APPENDIX [A]

m=2;
n=2;
k=2;
a=(n^m)^-1
s2= nchoosek(n,k);
summ=0;
for i=0:k-1;
    summ=summ+((-1)^i)*((k-i)^m)*nchoosek(k,i)
end

Prop_of_K=a*summ*s2 \hspace{1em} \% Probability of a HA engaged with at least on mobile node.

APPENDIX [B]

m=10;
n=10;
a=(n^m)^-1
s2=0;
for k=1:min(n,m);
    s2=s2+(nchoosek(n,k))*k;
    summ=0;
    for i=0:k-1;
        summ=summ+((-1)^i)*((k-i)^m)*nchoosek(k,i)
    end
end
kHAT_mean=a*summ*s2
APPENDIX [C]

dw = rand(1,20)  % Average time of latency in wired network
dn = rand(1,20)  % Average time of latency between nested networks
Beta = rand(1,20) % Average time effect comes from NEMO movement speed
m = 100;  % Total number of nodes that they want multicast service
L = 2;  % Nesting level of MR counting from AR
n = 50  % Total number of HAs, we assume n = 2^(l-1)
        % L = 4.32 ~ = 4 level in max let us choose L = 2
khat = 2
l = 0;
b = mean(Beta)
    for l = 1:L
        epsyof_BT = (khat.*dw + m.*(dw+Beta)*(1-(khat)^l)) + m*L.*(dw+Beta)
        epsyof_RS = (dw + L.*(dw+Beta)+L.*(dn+Beta))
        epsyof_AB = (2.*dw + L.*dn + b)
    end
%0-1 Knapsack Problem in Matlab
%algorit.com

clear; close all; clc;
%% input - feel free to change it
knapsack = 10; %size of knapsack
weight = [12 8 5 5]; %weight of items
worth = [40 35 17 19]; %worth of items

%% knapsack
%add the "zero item" to allow single items to be taken in the loop
weight = [weight 0];
worth = [worth 0];
s = length(weight);
solutions = zeros(1,s+2);
for ii=1:s % available items
    items(ii,:) = zeros(1,s+2);
    items(ii,ii) =1;
    items(ii,s+1) = worth(ii);
    items(ii,s+2) = weight(ii);
end
solutions = items;

proposedOptimal = [];
for kz = 1:knapsack %all sized from minimal weight to knapsack size
    %pick a combination of solution as the new solution
    %(double loop could be improved)
    for jj = 1:size(solutions,1) %consider all solutions
        for kk = 1:size(solutions,1) %in pairs
            if ~sum(items(jj,1:end-2).*items(kk,1:end-2))
                %never take an item twice
                %make sure weight sum does not exceed knapsack
                if (solutions(jj,end) + solutions(kk,end))<=kz
                    %propose an optimal solution
                    proposedOptimal(end+1,:) = solutions(jj,:) + solutions(kk,:);
                end
            end
        end
    end
end
if ~isempty(proposedOptimal) %if solution were proposed
    %pick THE optimal solution from proposed solutions
    temp = find(proposedOptimal(:,end-1) ==
                max(proposedOptimal(:,end-1))); temp = temp(1);
    %remember that solution in the array
    solutions(end+1,:) = proposedOptimal(temp,:);
end
proposedOptimal = [];
end
%% output - print results
fprintf('Optimal Items:
');
for ii=1:size(solutions(end,:),2)-3 %ignore zero items
    if solutions(end,ii)==1 %if the item is taken
        jj = solutions(end,ii);
        fprintf('Take item number %d with weight %d which is worth %d.
',ii, weight(ii), worth(ii));
    end
end
fprintf('Total weight of items %d, total worth %d
\n
',solutions(end,end), solutions(end,end-1) );
APPENDIX [E]

```matlab
%% Knapsack demonstration
% Integer weights of items
N = 10;
weights = randint(1,N,[1 1000])
%%
% Values of the items (don't have to be integers)
values = randint(1,N,[1 100])

%% Solve the knapsack problem
% Call the provided m-file
capacity = 5000;
[best amount] = knapsack(weights, values, capacity);
best;
items = find(amount)
%%
% Check that the result matches the contraint and the best value
sumwate=sum(weights(items))
sumvalue=sum(values(items))

%%KNAPSACK Solves the 0-1 knapsack problem for positive integer weights
% [BEST AMOUNT] = KNAPSACK(WEIGHTS, VALUES, CONSTRAINT)
% WEIGHTS : The weight of every item (1-by-N)
% VALUES : The value of every item (1-by-N)
% CONSTRAINT : The weight constraint of the knapsack (scalar)
% BEST : Value of best possible knapsack (scalar)
% AMOUNT : 1-by-N vector specifying the amount to use of each
% item (0 or 1)
%
% EXAMPLE :
% weights = [1 1 1 2 2 3];
% values = [1 1 2 3 1 3 5];
% [best amount] = KNAPSACK(weights, values, 7)
% best =
% 13
% amount =
% 0 0 1 1 0 1 1
%
%
% Copyright 2009 Petter Strandmark
```
function [best amount] = knapsack(weights, values, W)
    if ~all(is_positive_integer(weights)) || ...
        ~is_positive_integer(W)
        error('Weights must be positive integers');
    end
    %We work in one dimension
    [M N] = size(weights);
    weights = weights(:);
    values = values(:);
    if numel(weights) ~= numel(values)
        error('The size of weights must match the size of values');
    end
    if numel(W) > 1
        error('Only one constraint allowed');
    end
    % Solve the problem
    % Note that A would ideally be indexed from A(0..N,0..W) but MATLAB
    % does not allow this.
    A = zeros(length(weights)+1,W+1);
    % A(j+1,Y+1) means the value of the best knapsack with capacity Y
    % using % the first j items.
    for j = 1:length(weights)
        for Y = 1:W
            if weights(j) > Y
                A(j+1,Y+1) = A(j,Y+1);
            else
                A(j+1,Y+1) = ...
                max( A(j,Y+1), values(j) + A(j,Y-weights(j)+1));
            end
        end
    end
    best = A(end,end);

    %Now backtrack
    amount = zeros(length(weights),1);
    a = best;
    j = length(weights);
    Y = W;
    while a > 0
        while A(j+1,Y+1) == a
            j = j - 1;
        end
        j = j + 1; %This item has to be in the knapsack
        amount(j) = 1;
        Y = Y - weights(j);
        j = j - 1;
        a = A(j+1,Y+1);
    end
amount = reshape(amount,M,N);
end

function yn = is_positive_integer(X)
    yn = X>0 & floor(X)==X;
end
APPENDIX [F]

\[
\begin{align*}
\text{for } w & \text{ from } 0 \text{ to } W \text{ do} \\
& m[0, w] := 0 \\
\text{end for} \\

\text{for } i & \text{ from } 1 \text{ to } n \text{ do} \\
& \text{for } j \text{ from } 0 \text{ to } W \text{ do} \\
& \quad \text{if } j \geq w[i] \text{ then} \\
& \quad \quad m[i, j] := \max(m[i-1, j], m[i-1, j-w[i]] + v[i]) \\
& \quad \quad \text{else} \\
& \quad \quad m[i, j] := m[i-1, j] \\
& \quad \text{end if} \\
& \text{end for} \\
\text{end for}
\end{align*}
\]