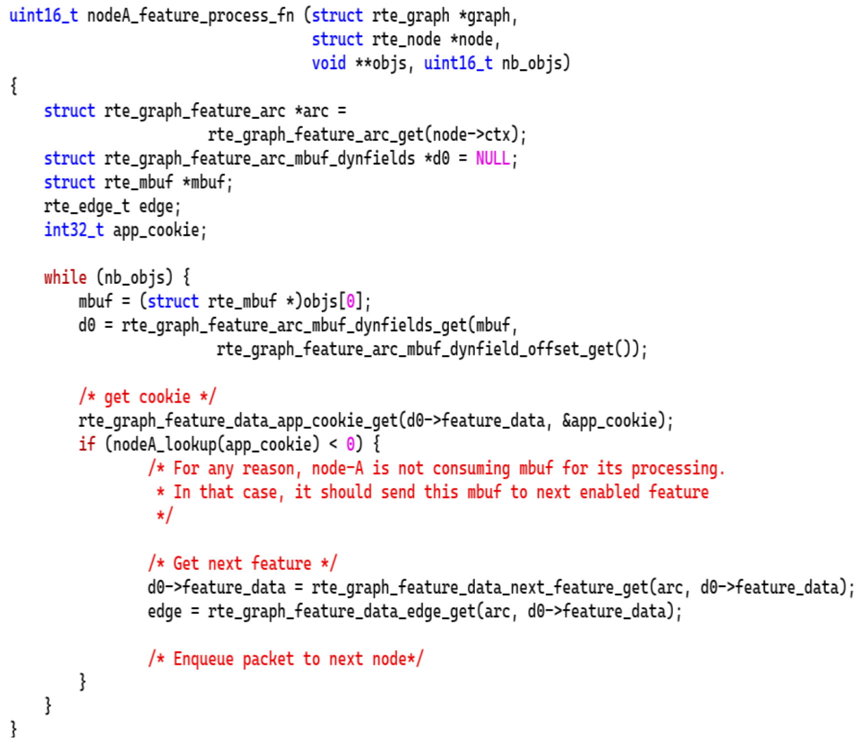
```
static int
nodeX_init_func(const struct rte_graph *graph, struct rte_node *node)
{
    rte_graph_feature_arc_t _arc;

    rte_graph_feature_arc_lookup_by_name("Arc1-output", _arc);
    node->ctx = _arc;
}
uint16_t nodex_process_fn()
{
    /* process() function provided in RTE_NODE_REGISTER() will not be called if
     * application calls rte_graph_feature_arc_init(), instead
     * RTE_GRAPH_FEATURE_ARC_REGISTER()->start_node_feature_process_fn() is called
     */
}
uint16_t
node_x_feature_process_fn (struct rte_graph *graph,
                           struct rte_node *node, void **objs,
                           uint16_t nb_objs)
{
    struct rte_graph_feature_arc *arc =
        rte_graph_feature_arc_get(node->ctx);

    if (unlikely(rte_graph_feature_arc_has_any_feature(arc))) {
        /* At least one feature is enabled on at least one interface */
        __nodeX_process(graph, node, objs, objs, nb_objs,
                arc, 1/* do arc processing */);
    } else {
        /* No feature is enabled on any interface */
        __nodeX_process(graph, node, objs, objs, nb_objs,
                NULL, 0 /* no arc processing */);
    }
}
```

```
uint16_t __nodeX_process(struct rte_graph *graph, struct rte_node *node,
              void **objs, uint16_t nb_objs,
              struct rte_graph_feature_arc *arc,
              const int do_arc_processing)
{
    struct rte_graph_feature_arc_mbuf_dynfields *d0 = NULL;
    rte_edge_t edge;

    while (nb_objs) {
        mbuf = (struct rte_mbuf *)objs[0];
        edge =  0; /* Node-Y added as .next_nodes[0] */

        if (do_arc_processing) {
            do = rte_graph_feature_arc_mbuf_dynfields_get(mbuf,
                         rte_graph_feature_arc_mbuf_dynfield_offset_get());
            /* Check if any feature enabled on mbuf->port */
            if (rte_graph_feature_data_first_feature_get(arc,
                                                         mbuf->port,
                                                         &d0->feature_data)) {
                /* First feature enabled on mbuf->port, get edge */
                rte_graph_feature_data_edge_get(d0->feature_data,
                                                &edge);
                /* enqueue mbuf with new edge */
            } else
                goto normal_processing;
        } else
            goto normal_processing;
normal_processing:
        /* Perform normal processing */
    }
}
```

Next Feature node processing (Node-A/B/C)



```c
static int
nodeA_init_func(const struct rte_graph *graph, struct rte_node *node)
{
    rte_graph_feature_arc_t _arc;

    rte_graph_feature_arc_lookup_by_name("Arc1-output", _arc);
    node->ctx = _arc;
}

uint16_t
nodeA_process_func (struct rte_graph *graph,
                    struct rte_node *node,
                    void **objs, uint16_t nb_objs)
{
    /* process() function provided in RTE_NODE_REGISTER()
     * will not be called but instead
     * RTE_GRAPH_FEATURE_REGISTER()->feature_process_fn() will be called
     */
}
```



```c
uint16_t nodeA_feature_process_fn (struct rte_graph *graph,
                                   struct rte_node *node,
                                   void **objs, uint16_t nb_objs)
{
    struct rte_graph_feature_arc *arc =
                   rte_graph_feature_arc_get(node->ctx);
    struct rte_graph_feature_arc_mbuf_dynfields *d0 = NULL;
    struct rte_mbuf *mbuf;
    rte_edge_t edge;
    int32_t app_cookie;

    while (nb_objs) {
        mbuf = (struct rte_mbuf *)objs[0];
        d0 = rte_graph_feature_arc_mbuf_dynfields_get(mbuf,
                    rte_graph_feature_arc_mbuf_dynfield_offset_get());

        /* get cookie */
        rte_graph_feature_data_app_cookie_get(d0->feature_data, &app_cookie);
        if (nodeA_lookup(app_cookie) < 0) {
                /* For any reason, node-A is not consuming mbuf for its processing.
                 * In that case, it should send this mbuf to next enabled feature
                 */

                /* Get next feature */
                d0->feature_data = rte_graph_feature_data_next_feature_get(arc, d0->feature_data);
                edge = rte_graph_feature_data_edge_get(arc, d0->feature_data);

                /* Enqueue packet to next node*/
        }
    }
}
```

MARVELL™

Essential technology, done right™

MARVELL™