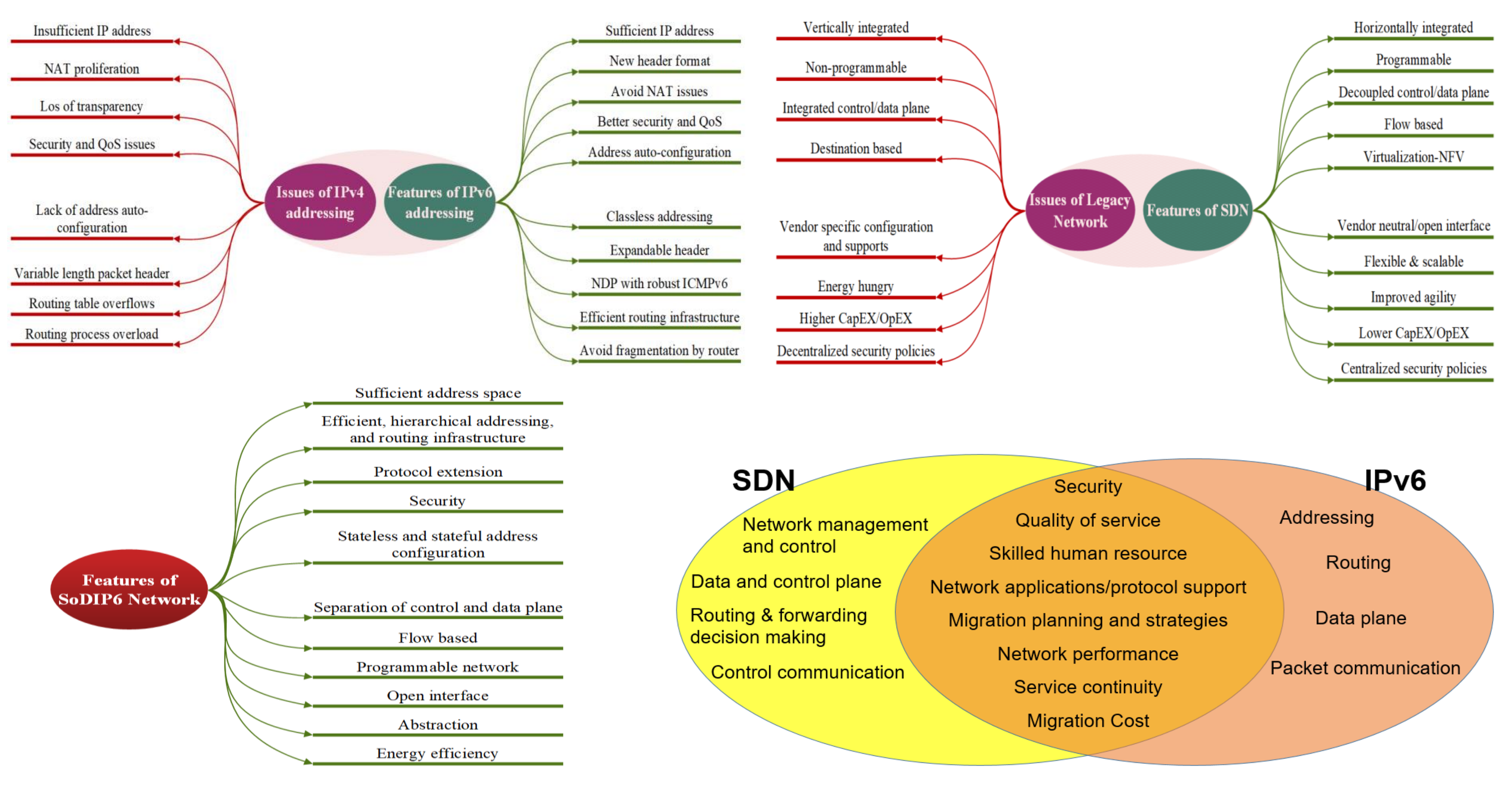
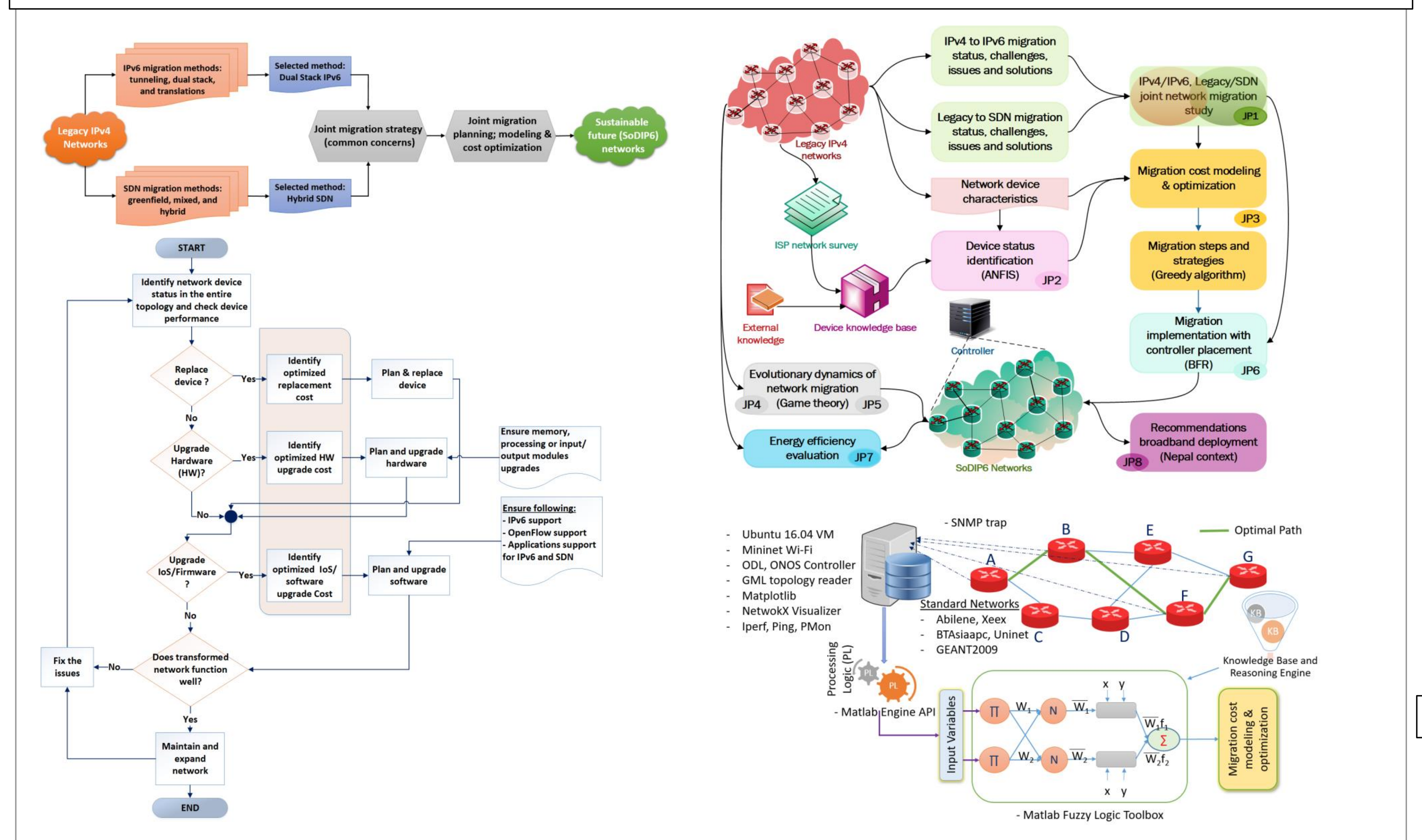
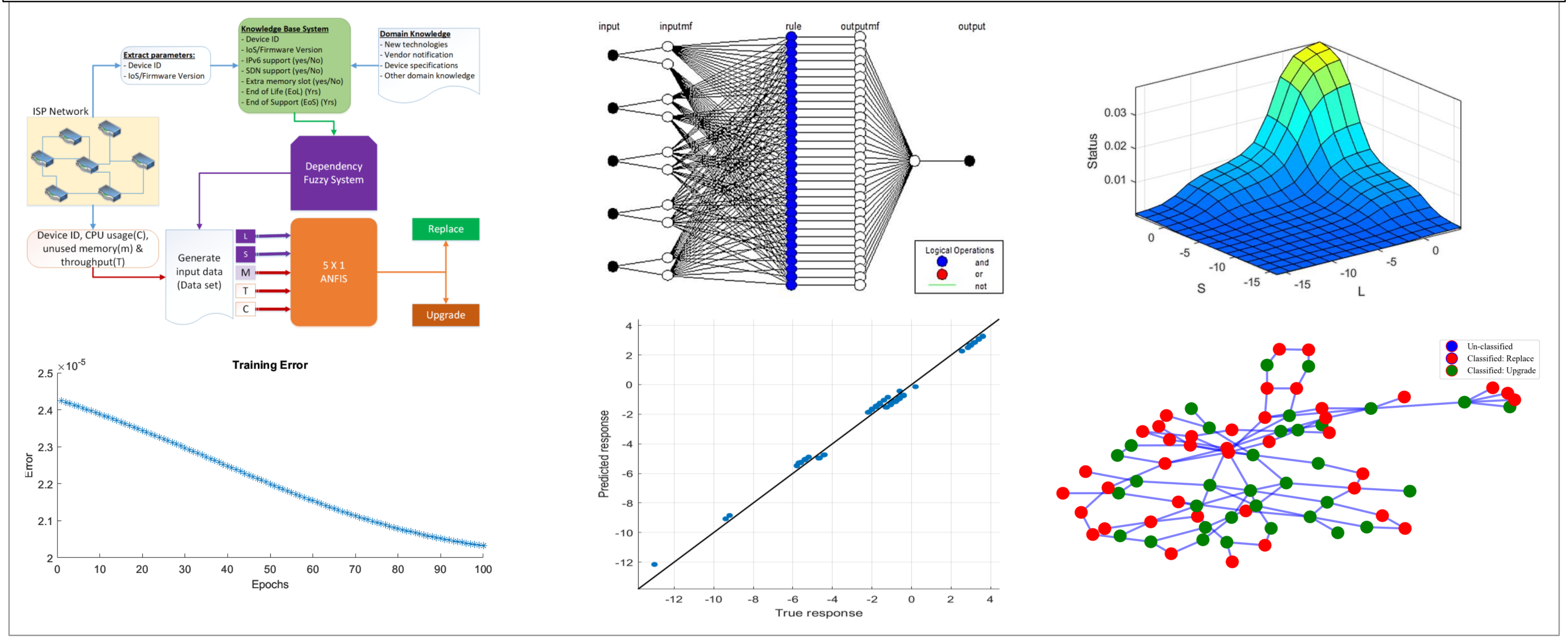
State-of-the-Art (RQ1, OB1) [1]



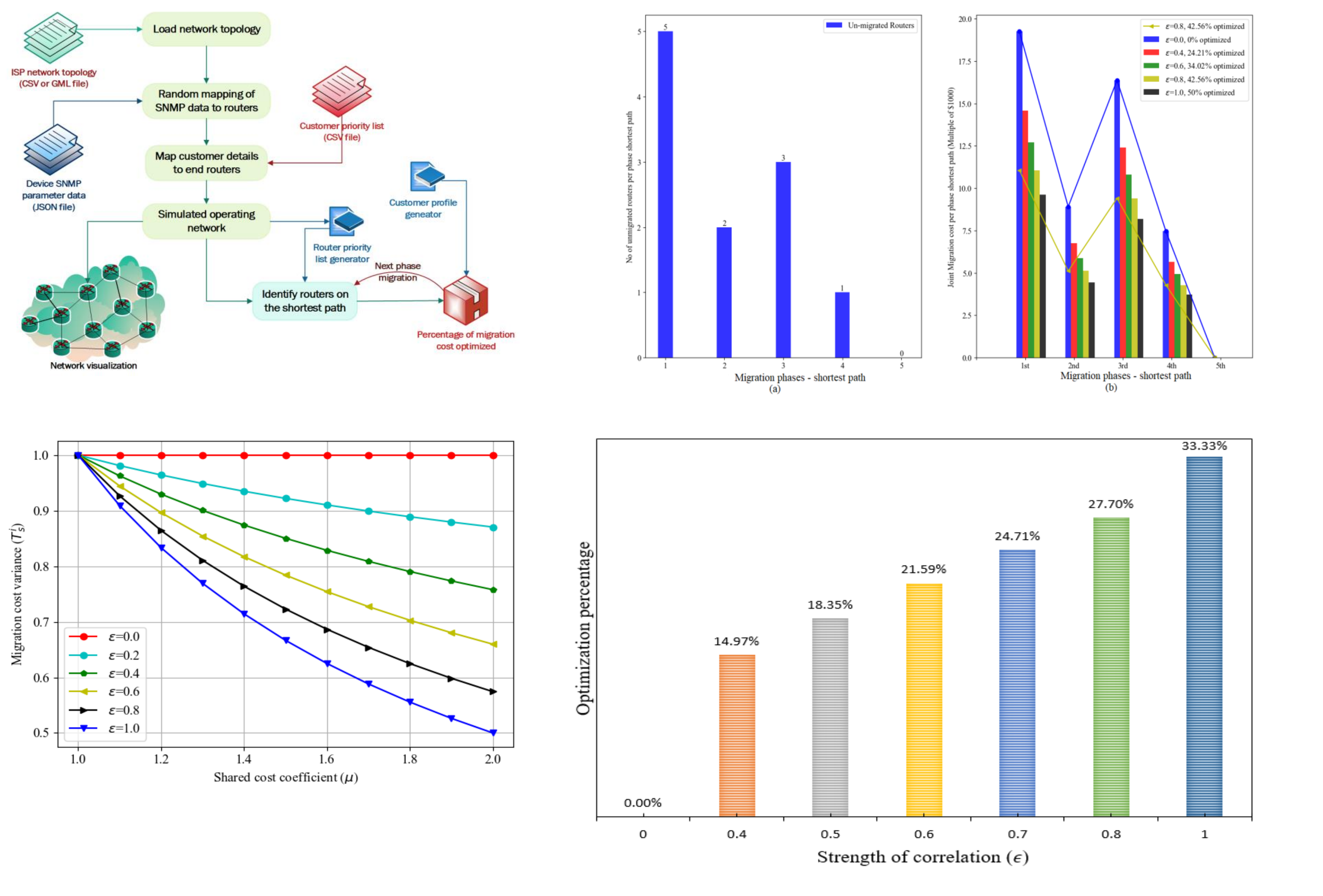
Methods/Framework [2]



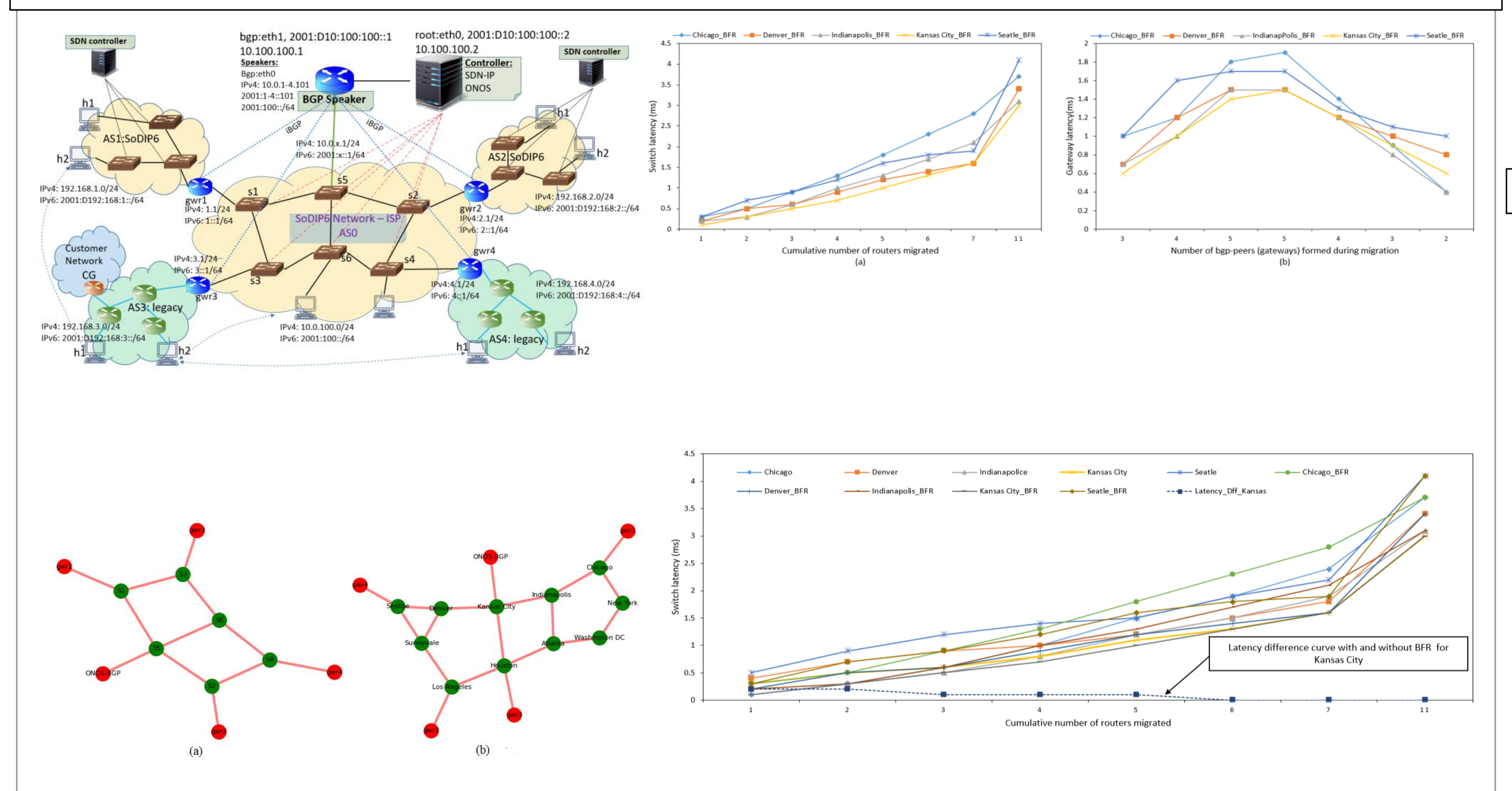
Intelligent Approach: Device Status Identification (RQ2, OB2) [3]



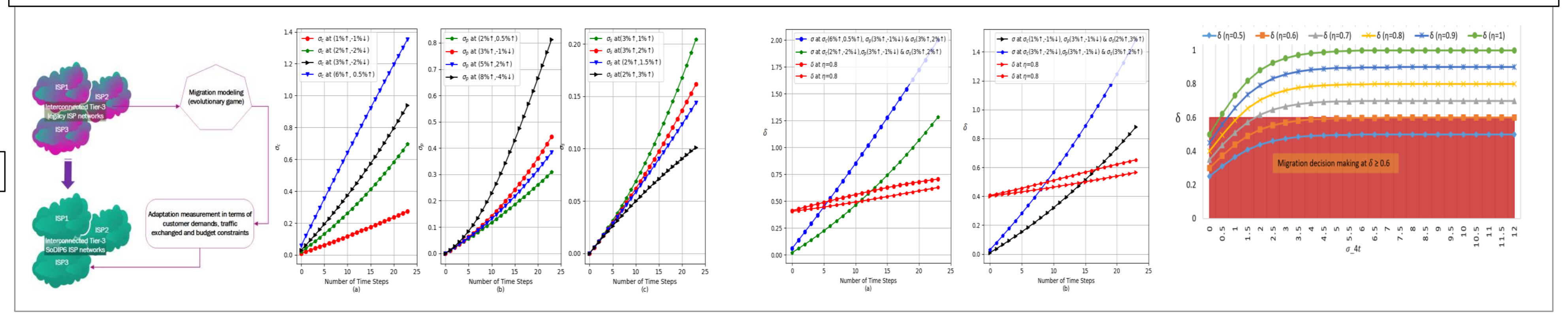
Joint Migration Modeling and Cost Optimization (RQ1/RQ2, OB1/OB2) [4]



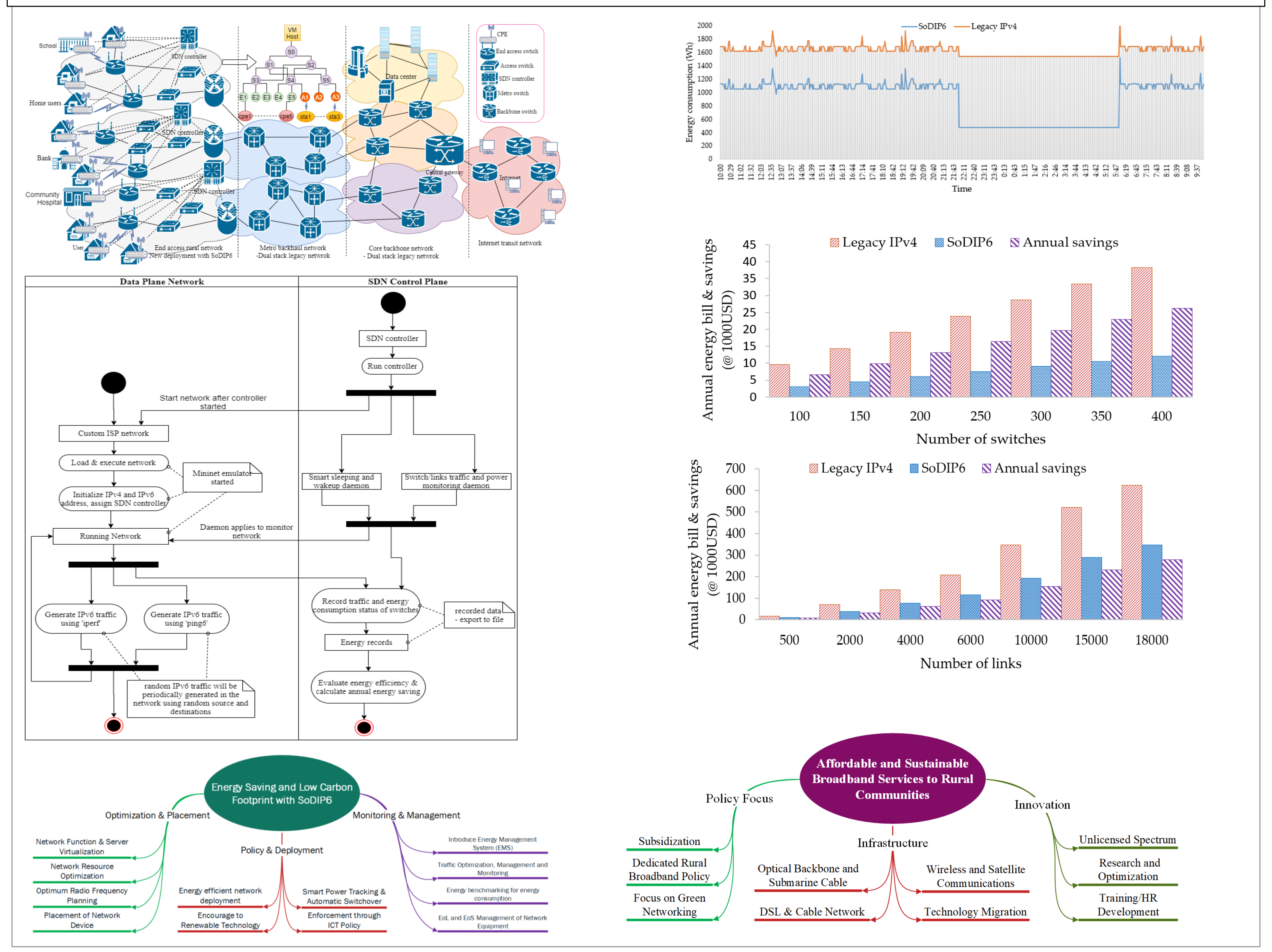
Hybrid SoDIP6 Network Implementation with Controller Placement (RQ2, OB2) [6]



Evolutionary Gaming Approach for Multi-ISP Network Migration Modeling (RQ2, OB2) [5]



Sustainable SoDIP6 Network, Energy Evaluation (RQ3, OB3) [7]



Algorithms

Algorithm 3.1: ANFIS implementation for device status identification

3

```
Function DFS(v):
    Input: CD ← today(), E ← 0, [EoS, EoL, M_E, UV] ← KB(v)
    // CD: Current Date, UV: upgraded IOS, get details of router 'v' from KB.
    if Router supports IOS upgrades then
        NV ← UV                          // NV: New IOS supports SoDIP6.
    else
        NV ← KB(new_IOS)                 // get new IOS information from the KB.
    if UV supports SoDIP6 then
        IO ← 1
    else
        IO ← 0
    if Device has extra memory slot then
        E ← M_E
    L ← (CD − EoL)
    S ← (EoS − CD)
    return L, S, E, IO
Function Main:
    Input: G ← (V, E)        // vertices 'V' and edges 'E' to network graph G.
    for v in G do
        [L, S, M_E, IO] ← DFS(v)                    // function call.
        T, C, m ← Snmp(v)                           // SNMP agent of node v.
        M = (m + M_E) · IO                          // calculate total available memory.
        status(v) ← Anfis(L, S, M, T, C)            // API to call ANFIS from MATLAB.
    plot(G)                                         // Visualize the network graph G.
```

Algorithm 3.2: Migration to SoDIP6 network based on shortest path routing and customer priority

4

```
Input: G ← (V, E), τ ← NULL
/* vertices 'V' and links 'E' in ISP network graph G.  total cost of
   migration at the beginning is set to NULL.                              */
Input: B^e_s ← Budget
/* per phase total budget of migration in the optimal path from end router to
   gateway router S.                                                       */
σ^e_k ← [α, β, θ, γ, δ, σ] /* cost metrics defined in Table 3.3 for estimation and
   optimization calculation.                                               */
∀e ∈ E_v : δ_c                      // customer priority value (highest).
∀N ∈ P_e : N /* number of un-migrated paths in the set of all available paths
   (P_e).                                                                  */
for i ← 1 to N do
        // find optimal path between customer end router and gateway router
    if optimal(P_i) then
        p ← p_i           // retrieve optimal path p from set of alternate paths.
n ← number of unmigrated nodes in the optimal path p
∀μ^e_p ← NULL            // initialize migration cost to NULL in the optimal path.
for k ← 1 to n do
        // find migration cost of each un-migrated router in optimal path p.
    if status(ρ^e_k) is 0 then
        μ^e_p += replace_cost(σ^e_k)
    else
        upgrade_cost(σ^e_k)
τ += μ^e_k                        // total cost of migration in optimal path p.
if μ^e_p < B^e_s then
    migrate(n) /* if phased budget B^e is sufficient, then migrate all routers
       (n) in the optimal path p.                                          */
else
    migrate(m)        // migrate m (<n) number of routers in the optimal path.
Repeat with next phase migration budget with next priority customer until all
   network routers are migrated.
Return T, the total optimized migration cost of whole ISP network.
```

Algorithm 3.4: Network migration and the controller placement using BFR

6

```
Data: Network topology net_data, Customer gateway records cg_data.
/* Load network data (net_data) also attached customer gateways (CGs).    */
Input: G ← net_data, G ← cg_data, foreign gateway FG,
       controller_capacity ← 1.1
LSSP, sp_lists ← get_LSSP(G, FG)    // Algorithm 3.3 returns LSSP and their
   routers
G ← CG_data                        // load customer gateways and attach to G
R ← median_router(LSSP)    // get median router from LSSP - equation (3.18)
RSS ← migrate(R)  // migrates legacy router R into a SoDIP6-capable switch.
add_link(rss, bgp)                                  // attaches RSS to bgp.
for sp_routers in sp_lists do
    R ← median_router(sp_routers)                 // get median router per SP.
    BFT_list ← BFTravel(sp_routers, R) /* function to implement BFT,
       returns the ordered list into BFT_list.                             */
    for router in BFT_list do
        if router is not SoDIP6 capable then
            migrate(router)    // legacy router becomes a SoDIP6-capable switch.
            sdn_switches ← get_switches(G)    // get SoDIP6-capable switches.
            sw_latency ← 0.0                  // initialize switch latency.
            for s in sdn_switches do
                sw_latency += Δ_{s,c} /* propagation latency from switch "s" to
                   controller "c" (i.e., bgp) is cumulatively added.        */
                store_switch_latency(sw_file_name) /* this stores the switch
                   numbers and latency records in a data file.              */
            switch_gws ← find_gateways(G)            // Ref.  equation (3.22)
            gw_latency ← 0.0                      // initialize gateway latency.
            for g in switch_gws do
                gw_latency += Δ_{g,c} /* propagation latency from gateway "g" to
                   controller "c" i.e.,bgp is cumulatively added           */
                store_gateway_latency(gw_file_name) /* records gateway
                   numbers & latency.                                      */
    controller_load ← control_switch_gateway(s, g) /* function to record total
       load to the controller.                                             */
    store_control_load(control_load_file)        // records controller load.
    if controller_load > controller_capacity then
        notify_controller_addition()     // warning at controller overloading.
    for R in G do
        migrate(R)     // migrates all stub routers to SoDIP6-capable switches.
    record_latency(S, gw)     // records latency of stub routers and gateways.
```

Algorithm 3.6: SoDIP6 network traffic and power monitoring

7

```
Input: G ← (V, E)
                   // vertices 'V' and links 'E' in network graph G.
run_network(G)     // initialize network, assign IPv6 address, set remote
   controller.
src_list = random(V) /* V belongs to CPEs.  This identify source hosts for
   random ipv6 traffic generation.                                         */
dst_list = random(V) /* V belongs to CPEs.  This identify destination hosts
   to send IPv6 traffic generated form source hosts.                       */
while running_network(G) do
    @every 2 minutes, Generate_IPv6_iperf(src_list) /* generate IPv6 traffic
       every 2 minutes using iperf.                                        */
    @every 3 minutes, Generate_IPv6_ping6(src_list) /* generate IPv6
       traffic every 3 minutes using ping6.                                */
    @every 2 to 3 minutes, for node in V do
        if node is active then
            record_active_power(node) /* record active power of each switch
               with average traffic volume passing through it.             */
        else
            record idle/sleep mode power of node
        if no_traffic(node) then
            sleep_node(node)/* if node is idle then the node is signaled to
               enter into sleep mode.                                      */
    @every 2 to 3 minutes, /* record power of links, record power of active
       and sleep switch on every 2-3 minutes.                              */
    for link in E do
        if link is active then
            record_link_power(link)              // record power of each link.
    @10PM to 6AM, sleep_all_node() /* sleep the node during the night from
       10 PM to 6AM.                                                       */
    @6AM, wakeup_all_node()                 // wake-up nodes if in sleep mode.
```

Algorithm 3.5: Evolutionary process of multi-ISP network migration

5

```
Input: G1 ← list of legacy ISPs, G2 ← list of SoDIP6 capable ISPs, η ← 0.7,
       {σ_{4t−2}, σ_{4t−1}, σ_{4t}} ← NULL
1  @every 3 months,
2  for n ∈ G1 do
3      γ1, γ2 ← calculate utilities                      // Ref.  equation 3.30.
4      σ^n_{4t} ← calculate value of σ_{4t}              // Ref.  equation 3.27.
5      δ^n_{4t} ← calculate value of δ                   // Ref.  equation 3.28.
6      if (σ^n_{4t−2} < σ^n_{4t−1} < σ^n_{4t}) and (δ^n_{4t} ≥ 0.6) then
7          proceed_migration(n)     // Migrate this ISP 'n' using Algorithm 3.4.
8          G1.remove(n)                      // remove migrated ISP form n1 list.
9          G2.add(n)                           // add migrated ISP into n2 list.
10     σ^n_{4t−2} = σ^n_{4t−1}
11     σ^n_{4t−1} = σ^n_{4t}
12 Repeat from step-1 to measure strength of unmigrated ISPs for next phase
   migration.
```

Algorithm 3.3: LSSP identification approach in an AS from network G(V,E).

6

```
Function get_LSSP(G, FG):
    gw ← extract_gateway(G, AS_num) /* extract the list of gateway
       routers from the provided AS (AS_num) of network G.                 */
    for CG in gw do
        sp_lists ← optimal_path(G, CG, FG); /* this function identifies the
           number of routers in the shortest path, which are then appended to
           the shortest path list (sp_lists).                              */
    LSSP ← path_highest_count(sp_list)
    /* this function sorts the sp_lists and returns the highest count shortest
       path as LSSP.                                                       */
    return LSSP, sp_lists
```

Research Questions (RQ) and Objectives (OB)

RQ1: What are the challenges, risks, and strategies for the migration of legacy IPv4 networks into SoDIP6 networks?

RQ2: What is/are the intelligent approach(s) for joint network migration planning and cost optimization?

RQ3: What are the roles of SoDIP6 networks for future sustainability of service providers and the society?

OB1: Analyze the joint approach of legacy IPv4 network migration to SDN enabled IPv6 network.

OB2: Develop intelligent approach for migration cost optimization and modeling of service provider legacy network migration to SoDIP6 network.

OB3: Evaluate the features of SoDIP6 network in terms of energy efficiency for service provider sustainability.

Dissertation Title:

Analysis, Modeling, and Evaluation of Service Provider Legacy Network Migration to Software-Defined IPv6 Network

Important Dates, Degree, and Scholarship:

PhD Enrollment: 17Th Nov., 2017
DRC Date: 26th Jan., 2021
IERC Date: 20th Aug., 2021

Degree and Department/Campus/University

Computer Engineering PhD Program
Dept. of Electronics and Computer Engineering
Institute of Engineering, Pulchowk Campus
Tribhuvan University, Nepal

Scholarhip

NTNU Norwegian University of Science and Technology
EnPe

PhD Candidate	Advisors		Visiting Advisors	
				
B. R. Dawadi	Prof. Dr. S. R. Joshi	Prof. Dr. D.B. Rawat	Prof. Dr. P. Manzoni	Prof. Dr. M. Keitsch