Wireless Networks

Distributed Joint Flow-Radio and Channel Assignment Using Partially Overlapping Channels in Multi-Radio Wireless Mesh Networks

Online Resource 1
Supplementary Algorithms for DFRCA

Alper Rifat Ulucinar · Ibrahim Korpeoglu

This work is supported in part by the European Union FP7 Programme with project FIRESENSE, FP7-ENV-2009-1-244088-FIRESENSE.

Alper Rifat Ulucinar
Department of Computer Engineering, Bilkent University, 06800, Ankara, Turkey
Tel.: +90-312-2902599
Fax: +90-312-2664047
E-mail: ulucinar@cs.bilkent.edu.tr

Ibrahim Korpeoglu
Department of Computer Engineering, Bilkent University, 06800, Ankara, Turkey
E-mail: korpe@cs.bilkent.edu.tr

Appendix A

Algorithm A.1 Flow Balancer on n_i
Input: C
Output: C

procedure FLOWBALANCER(C)

$$\Omega \leftarrow \left\{ (i,k) : \; k \in [1,D] \wedge \sum_{\forall (j,l): f_{i,j,k,l} \in F} |f_{i,j,k,l}| + \sum_{\forall (j,l): f_{j,i,l,k} \in F} |f_{j,i,l,k}| > \rho_{max} \right\}$$

▷ Ω is the set of overflown radios

$$\Upsilon \leftarrow \left\{ (i,k) : \; k \in [1,D] \wedge \sum_{\forall (j,l): f_{i,j,k,l} \in F} |f_{i,j,k,l}| + \sum_{\forall (j,l): f_{j,i,l,k} \in F} |f_{j,i,l,k}| < \rho_{max} \right\}$$

▷ Υ is the set of underflown radios, i.e. radios with positive residual capacity

  for all $(i,k) \in \Omega$ **do**
    $E_k \leftarrow \sum_{\forall (j,l): f_{i,j,k,l} \in F} |f_{i,j,k,l}| + \sum_{\forall (j,l): f_{j,i,l,k} \in F} |f_{j,i,l,k}| - \rho_{max}$
    for all $f_{R_k} : f_{i,j,k,l} \in F \vee f_{j,i,l,k} \in F$ **do** ▷ For each flow on (i,k)
      for all $(i,k') \in \Upsilon$ **do**
        for all $f_{R_{k'}} : f_{i,j,k',l} \in F \vee f_{j,i,l,k'} \in F$ **do** ▷ For each flow on (i,k')
          $\delta \leftarrow \left| f_{R_k} \right| - \left| f_{R_{k'}} \right|$
          $\Delta_{R_{k'}} \leftarrow \rho_{max} - \sum_{\forall (j,l): f_{i,j,k',l} \in F} \left| f_{i,j,k',l} \right| - \sum_{\forall (j,l): f_{j,i,l,k'} \in F} \left| f_{j,i,l,k'} \right|$
          if $\delta > 0 \wedge E_k - \delta \geq 0 \wedge \Delta_{R_{k'}} - \delta \geq 0$ **then**
            $E_k \leftarrow E_k - \delta$
            $C[f_{R_{k'}}] \leftarrow k$
            $C[f_{R_k}] \leftarrow k'$
          end if
        end for
      end for
    end for
    if $E_k > 0$ **then**
      `exit` ▷ Heuristic failed to balance
    end if
  end for
end procedure

Algorithm A.2 Computation of the Set of k-neighborhood Subgraphs on n_i
Input: (N_k, F_k)
Output: Ψ, the set of subgraphs

procedure FindSubgraphs((N_k, F_k))
  $\Psi \leftarrow \varnothing$ ▷ Ψ is the set of subgraphs. The number of subgraphs will be $|\Psi|$
  for all $f_{i',j',k',l'} \in F_k$ **do**
    if $i' \notin H_1 \wedge i' \notin H_k \wedge i' \neq i$ **then**
      `continue`
    end if
    $R_t \leftarrow (i', k')$
    $R_r \leftarrow (j', l')$
    $R_o \leftarrow (-1, -1)$
    for all $\psi_1 \in \Psi$ **do**
      if $R_t \in \psi_1$ **then**
        $R_o \leftarrow R_r$
      else if $R_r \in \psi_1$ **then**
        $R_o \leftarrow R_t$
      end if
      if $R_o \neq (-1, -1)$ **then**
        $\psi_1 \leftarrow \psi_1 \cup \{R_t, R_r\}$
        for all $\psi_2 \in \Psi$ **do**
          if $\psi_2 \neq \psi_1 \wedge R_o \in \psi_2$ **then**
            $\psi_1 \leftarrow \psi_1 \cup \psi_2$
            $\Psi \leftarrow \Psi \setminus \{\psi_2\}$
          end if
        end for
      end if
    end for
    if $R_o = (-1, -1)$ **then** ▷ Then no other subgraph contains $f_{i',j',k',l'}$
      $\psi_{new} \leftarrow \{R_t, R_r\}$
      $\Psi \leftarrow \Psi \cup \{\psi_{new}\}$
    end if
  end for
end procedure

Algorithm A.3 Computation of the k-neighborhood Conflict Graph on n_i
Input: Ψ
Output: $G_c(\Psi, E)$, the conflict graph of the k-neighborhood subgraphs

procedure FindConflictGraph(Ψ)
  $E \leftarrow \varnothing$ ▷ The edge set of G_c
  for all $\psi_1 \in \Psi$ **do**
    for all $(i, k) \in \psi_1$ **do**
      for all $f_{i,j,k,l} \in F$ **do**
        for all $\psi_2 \in (\Psi \setminus \{\psi_1\})$ **do**
          for all $(j', l') \in \psi_2$ **do**
            if $(\psi_1, \psi_2) \in E$ **then**
              `break`
            end if
            for all $f_{i',j',k',l'} \in F$ **do**
              if $d(P_i, P_{j'}) \leq d_I$ **then**
                $E \leftarrow E \cup \{(\psi_1, \psi_2)\}$
                `break`
              end if
            end for
          end for
        end for
      end for
    end for
  end for
end procedure

Algorithm A.4 Select Radio Manager
Input: (i, k), radio whose channel selector is to be determined
Input: $M_{|\Psi|}$
Input: M_{Ψ_c}
Output: m, channel selector node's id

procedure SELECTORID((i, k), $M_{|\Psi|}$, M_{Ψ_c})
  $m \leftarrow -1$
  $c_{max} \leftarrow 0$
  $d_{min} \leftarrow \infty$
  for all $i' \in \text{keys}[M_{\Psi_c}]$ **do**
    for all $\psi \in M_{\Psi_c}[i']$ **do**
      if $(i,k) \in \psi \ \wedge \ (m = -1 \ \vee \ M_{|\Psi|}[i'] > c_{max} \ \vee \ (M_{|\Psi|}[i'] = c_{max} \ \wedge \ d(P_{i'}, P_i) > d_D \wedge d_{min} > d(P_{i'}, P_i)) \vee (M_{|\Psi|}[i'] = c_{max} \wedge (d(P_{i'}, P_i) \leq d_D \vee d(P_{i'}, P_i) = d_{min}) \wedge m > i'))$ **then**
        $m \leftarrow i'$
        $c_{max} \leftarrow |M_{|\Psi|}[i']|$
        $d_{min} \leftarrow d(P_{i'}, P_i)$
      end if
    end for
  end for
end procedure

Algorithm A.5 Prepare Delegation Map
Inputs: Ψ, S_i, T
Output: M_D, the dictionary that holds the master radio of a remotely managed slave radio

procedure PREPAREDLGMAP(Ψ, S_i, T)
  for all $\psi_c \in S_i$ **do**
    $\psi_d \leftarrow \psi_c \setminus T$
    if $\psi_d \neq \psi_c$ **then** ▷ if this colour class is remotely managed by radio (m, k_m)
      for all $(i, k) \in \psi_d$ **do**
        $\psi \leftarrow \{(i', k') : \exists\ \psi \in \Psi, (i, k) \in \psi \wedge (i', k') \in \psi\}$ ▷ ψ is the subgraph of radio (i, k)
        $d_{min} \leftarrow \infty$
        $(i_d, k_d) \leftarrow (-1, -1)$
        for all $(i', k') \in \psi$ **do**
          if $d(P_{i'}, P_m) \leq d_{min}$ **then**
            $(i_d, k_d) \leftarrow (i', k')$
            $d_{min} \leftarrow d(P_{i'}, P_m)$
          end if
        end for
        if $d_{min} \leq d_D$ **then** ▷ d_D is the delegation range
          $M_D[(i_d, k_d)] \leftarrow (m, k_m)$
          $\psi_d \leftarrow \psi_d \setminus \psi$
        end if
      end for
      if $\psi_d = \varnothing$ **then**
        $S_i \leftarrow S_i \setminus \{\psi_c\}$
      else
        $\psi_c \leftarrow \psi_d$
      end if
    end if
  end for
end procedure

Algorithm A.6 Computation of the Colour Classes' Weighted Conflict Graph on n_i
Input: S_A
Output: $G_c(S_A, E)$, conflict graph of the colour classes
Output: W_E, dictionary of edge weights of $G_c(S_A, E)$

procedure FindWeightedConflictGraph(S_A)
  $E \leftarrow \varnothing$ ▷ The edge set of G_c
  for all $\psi_1 \in S_A$ **do**
    for all $(i, k) \in \psi_1$ **do**
      for all $f_{i,j,k,l} \in F$ **do**
        for all $\psi_2 \in (S_A \setminus \{\psi_1\})$ **do**
          for all $(j', l') \in \psi_2$ **do**
            if $d(P_i, P_{j'}) > d_I$ **then**
              `continue`
            end if
            for all $f_{i',j',k',l'} \in F$ **do**
              $E \leftarrow E \cup \{(\psi_1, \psi_2)\}$
              if $d(P_i, P_{j'}) < 1.0$ **then**
                $d \leftarrow 1.0$
              else
                $d \leftarrow d(P_i, P_{j'})$
              end if
              $W_E[(\psi_1, \psi_2)] \leftarrow W_E[(\psi_1, \psi_2)] + \frac{1}{d^\alpha}$ ▷ α is the path loss exponent
            end for
          end for
        end for
      end for
    end for
  end for
end procedure

Algorithm A.7 Channel List Initialization Algorithm Running on n_i
Input: $|S_A|$, cardinality of the set of colour classes of remotely and locally managed radios
Output: L, channel list

procedure ChList($|S_A|$)
  $\delta \leftarrow 0$
  $f \leftarrow 1$
  $C \leftarrow 11$ ▷ 11 channels for 802.11b/g
  if $|S_A| = 1$ **then** ▷ Then randomly select a channel
    $f \leftarrow$ A *random* channel
  else
    $\delta \leftarrow \frac{C-1}{|S_A|-1}$
  end if
  for all $i \in \mathbb{Z} \wedge i \in [0, |S_A|)$ **do**
    $ch \leftarrow f + i\,\delta$
    Round ch to the nearest integer
    if $ch > C$ **then**
      $ch \leftarrow C$
    end if
    $L_i \leftarrow ch$
  end for
end procedure

Algorithm A.8 Channel Selection Estimation for the k-neighborhood of n_i
Input: L, list of available (not yet assigned, free) channels
Input: M_C
Input: m, current channel selector node's id
Input: c, number of channels to be selected for n_m
Output: M_L, node id-selected channels list mappings

```
1: procedure ChSelection(L, M_C, m, c)
2:   if c = 0 then
3:     return
4:   end if
5:   l ← LeastPriorNode(M_C)
6:   if l ≠ −1 then
7:     c' ← M_C[l]
8:     Del M_C[l]                                  ▷ Remove key l from M_C
9:     ChSelection(L, M_C, l, c')
10:  end if
11:  if m = −1 then
12:    return
13:  end if
14:  L_S ← SelectCh(L, c)
15:  M_L[m] ← L_S
16: end procedure
```

Algorithm A.9 Channel Selection from Available Channels
Input: L, list of available (not yet assigned, free) channels
Input: c, number of channels to be selected
Output: L_S, list of c channels selected from L

```
1: procedure SelectCh(L, c)
2:   δ ← 0
3:   L_S ← ∅
4:   C ← 11                                          ▷ 11 channels for 802.11b/g
5:   if c ≤ 1 then          ▷ Then the median of the available channels list is selected
6:     Append median[L] to L_S
7:     Remove one instance of median[L] from L
8:   else
9:     δ ← (C−1)/(c−1)
10:  end if
11:  for all i ∈ ℤ ∧ i ∈ [0, c) do
12:    ch ← 1 + i δ
13:    Round ch to the nearest integer
14:    if ch > C then
15:      ch ← C
16:    end if
17:    if ch ∈ L then
18:      selectedCh ← ch
19:    else                                  ▷ Then select the closest channel to ch from L
20:      dist ← ∞
21:      for all channel ∈ L do
22:        if |ch − channel| < dist then
23:          selectedCh ← channel
24:          dist ← |ch − channel|
25:        end if
26:      end for
27:    end if
28:    Append selectedCh to L_S
29:    Remove selectedCh from L
30:  end for
31: end procedure
```

Algorithm A.10 Least Prior Node Selection on n_i
Input: M_C
Output: l, least prior node's id

procedure LEASTPRIORNODE(M_C)
  $c_{min} \leftarrow \infty$
  $l \leftarrow -1$
  $d_{max} \leftarrow -1$
  for all $j \in$ keys$[M_C]$ **do**
    if $c_{min} > M_C[j] \vee (c_{min} = M_C[j] \wedge d(P_i, P_j) > d_D \wedge d_{max} < d(P_i, P_j)) \vee (c_{min} = M_C[j] \wedge (d(P_i, P_j) \leq d_D \vee d_{max} = d(P_i, P_j)) \wedge l < j)$ **then**
      $l \leftarrow j$
      $c_{min} \leftarrow M_C[j]$
      $d_{max} \leftarrow d(P_i, P_j)$
    end if
  end for
end procedure

Algorithm A.11 Distribute Channels Using Colour Classes' Conflict Graph on n_i
Input: S_i, set of sets of radios n_i is responsible for selecting channels
Input: $G_c(S_A, E)$, colour classes' conflict graph
Input: W_E, dictionary of edge weights of $G_c(S_A, E)$
Input: M_I, dictionary of manager node id's for the colour classes in S_A
Input: M_L, node id-selected channels list mappings
Output: π_i, candidate channel configurations for the radios of n_i
Output: M_R, dictionary of remotely managed radios' channels

procedure CHDIST(S_i, $G_c(S_A, E)$, W_E, M_I, M_L)
  if $|S_i| = 0$ **then**
    `return`
  end if
  $M_V \leftarrow \varnothing$ ▷ initialize a dictionary that holds vertex-channel mappings
  $v \leftarrow$ TRAVERSENEXT(**nil**, $G_c(S_A, E)$, W_E)
  while $v \neq$ **nil do**
    $\forall c, I_c \leftarrow 0.0$
    for all $c \in M_L[M_I[v]]$ **do** ▷ for each candidate channel c
      for all $(v, w) \in E$ **do**
        if $w \in$ keys$[M_V]$ **then** ▷ a channel has been selected for w
          $I_c \leftarrow I_c + \frac{W_E[(v,w)]}{|c - M_V[w]|}$
        end if
      end for
    end for
    $M_V[v] \leftarrow c$, such that I_c is minimum in I ▷ channel c is selected for colour class v
    $M_L[M_I[v]] \leftarrow M_L[M_I[v]] \setminus M_V[v]$ ▷ Remove channel c from candidate channels for node $M_I[v]$
    $v \leftarrow$ TRAVERSENEXT(v, $G_c(S_A, E)$, W_E)
  end while
  for all $v \in S_i$ **do**
    for all $(i', k') \in v$ **do**
      if $i' = i$ **then**
        $\pi_{i'}^{k'} \leftarrow M_V[v]$
      else
        $M_R[(i', k')] \leftarrow M_V[v]$
      end if
    end for
  end for
end procedure

Algorithm A.12 Traverse Next Vertex
Input: v, current vertex being visited
Input: $G_c(S_A, E)$
Input: W_E
Output: v, next vertex to be visited

procedure TraverseNext(v, $G_c(S_A, E)$, W_E)
  $\forall v \in S_A, W_V[v] \leftarrow \sum_{(v,w)\in E} W_E[(v,w)]$ ▷ Vertex weight is the sum of incident edges' weights
  if $v \neq$ **nil then**
    Select $(v, w) \in E$ such that $W_E[(v, w)]$ is maximum in W_E and w has not been visited yet
    if $w \neq$ **nil then** ▷ if such a w exists
      `return` w
    end if
  end if
  Select $v \in S_A$ such that $W_V[v]$ is maximum in W_V and v has not been visited yet
  `return` v ▷ v is **nil** if no such vertex exists
end procedure

Algorithm A.13 Announce Channel Selections on n_i
Input: S_i, set of sets of radios n_i is responsible for selecting channels
Input: M_D, dictionary that holds the master radio of a remotely managed slave radio
Input: π, candidate channel configurations for the radios in S_i
Input: Π, list of sets of channel selection announcements
Input: C, list of channel selection variables

procedure AnnounceSelections(S_i, M_D, π, Π, C)
  for all $\psi \in S_i$ **do**
    for all $(i', k') \in \psi$ **do**
      if $i' = i$ **then**
        $C_{k'} \leftarrow \pi_{i'}^{k'}$ ▷ Initialize channel selection variables
        $\Pi_{k'} \leftarrow \Pi_{k'} \cup \left\{ (|\Psi_i|, X_i, i, k', \pi_{i'}^{k'}) \right\}$
        Send `CS` message, $\left(|\Psi_i|, X_i, i, k', \pi_{i'}^{k'}\right)$, to one-hop neighbors on the subgraph of (i', k')
      else
        Send `CS` message, $\left(|\Psi_i|, X_i, i, k', M_R[(i', k')]\right)$, to (i', k')
      end if
    end for
  end for
  for all $(i', k') \in$ keys$[M_D]$ **do** ▷ Send delegation requests
    Send `DR` message, $\left((i', k')\right)$, to $M_D[(i', k')]$
  end for
end procedure

Algorithm A.14 Handle CS Announcement on n_i
Input: Π, list of sets of channel selection announcements
Input: M_P, proxy dictionary
Input: k, radio id for which the announcement has been received

procedure HANDLECSANNOUNCEMENT(Π, M_P)

$$\Psi_{max_k} \leftarrow \max\Big\{|\Psi_z|\text{: for } \exists(X_z, z, l, \pi_z^l),\ (|\Psi_z|, X_z, z, l, \pi_z^l) \in \Pi_k\Big\}$$

$$X_{max_k} \leftarrow \max\Big\{X_z : \text{for } \exists\Big(\Psi_{max_k}, z, l, \pi_z^l\Big),\ (\Psi_{max_k}, X_z, z, l, \pi_z^l) \in \Pi_k\Big\}$$

$$N_{min_k} \leftarrow \min\Big\{z : \text{for } \exists(\Psi_{max_k}, X_{max_k}, l, \pi_z^l),\ (\Psi_{max_k}, X_{max_k}, z, l, \pi_z^l) \in \Pi_k\Big\}$$

$$D_{min_k} \leftarrow \min\Big\{l : \text{for } \exists(\Psi_{max_k}, X_{max_k}, N_{min_k}, l, \pi_z^l),\ (\Psi_{max_k}, X_{max_k}, N_{min_k}, l, \pi_z^l) \in \Pi_k\Big\}$$

$C_k \leftarrow \pi_z^l$ ▷ Where $\Big(\Psi_{max_k}, X_{max_k}, N_{min_k}, D_{min_k}, \pi_z^l\Big) \in \Pi_k$
if $|\Psi_j| > \Psi_{max_k} \vee (|\Psi_j| = \Psi_{max_k} \wedge X_j > X_{max_k}) \vee \big(|\Psi_j| = \Psi_{max_k} \wedge X_j = X_{max_k} \wedge N_{min_k} > j\big) \vee \big(N_{min_k} == j \wedge D_{min_k} > k'\big)$ **then**
  Send `CS` message, $\Big(|\Psi_j|, X_j, j, k', \pi_j^{k'}\Big)$, to one-hop neighbors on the subgraph of (i, k)
  for all $(z, l) \in M_P[(i, k)]$ **do** ▷ Announce to delegated radios
    Send `CS` message, $\Big(|\Psi_j|, X_j, j, k', \pi_j^{k'}\Big)$, to (z, l)
  end for
end if
$\Pi_k \leftarrow \Pi_k \cup \Big\{(|\Psi_j|, X_j, j, k', \pi_j^{k'})\Big\}$
end procedure