Έλεγχος ισχύος σε ασύρματα ad-hoc δίκτυα, για throughput maximization

Αναστάσιος Γιαννούλης

Επιβλέποντες καθηγητές: Λέανδρος Τασιούλας, Κώστας Τσουκάτος

1 Εισαγωγή

Με την ευρεία ανάπτυξη ασυρμάτων δικτύων και απαιτητικών εφαρμογών σε ρυθμό εξυπηρέτησης, είναι επιτακτική η ικανοποιήση της ζήτησης υψηλών ρυθμών. Επομένως είναι αναγκαίο να εκμεταλλευτεί πλήρως η χωρητικότητα του δικτύου. Ο έλεγχος ισχύος συνήθως χρησιμοποιούταν για να μετριαστεί η παρεμβολή και επομένως να ικανοποιηθούν απαιτήσεις ποιότητας του λαμβανομένου σήματος. Ωστόσο πρόσφατη δουλεία προσανατολίζει τον έλεγχο ισχύος στην κατεύθυνση της μεγιστοποίησης μίας συνολικής ωφελιμότητας του δικτύου ([8],[2],[9]). Οι ρυθμοί μετάδοσης σε ένα ασύρματο κανάλι είναι άρτια συνδεδεμένη με την μεταδίδουσα ισχύ. Είναι λογικό να σκεφτούμε πως ένας κατάλληλος έλεγχος ισχύος θα οδηγήσει σε βελτιστοποιηση του throughput του δικτύου. Ωστόσο η έλλειψη κεντρικοποιημένης υποδομής στα ασύρματα ad-hoc δίκτυα επιτάσσει μία κατανεμημένη αντιμετώπιση του προβλήματος. Σε αυτή την δουλειά, προτείνεται ενας κατανεμημένος throughput-optimal αλγόριθμος ελέγχου ισχύος.

Το δίκτυο θα πρέπει να είναι ικανό να έχει την επιθυμητή λειτουργεία, όποιες και αν είναι οι στατιστικές των ρυθμών αφιξης της κίνησης. Επομένως ειναι επιτακτική η λειτουργία για ρυθμούς άφιξης άγνωστων στατιστικών, καθώς και για ρυθμούς άφιξης εκτός και εντός της περιοχής ευστάθειας του δικτύου.

Όπως θα δούμε στη συνέχεια, έλεγχος ροής εφαρμόζεται σε κάθε χρήστη προκειμένου να εξασφαλίσουμε τέτοια συμπεριφορά.

Η συνέχεια είναι οργανωμένη ως εξής: Στο τμήμα 2 περιγράφουμε το μοντέλο του ασυρμάτου δικτύου. Στο τμήμα 3 διατυπώνουμε μαθηματικά το πρόβλημα. Τεχνικές ελέγχου ροής μελετώνται στο τμήμα 4. Τέλος το τμήμα 5 περιέχει αποτελέσματα προσομοίωσης.

2 Μοντέλο συστήματος

Θεωρούμε ένα ασύρματο ad-hoc δίκτυο που αποτελείται από N κόμβους/χρήστες. Έστω $\mathcal{N}$ το σύνολο όλων των χρηστών, $|\mathcal{N}| = N$. Δηλώνουμε με $\mathcal{N}_i \subseteq \mathcal{N}$, το σύνολο των χρηστών στους οποίους ο χρήστης i μπορεί να στείλει δεδομένα σύμφωνα με κάποιο κριτήριο γειτνίασης. Δεδομένα μπορούν να μεταδοθούν από έναν κόμβο j σε οποιονδήποτε άλλον k, εφόσον $k \in \mathcal{N}_j$. Έστω σύνδεσμος l ένα πιθανό ζευγάρι από χρήστες (j,k) που επικοινωνούν, και $\mathcal{L}$ η συλλογή όλων των δυνατών τέτοιων ζευγαριών, $|\mathcal{L}| = L$. Δηλώνουμε με $trans(l)$ και $rcv(l)$ τον πομπό και τον δέκτη της μετάδοσης στο σύνδεσμο l αντίστοιχα. Για κάθε ζευγάρι μεταδόσεων (l_i, l_j), το κέρδος του συνδέσμου μεταξύ $trans(l_i)$ και $rcv(l_j)$ ορίζεται με G_{ij}, και $\mathbf{G} = \{G_{ij}, \quad i,j = 1:L\}$ είναι ο πίνακας κερδών. Η γειτνίαση μπορεί να καθοριστεί ούτως ώστε ενδεχόμενα G_{ii} να είναι μεγαλύτερα από ένα συγκεκριμένο όριο.

Εισερχόμενη κίνηση στον χρήστη i μπορεί να μεταδοθεί απ' ευθείας σε έναν χρήστη j εάν $j \in \mathcal{N}_i$. Αλλιώς, η κίνηση μπορεί να προωθηθεί στον χρήστη j μετά από μια πορεία πολλών ενδιάμεσων αλμάτων. Έστω λ_{ij} ο ρυθμός άφιξης της εξογενούς κίνησης στον χρήστη i με προορισμό τον j, και $\boldsymbol{\lambda} = \{\lambda_{ij}, \quad j = 1:N, i = 1:N\}$ το διάνυσμα ρυθμών αφίξεων του συστήματος. Εισερχόμενη κίνηση στον χρήστη i, εάν δεν μεταδίδεται απευθείας χωρίς καθυστέρηση αποθηκεύεται στην ουρά του χρήστη i. Έστω W_i το συνολικό μήκος ουράς του χρήστη i, και W_i^j η συνολική ποσότητα κίνησης αποθηκευμένη στην ουρά του i, η οποία έχει προορισμό των χρήστη j. Προφανώς, $\sum_{j=1}^{N} W_i^j = W_i$.

Δεδομένα μπορούν να σταλούν σε κάθε σύνδεσμο $l = (j,k)$ με μία συγκεκριμένη ισχύ p_l, και έστω $\mathbf{p} = \{p_i, \quad i = 1:L\}$, το διάνυσμα ισχύος του συστήματος. Τότε η συνολική ισχύς εκπομπής του χρήστη k είναι $P_k^{total} = \sum p_i, \quad if \quad k = trans(i)$. Η ισχύς μετάδοσης p_i για μια μετάδοση i περιορίζεται από ένα άνω όριο P_i^{max}, και έστω $\mathbf{P^{max}} = \{P_i^{max}, \quad i = 1:L\}$ το αντίστοιχο διάνυσμα περιοριστικών ορίων.

Το SINR γ_i και η παρεμβολή I_i στον δέκτη της μετάδοσης i δίνονται από:

$$\gamma_i = \frac{p_i G_{ii}}{I_i + n},$$

$$I_i = \frac{1}{G_1} \sum_{\substack{j=1, j\neq i \\ trans(j)=trans(i)}}^{L} p_j G_{ji} + \frac{1}{G_2} \sum_{\substack{j=1, j\neq i \\ trans(j)\neq trans(i)}}^{L} p_j G_{ji}$$

όπου G_1, G_2 είναι οι CDMA παράμετροι κέρδους για σήματα από τον ίδιο και διαφορετικούς πομπούς αντίστοιχα, και n θεωρείται προσθετικός λευκός γκαουσιανος θόρυβος.

Ο ρυθμός μετάδοσης C_i για μία μετάδοση i καθορίζεται από το θεώρημα χωρητικότητας του Shannon : $C_i = \log(1 + \gamma_i)$. Είναι σημαντικό να παρατηρηθεί πως στην περίπτωση υψηλών SINR , τότε οι ρυθμοί μετάδοσης μπορούν επιτυχώς να προσεγγιστούν από την απλουστευμένη σχέση $C_i = \log(\gamma_i)$. Εστω $\mathbf{C} = \{C_i, \quad i = 1 : L\}$ το δυάνυσμα ρυθμών μετάδοσης των συνδέσμων.

3 Διαμόρφωση Προβλήματος

Προφανώς αλλάζοντας απλά την κάθε ισχύ p_i δεν αρκεί για να αυξήσουμε τον κάθε ρυθμό μετάδοσης $C_i(\gamma_i(\mathbf{p}))$, διότι το SINR $\gamma_i(\mathbf{p})$ επηρεάζεται επίσης από τις υπόλοιπες ισχείς $\mathbf{p}_{-i} = \{p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_L\}$. Επίσης, ταυτόχρονη αύξηση περισότερης από μίας C_i είναι περισότερο περίπλοκο πρόβλημα, αφού αυξάνοντας μία ισχύ p_i καταλήγει σε υψηλότερο ρυθμό C_i, αλλά μειώνει άλλους ρυθμούς μετάδοσης $\mathbf{C}_{-i} = \{C_1, \ldots, C_{i-1}, C_{i+1}, \ldots, C_L\}$ εξάιτίας της παραγόμενης παρεμβολής. Ωστοσο είναι επιθυμητό να εκμεταλλευτούμε πλήρως όλη τη διαθέσιμη χωρητικότητα του δικτύου. Επομένως ο σκοπός αυτής της δουλειας προσανατολίζεται στην εύρεση και επιλογή του κατάλληλου διανύσματος ισχύος σε χρονική στιγμή t, ουτως ώστε μακροπρόθεσμα η διεκπαιραίωση του δικτύου να μεγιστοποιείται. Ισοδύναμα, το σύστημα πρέπει να παραμένει σταθερό για κάθε εφικτή ακολουθία αφίξεων, έστω Λ η συλλογή όλων των εφικτών διανυσμάτων ρυθμών αφίξεων. Συμβολίζουμε με $\mathbf{p}^*(l)$ την κατάλληλη επιλογή του διανύσματος ισχύος τη χρονική στιγμή t, ουτως ώστε το σκοπός μας να εξυπηρετείται μακροπρόθεσμα.

Στο [2],ένας κατανεμημένος αλγόριθμος είχε προταθεί σύμφωνα με τον οποίο το επιθυμητό διάνυσμα ισχύος $\mathbf{p}(t)$, μεγιστοποιούσε το άθροισμα των ωφελιμότητων των καναλιών. Οι ωφελιμότητες είναι μία έκφραση του ρυθμού μετάδοσης του καναλιού C_i: $(u_i(\gamma_i(\mathbf{p})) = \theta_i \log(\gamma_i(\mathbf{p}))$ ή $u_i(\gamma_i(\mathbf{p})) = \theta_i \log(1 + \gamma_i(\mathbf{p})))$, όπου θ_i είναι παράμετροι προτεραιότητας μετάδοσης. Η εύρεση ένός διανύσματος ισχύος $\mathbf{p}$ που απλά μεγιστοποιεί το άθροισμα των χωρητικοτήτων $(\sum_{i=1}^{L} \theta_i C_i(t))$ δεν είναι αρκετό για throughput maximization .

Στο [1], μία maximum throughput policy προτάθηκε, δίνοντας προτεραιοτητες σε ροές με μεγαλύτερες ουρές. Επομένως είναι λογικό να υποθέσουμε πως είναι επιτακτικό να χρησιμοποιήσουμε πληροφορία σχετίζουσα με το μήκος

των ουρών για να βρούμε το $\mathbf{p}^*(t)$.

Για κάθε μετάδοση i, και για κάθε προορισμό j θεωρούμε την εξής ποσότητα: $i = 1 : L, j = 1 : N$

$$X_i^j(t) = \begin{cases} W^j_{trans(l)}(t) - W^j_{rcv(l)}(t), & if \quad rcv(l) \neq j \\ W^j_{trans(l)}(t), & if \quad rcv(l) = j \end{cases}$$

Στο παρελθόν μία back-pressure policy προτάθηκε. Υιοθετόντας αυτήν την προσέγγιση, σε ένα multi-hop δίκτυο, μία μετάδοση i εξυπηρετεί δεδομένα με προορισμό τον χρήστη j ουτως ώστε το $\sum_{i=1}^{L} X_i^j C_i$ να μεγιστοποιείται. Ποιανής κλάσης προορισμού δεδομένα εξυπηρετούνται τη χρονική στιγμή t δηλώνεται από το δυαδικό δυάνυσμα:

$$I(t) = \{I_i^j(t), \quad i = 1 : L, j = 1 : N\} \quad , kai$$

$$I_i^j(t) = \begin{cases} 1, & \text{αν ο σύνδεσμος } i \text{ εξυπηρετεί δεδομένα με προορισμό το χρήστη } j \\ 0, & \text{αλλιώς} \end{cases}$$

Στο [3], αποδείχτηκε πως όταν υιοθετείται το $C_i = \log(\gamma_i)$ μοντέλο χωρητικότητας, τότε η capacity region είναι κοίλη. Σε αυτή την περίπτωση, όσο πιο πολλές μεταδόσεις γίνονται ταυτόχρονα, τόσο μεγαλύτερη η αύξηση του throughput του δικτύου χώρις να ανυσηχούμε εξάιτίας της παραγόμενης παρεμβολής. Επομένως , η έυρεση του βέλτιστου δυιανύσματος ενεργοποίησης $I^*(t)$ δεν απαιτεί εξαντλητική εύρεση όλων των δυνατών συνδυασμών μεταδόσεων, αφού όλες οι δυνατές μεταδόσεις είναι ενεργοποιημένες.

Δηλώνουμε με απλά $X_i(t)$, το $X_i^j(t)$ για το οποίο $I_i^j(t) = 1$.

Ισχυριζομαστε πως το $\mathbf{p}^*(t)$ είναι η λύση στο ακόλουθο πρόβλημα βελτιστοποίησης:

Πρόβλημα 1

$$\max \sum_{i=1}^{L} X_i u_i(\gamma_i(\mathbf{p}))$$

$$s.t. \qquad 0 < p_i < P_i^{max}$$

όπου η σύνάρτηση ωφελιμότητας του κάθε πομπού u_i αντιστοιχεί στο μοντέλο χωρητικότητας καναλιού, $u_i(\gamma_i(\mathbf{p})) = C_i(\gamma_i(\mathbf{p})) = \log(1+\gamma_i(\mathbf{p}))$, ή ισοδύναμα $u_i(\gamma_i(\mathbf{p})) = C_i(\gamma_i(\mathbf{p})) = \log(\gamma_i(\mathbf{p}))$ στην περίπτωση υψηλών SINR .

Το $\mathbf{p}^*$ σαν τοπικό βέλτιστο στο πρόβλημα 1, θα ικανοποιεί τις KKT συνθήκες:

$$\left(X_i \frac{\partial u_i(\gamma_i(p))}{\partial p_i} + \sum_{\substack{j=1 \\ j \neq i}}^{L} X_j \frac{u_j(\gamma_j(p))}{\partial p_i} \right) \Bigg|_{p=p^*} = \lambda_i^* - \mu_i^*$$

$$\lambda_i^*(p_i^* - P_i^{max}) = 0, \qquad \mu_i^* p_i^* = 0, \qquad \lambda_i^*, \mu_i^* \geq 0$$

όπου $\boldsymbol{\lambda}^* = \{\lambda_i^*, \quad i = 1 : L\}$, $\boldsymbol{\mu}^* = \{\mu_i^*, \quad i = 1 : L\}$ είναι μοναδικοί πολλαπλασιαστές Lagrange .

Ο χρήστης j χρεώνει άλλους χρήστες για παραγωγή παρεμβολής την ακόλουθη τιμή κόστους π_j:

$$\pi_j = -X_j \frac{\partial u_j(\gamma_j(\mathbf{p}))}{\partial I_j(\mathbf{p})},$$

έστω $\boldsymbol{\pi}$ το αντίστοιχο διάνυσμα τιμών κόστους $\boldsymbol{\pi} = \{\pi_i, \quad i = 1 : L\}$.

Χρησιμοποιώντας τιμές κόστους, οι KKT συνθήκες μπορούν να μετασχηματιστούν ως εξής:

$$X_i \left(\frac{\partial u_i(\gamma_i(p))}{\partial p_i} \right) \Bigg|_{\boldsymbol{p}=\boldsymbol{p}^*} - \sum_{\substack{j=1 \\ j \neq i}}^{L} \pi_j G_{ij} = \lambda_i^* - \mu_i^*$$

Όσον αφορά τα εσωτερικά σημεία του προβλήματος, οι πολλαπλασιαστές Lagrange λ_i^*, μ_i^* είναι ίσοι με 0, $\forall i = 1 : L$.

Επομένως το να ικανοποιούνται οι KKT συνθήκες είναι ισοδύναμο με το να μεγιστοποιείται η ακόλουθη συνάρτηση επιπρόσθετου κέρδους:

$$s_i(\mathbf{p}, \boldsymbol{\pi}) = X_i u_i(\gamma_i(\mathbf{p})) - p_i \sum_{\substack{j=1 \\ j \neq 1}}^{L} \pi_j G_{ij},$$

$$and \qquad \mathbf{p}^* = \arg\max s(\mathbf{p}, \boldsymbol{\pi}).$$

Θεωρούμε με u_i' τη μερική παράγωγο του u_i ως προς p_i:

$$u_i'(\mathbf{p}) = \frac{\partial u_i(\gamma_i(\mathbf{p}))}{\partial p_i}$$

Επομένως,

$$p_i^* = u_i'^{-1} \left(\frac{1}{X_i} \sum_{\substack{j=1 \\ j \neq i}}^{L} \pi_j G_{ij} \right)$$

Στην περίπτωση που $u_i = \log(\gamma_i)$, τότε

$$\pi_j(t+1) = -\frac{X_j}{I_j(\mathbf{p}) + n},$$

$$p_i(t+1) = \frac{X_i}{\sum_{\substack{j=1 \\ j \neq i}}^{L} \pi_j G_{ij}}$$

ενώ

$$\pi_j(t+1) = -\frac{X_j}{I_j(\mathbf{p}) + n}\frac{\gamma_j}{\gamma_j + 1},$$

$$p_i(t+1) = \frac{X_i}{\sum_{\substack{j=1 \\ j\neq i}}^{L} \pi_j G_{ij}}\frac{\gamma_i}{\gamma_i + 1}$$

όταν $u_i = \log(1+\gamma_i)$.

Για σταθερές τιμές X_i, αυτή η προσέγγιση είναι όπως στο [2], όπου οι σταθερές τιμές X_i δίνουν τα βάρη προτεραιότητας, και το $\mathbf{p}$ συγκλίνει στο διάνυσμα ισχύος το οποίο μεγιστοποιεί το αντίστοιχο βεβαρυμένο άθροισμα των χωρητικοτήτων των καναλιών.

Σύμφωνα με το [2], όταν η συνάρτηση ωφελιμότητας είναι $u_i = \log(\gamma_i)$ (η οποία είναι και τύπου 1 και τύπου 2 κυρτή συνάρτηση) ένα τέτοιο πρόβλημα και επομένως 1 έχει μία μοναδική βέλτιστη λύση, και οι KKT συνθήκες γίνονται ικανές και αναγκαίες. Ωστόσο όταν η συνάρτηση ωφελιμότητας είναι $u_i = \log(1+\gamma_i)$, τότε υπάρχουν περισσότερα του ενός σημεία ισορροπίας, και η σύγκλιση στο καθολικό βέλτιστο σημείο δεν είναι εγγυημένη. Στην πραγματικότητα ο έλεγχος ισχύος μπορεί να οδηγήσει σε οποιοδήποτε σημείο ισορροπίας ανάλογα με τις αρχικές τιμές των διανυσμάτων ισχύος και τιμών κόστους $(\mathbf{p_0},\boldsymbol{\pi}_0)$. Επομένως είναι δυνατό να μεγιστοποιήσουμε το $\sum_{i=1}^{L} X_i u_i(\gamma_i(\mathbf{p}))$ όταν $u_i = \log(\gamma_i)$. Στην περίπτωση υψηλών SINR , αυτό είναι ισοδύναμο με το να μεγιστοποιούμε $\sum_{i=1}^{L} X_i C_i(\gamma_i(\mathbf{p}))$.

Φυσικά σε κάθε χρονική στιγμή, τα μήκη ουρών, και οι ενεργοποιημένες κλάσεις για κάθε μετάδοση μεταβάλλονται. Επομένως τη χρονική στιγμή $t+1$ ένα νέο πρόβλημα βελτιστοποίησης προκύπτει, πριν συγκλίνουμε στη λύση του προηγούμενου προβλήματος βελτιστοποίησης της χρονικής στιγμής t. Ωστόσο ισχυριζόμαστε πως μακροπρόθεσμα οι βελτιστες αλλαγές του διανύσματος ισχύος είναι επαρκείς για να μεγιστοποιήσουμε την διαικπεραιότητα του δικτύου, βασιζόμενοι στο γεγονός πως τέτοιες άλλαγες (ουρές κ.α) είναι μικρότερου ρυθμού από το ρυθμό σύγκλισης.

4 Έλεγχος ροής

Απ'οτι ειπώθηκε πριν, δεν υπάρχει καμμία εγγύηση μέχρι στιγμής για την απόδοση του δικτύου για ρυθμούς αφίξεων πέρα από την capacity region . Όταν ένας ρυθμός άφιξης είναι πέρα από την capacity region , το μήκος ουράς τείνει να γίνει απεριόριστα μεγάλο. Είναι φανερό πως εφόσον η ισχύς εκπομπής ενός πομπού p_i εξαρτάται από το μήκος ουράς W_i, τότε ένας σχετικά μεγαλύτερος ρυθμός άφιξης θα κέρδιζε αποκλειστική χρήση της χωρητικότητας

του καναλιού, οδηγώντας τους άλλους χρήστες σε λιμοκτονία. Έιναι επιτακτικό να εφαρμοστεί ένας μηχανισμός ελέγχου ροής, σε τέτοιο τρόπο που να διατηρείται η throughput optimality αλλά επίσης και ένα επιθυμητό χαρακτηριστικό δικαιοσύνης να ικανοποιείται.

Οι στοχαστικές αφίξεις στο χρήστη i αποθηκεύονται αρχικά σε μια ενδιάμεση αποθηκευτική ουρά περιεχομένου A_i, πριν γίνουν διαθέσιμες για μετάδοση. Σε κάθε χρονική στιγμή t, αποφάσεις ελέγχου ροής καθορίζουν το ποσό της κίνησης R_{ij} με προορισμό το χρήστη j, που θα αφαιρεθούν από το buffer A_i (αν υπάρχει τόση διαθέσιμη) και θα προστεθούν στο W_i . Ο έλεγχος ισχύος γίνεται όπως πριν βασιζόμενοι στην κάθε ουρά W_i για κάθε χρήστη i. Κάθε κόμβος αξιολογεί το ρυθμό ροής του με μία άλλη συνάρτηση ωφελιμότητας $U_{ij}(R_{ij})$.

Πρόβλημα 2

$$\max \sum_{i=1}^{N} \sum_{j=1}^{N} U_{ij}(R_{ij})$$

$$s.t. \quad \mathbf{R} \in \Lambda$$

όπου Λ η περιοχή ευστάθειας του δικτύου. Δηλώνουμε με $C_i^{out}(t)$ τη συνολική εξερχόμενη κίνηση από το χρήστη i σε κάθε δυνατή διεύθυνση, τη χρονική στιγμή t. Θεωρούμε το αντίστοιχο δυαδικό πρόβλημα το οποίο είναι χωρίς περιορισμούς:

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} U_{ij}(R_{ij}(t)) - \sum_{i=1}^{N} \xi_i(t) \left(\sum_{j=1}^{N} R_{ij}(t) - C_i^{out} \right)$$

$$\frac{\partial U_{ij}(R_{ij}(t))}{\partial R_{ij}(t)} - \xi_i(t) = 0$$

$$\xi_i(t+1) = U'_{ij}(R_{ij}(t))$$

$$R_{ij}(t+1) = U'^{-1}_{ij}(\xi_i(t))$$

Από [6] μία κατανεμημένη λύση σε αυτό το δυαδικό πρόβλημα μπορεί να βρεθεί από:

$$\xi_i(t+1) = \left[\xi_i(t) + \frac{1}{K} \left(\sum_{j=1}^{N} R_{ij}(t) - C_i^{out}(t) \right) \right]^+$$

$$R_{ij}(t+1) = U'^{-1}_{ij}(\xi_i(t)),$$

όπου $K > 0$ και μεγάλο είναι μία σταθερά. Παρατηρούμε ότι , $\xi_i(t) = \frac{W_i(t)}{K}$, όπου $W_i(t)$ όπως ειπώθηκε προηγουμένως είναι το συνολικό μήκος ουράς $\sum_{j=1}^{N} W_i^j(t)$ στο κόμβο i , $i = 1 : N$. Το K ρυθμίζει τη συμπεριφορά του ελέγχου ροής. Υψηλότερο K οδηγεί σε καλύτερη προσέγγιση του επιθυμητου σημείο λειτουργίας, ωστόσο τα μήκη ουρών μεγαλώνουν, και επομένως η καθυστέρηση μεγαλώνει.

Ανάλογα με τη συνάρτηση ωφελιμότητας του ελέγχου ροής U_{ij}, μπορούμε να καταλήξουμε σε διαφορετικούς στόχους δικαιοσύνης. Στη συνέχεια θα ασχοληθούμε με δύο κριτήρια δικαιοσύνης: proportional fairness , και στη μεγιστοποίηση του βεβαρυμένου αθροίσματος των throughputs .

4.1 Proportional fairness

Η Proportional fairness μπορεί να επιτευχθεί όταν η συνάρτηση ωφελιμότητας του ελέγχου ροής είναι του τύπου : $U_{ij}(R_{ij}) = a_{ij}\log(R_{ij})$, όπου ο πίνακας προτεραιοτήτων a $= \{a_{ij}, \quad i,j = 1 : N\}$ καθορίζει την επιθυμητή δικαιοσύνη. Σε μία τέτοια περίπτωση η λύση στο πρόβλημα 2 δίνεται από:

$$R_{ij}(t+1) = \frac{a_{ij}}{\lambda_i(t)} = \frac{a_{ij}K}{W_i(t)}$$

4.2 Βεβαρυμένο άθροισμα throughputs

Ο σκοπός είναι να φτάσουμε το σημείο λειτουργείας για το οποίο ένα βεβαρυμένο άθροισμα throughputs μεγιστοποιείται [5], όπου τα βάρη καθορίζονται από τα a_{ij}. Αυτό σημαίνει ότι η εξυπηρέτηση πρέπει να γίνει τέτοια ώστε το $\sum a_{ij}r_{ij}$ να μεγιστοποιείται, όπου r_{ij} είναι ο μακροπρόθεσμος ρυθμός εξυπηρέτησης της αφικνούμενης στον χρήστη i κίνησης , με προορισμό το χρήστη j. Σε αυτή την περίπτωση $U_{ij}(R_{ij}) = a_{ij}R_{ij}$, και η λύση στο πρόβλημα 2 είναι :

$$R_{ij}(t+1) = \begin{cases} A_{ij}(t), & when \quad W_i(t) < a_{ij}K \\ 0, & when \quad W_i(t) > a_{ij}K \end{cases}$$

5 Πειραματικά αποτελέσματα

Στην αρχή, για απλότητα η συμπεριφορά ενός συμμετρικού 2χ2 δικτύου θα μελετηθεί. Πιο συγκεκριμένα μελετούμε την capacity region του δικτύου, και εξετάζουμε εάν είναι όσο πιο μεγάλη γίνεται. Είναι αναγκέιο να έχουμε μία αίσθηση για την ανάμενόμενη μέγιστη capacity region του δικτύου, για

να μπορούμε να κάνουμε τη σύγκριση. Επομένως υπολογίζουμε την Pareto Surface . Η Perron-Frobenius ανάλυση που ακολουθεί για ένα τέτοιο δίκτυο, είναι αναγκαία για τον υπολογισμό της Pareto surface . Αργότερα η απόδοση των τεχνικών ελέγχου ροής αξιολογείται. Τέλος εστιάζουμε στην multi-hop περίπτωση.

5.1 Perron-Frobenius Αναλυσις

Όπως μπορούμε να δούμε στο [3], εάν ένα σημείο ανήκει στη Pareto surface , τότε η Perron-Frobenius ιδιοτιμή είναι ίση με 1. Επομένως,

$$\lambda_{PF}(D\tilde{G}) = 1, \tag{1}$$

όπου

$$D \triangleq diag\{\frac{2^{C_i}}{G_{ii}}\},$$

$$\tilde{G} = \{\widetilde{G_{ij}}\}, \qquad \widetilde{G_{ij}} = \begin{cases} G_{ij}, & when \quad i \neq j \\ 0, & when \quad i = j \end{cases}$$

Για ένα 2χ2 δίκτυο, η λύση στην 1 είναι :

$$C_2 = \log \frac{G_{22}G_{11}}{G_{12}G_{21}} - C_1, \qquad if \quad C_i = \log \gamma_i$$

$$C_2 = \log \left(\frac{G_{22}G_{11}}{G_{12}G_{21}\,(2^{C_1})} - 1 \right), \qquad if \quad C_i = \log (1 + \gamma_i)$$

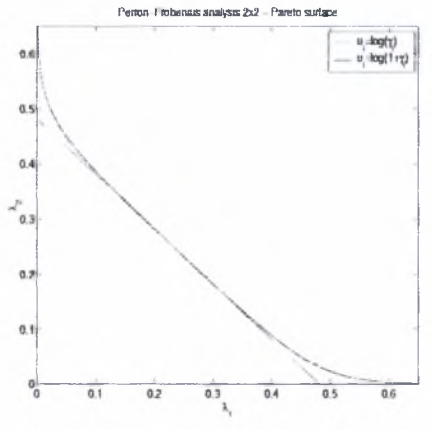
Οι 2 διαφορετικές Pareto surfaces που προκύπτουν δίνονται στο σχήμα 1. Είναι σημαντικό να παρατηρήσουμε στο σχήμα 1 ότι σε περίπτωση υψηλών SINR , όποια συνάρτηση από τις δύο χρησιμοποιήσουμε, δεν κάνει σημαντική διαφορά στην εκτίμηση της χωρητικότητας.

5.2 2χ2 συμμετρικό δίκτυο

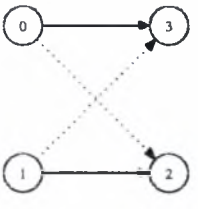
Θεωρούμε ένα απλό συμμετρικό 2χ2 δίκτυο, όπως φαίνεται στο σχήμα 2. Κίνηση φτάνει στον κόμβο 0 με προορισμό τον κόμβο 3, και στον κόμβο 1 με προορισμό τον κόμβο 2. Πιο συγκεκριμένα ο πίνακας κερδών είναι:

$$\mathbf{G} = \begin{bmatrix} 123.45 & 0.59 \\ 0.59 & 123.45 \end{bmatrix}$$

Στο σχήμα 3, η περιοχή ευστάθειας του συστήματος βρίσκεται για συνάρτηση ωφελιμότητας $u_i = \log(\gamma_i)$, ενώ στο σχήμα 4 για συνάρτηση $u_i = \log(1 + \gamma_i)$,



Σχήμα 1:



Σχήμα 2:

για διαφορετικούς περιορισμούς ισχύς P_i^{max}. Στην πρώτη περίπτωση εφόσον υπάρχει μόνο μία λύση στο πρόβλημα 1, τότε ακόμα και για υπερβολικά μεγάλες τιμές του P_i^{max}, η διαικπεραίωση του δικτύου μπορεί να γίνει όσο πιο μεγάλη γίνεται.

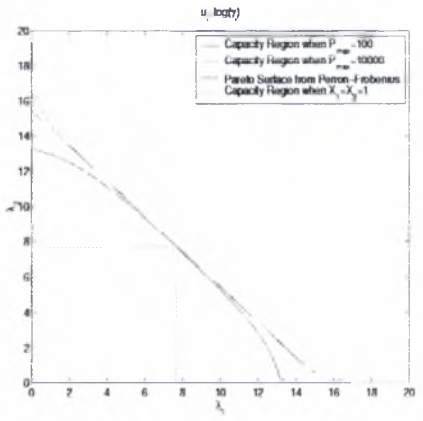
Ωστόσο για τη δεύτερη συνάρτηση, δεν υπάρχει εγγύηση για την απόδοση του δικτύου, αφού δεν είναι σίγουρο που οδηγεί ο έλεγχος ισχύος. Η Pareto surface δεν είναι κοίλη, και είναι δυνατό να επιτευχθεί διεκπαιρεωση μεγαλύτερη πέρα από την Pareto-Surface με time-sharing , όπως επίσης είναι πιθανό να κατλήξουμε σε υπο-βελτιστες λύσεις.

Επίσης και στις 2 περιπτώσεις βελτίωση στο throughput επιδυκνείεται σε σύκριση με τη μεγιστοποίηση του απλού αθροίσματος των βεβαρυμένων χωρητικοτήτων (Η τετράγωνη περιοχή ευστάθειας).

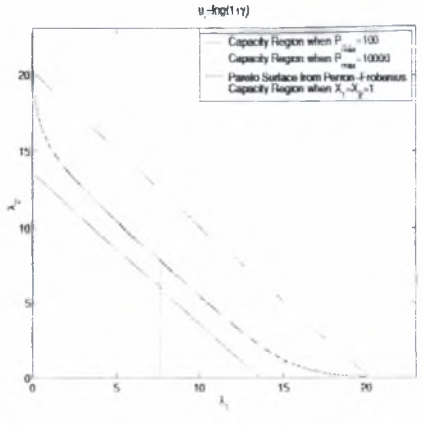
5.3 expo-rule

Μία διαφορετική αντιμετώπιση στο πρόβλημα 1, μπορεί να γίνει εάν στη θέση των W_i, χρησιμοποιηθούν:

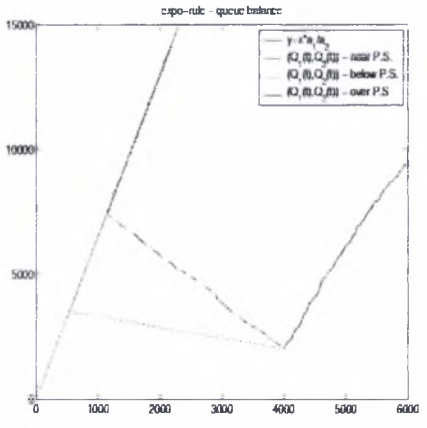
$$W_i' = e^{\left(\frac{a_i W_i - \overline{a_i W_i}}{1+\sqrt{\overline{a_i W_i}}}\right)}$$



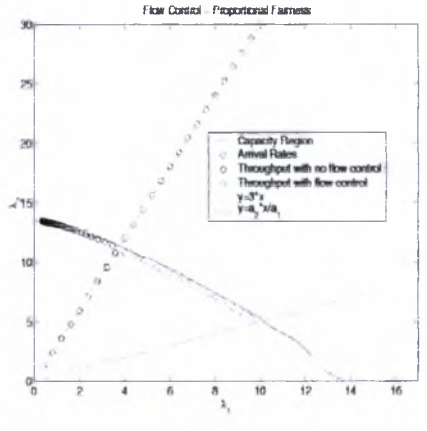
Σχήμα 3:



Σχήμα 4:



Σχήμα 5:

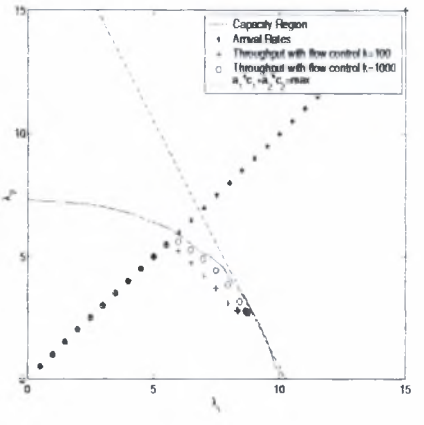


Σχήμα 6:

όπου $\overline{x_i} = \sum_{i=1}^{m} x_i/m$ είναι το μέσο x. Το πλεονέκτημα είναι ότι μπορεί να επιτευχθεί ένας σχετικός έλγχος στα μήκη των ουρών, για ρυθμούς άφιξης $\lambda \in \Lambda$, και κοντά στο όριο της περιοχής ευστάθειας, όπως φαίνεται στο σχήμα 5.

5.4 Ελεγχος ροής

Στο σχήμα 6 ο proportionally fair έλεγχος ροής δοκιμάζεται. Εισέρχονται ρυθμοί άφιξης που ικανοποιούν: $\lambda_2 = 3\lambda_1$, και η ρυθμοί εξυπηρέτησης με και χωρίς έλεγχο ροής συγκρίνονται. Οι προτεραιότητες είναι : $a_1 = 2$, $a_2 = 1$. Όπως βλέπουμε, χωρίς έλεγχο ροής ένας χρήστης κερδίζει αποκλειστική εξυπηρέτηση, ενώ με έλεγχο ροής η εξυπηρέτηση των δύο ροών προσεγγίζει την γραμμή $r_2 = \frac{a_2}{a_1} r_1$.



Σχήμα 7:

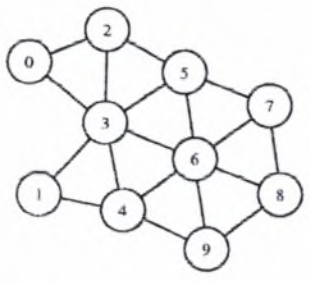
Στο σχήμα 7, δοκιμάζεται ο έλεγχος ροής για βεβαρυμένο άθροισμα throughputs . Για δεδομένους ρυθμούς που βρίσκονται πάνω από την Pareto surface , ρυθμοί εξυπηρέτησης προσεγγίζουν το σημεί για το οποίο το $a_1 r_1 + a_2 r_2$ γίνεται μέγιστο. Όπως βλέπουμε η απόδοση επηρεάζεται και από την παράμετρο K.

5.5 multi-hop case

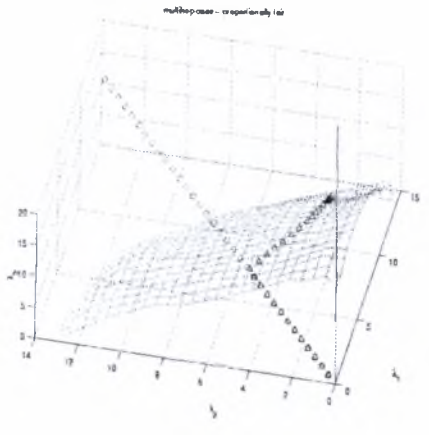
Μελετούμε την απόδοση ενός multi-hop δικτύου με 10 κόμβους όπως αποικονίζεται στο σχήμα φιγυρε 8. Τρεις ροές προέρχονται από το χρήστη 0 στο χρήστη 8, από το χρήστη 1 στο χρήστη 7,από το χρήστη 5 στο χρήστη 4,. Η capacity region για αυτή την απλή περίπτωση υπολογίζεται, και επομένως ο έλεγχος ροής για την multi-hop περίπτωση μπορεί να δοκιμαστέι. Το σχήμα 9 αποικονίζει παρόμοια αποτελέσματα για proportional fairness , ενώ το σχήμα 10 για βεβαρυμένο άθροισμα throughputs . Και στις δυο περιπτώσεις μία ακολουθία ρυθμών αφίξεων εισέρχεται και δοκιμάζεται. Οι ρυθμοί εξυπηρέτησης ταυτίζονται με τους ρυθμούς άφιξης όταν είναι εφικτοί, αλλιώς προσεγγίζουν το εκτιμώμενο δίκαιο σημείο.

6 Συμπεράσματα

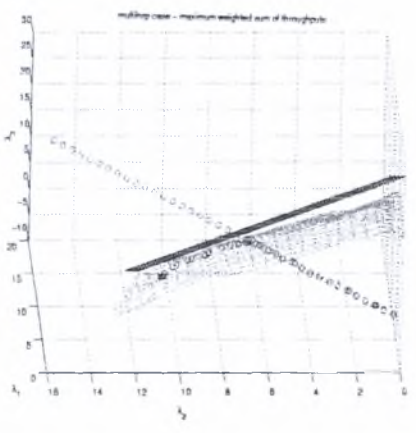
Προτείναμε έναν κατανεμημένο αλγόριθμο ελέγχου ισχύος για ασύρματα ad-hoc δίκτυα, με σκοπό να μεγιστοποιήσουμε το throughput του δικτύου. Στην περίπτωση υψηλών SINR , αυτό όντως συμβαίνει, εκτιμώντας την χωρητικότητα του καναλιού με $C_i(\gamma_i(\mathbf{p})) = \log(\gamma_i(\mathbf{p}))$. Οι χρήστες χρεώνουν ο ένας τον άλλον μια τιμή κόστους σχετιζόμενη με την παραγόμενη παρεμ-



Σχήμα 8:



Σχήμα 9:



Σχήμα 10:

βολή, και βέλτιστες αλλαγές στην ισχύ είναι η λύση ενός convex optimization προβλήματος. Η ουρά του κάθε χρήστη περιέχει κίνηση με διαφορετικούς προορισμούς και οι αποφασεις βασίζονται στα μήκη ουρών. Αποφάσεις δρομολόγησης και scheduling παίρνονται με χρήση της back-pressure τεχνικής. Παρουσιάστηκαν τεχνικές ελέγχου ροής που διασφαλίζουν την επιθυμητή λειτουργία του δικτύου για ρυθμούς αφίξεων πέρα από τη περιοχή ευστάθειας του συστήματος. Πειραματικά αποτελέσματα επιβεβαιώνουν την επιθυμητή λειτουργία.

References

[1] L. Tassiulas and A. Ephremides, "Stability Properties of Constrained Queueing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks", IEEE Transactions on Automatic Control, vol.37, no. 12, Dec. 1992.

[2] J. Huang, R.A. Berry, and M.L. Honig, "A Game Theoretic Analysis of Distributed Power Control for Spread Spectrum Ad Hoc Networks", to appear in IEEE International Symposium on Information Theory (ISIT'05), Adelaide, Australia, September 2005.

[3] D. ONeill, D. Julian, and S. Boyd, "Seeking Foschini's Genie: Optimal Rates and Powers in Wireless Networks", IEEE Transactions on Vehicular Technology, 2004.

[4] M. Chiang, "Balancing Transport and Physical Layers in Wireless Multihop Networks: Jointly Optimal Congestion Control and Power Control", IEEE Journal on Selected Areas in Communications, vol. 23, no. 1, Jan. 2005.

[5] V. Tsibonis, L. Georgiadis, L. Tassiulas, "Exploiting Wireless Channel State Information for Throughput Maximization", IEEE Transactions on Information Theory, vol. 50, no. 11, November 2004, pp 2566-2582. Auto i to antistoixo infocom03?

[6] S.H. Low and D.E. Lapsley, "Optimization Flow Control, I: Basic Algorithm and Convergence", IEEE/ACM Transactions on Networking, 7(6), pp. 861-874, Dec. 1999.

[7] X. Qiu, and K. Chawla, "On the Perfomance of Adaptive Modulation in Cellular Systems", IEEE Transactions on Communications, vol. 47, no. 6, Jun. 1999.

[8] C.W. Sung and K.K. Leung, "Opportunistic power control for throughput maximization in mobile cellular systems", IEEE Proc. ICC, pp. 2954-2958, Paris, France, Jun. 2004.

[9] A. Eryilmaz and R. Srikant, "Fair Resource Allocation in Wireless Networks using Queue-length-based Scheduling and Congestion Control," Proceedings of IEEE INFOCOM, Miami, FL, March 2005.

```
import java.util.Vector;

public class AdHocNetwork{

   int nofUsers;
   int nofTransmissions;

   Links links;
   Vector users;

   TrafficMatrix tm;
   Transmissions transmissions;

   public AdHocNetwork(int nofUsers,Transmissions t,TrafficMatrix matrix,UserPositions up){
      this.nofUsers=nofUsers;

      tm=matrix;
      transmissions=t;

      users=new Vector();
      createSpecificUsers(up);

      nofTransmissions=t.getNofTransmissions();
      transmissions.printTransmissions();

      links=new Links(transmissions,users);
      links.printLinks();
   }


   public void createUsers(){
      for (int i=0; i<nofUsers; i++){
         User u = new User(i,nofUsers,tm);
         users.add(u);
      }
   }

   public void createSpecificUsers(UserPositions up){

      for (int i=0; i<up.getNofUsers(); i++){
         User ui = new User(i,(float) up.getX(i),up.getY(i),nofUsers,tm);
         users.add(ui);
   // User u2 = new User(1,(float) 0.8,(float) 0.1,nofUsers,tm);
   // users.add(u2);
   // User u3 = new User(2,(float) 1.1,(float) 0.1,nofUsers,tm);
   // users.add(u3);
      }


   }

   public boolean areNeighbours(int from,int to){

      if (from==to) {
         return false;
      }

      User u1=(User) users.elementAt(from);
      User u2=(User) users.elementAt(to);

      float distanceSquared=(float)
(Math.pow((u1.getXPos()-u2.getXPos()),2)+Math.pow((u1.getYPos()-u2.getYPos()),2));
      float d4=(float) Math.pow(distanceSquared,2);

      return (1/d4)>15;
   }

   public void printNetwork(){
```

```
        for (int i=0; i<nofUsers; i++){
            User u = (User) users.elementAt(i);
            System.out.println("User #"+u.getNo()+" at ("+u.getXPos()+","+u.getYPos()+")");
            for (int j=0; j<nofUsers; j++){
                User uj = (User) users.elementAt(j);
                    if (areNeighbours(i,j)){
                        System.out.println(i+"->"+j);
                    }
            }
        }
    }

    public void updateTransmissions(Transmissions trans){
        transmissions=trans;
        links=new Links(transmissions,users);
        nofTransmissions=transmissions.getNofTransmissions();
    }

    public static void main(String args[]){

    }
}
```

```
public class UserPositions{

   int maxNofUsers=100;

   int nofUsers;
   float xy[][];

   public UserPositions(){
      xy=new float[maxNofUsers][2];
   }

   public void addUser(float x,float y){

      xy[nofUsers][0]=x;
      xy[nofUsers][1]=y;
      nofUsers++;
   }

   public float getX(int user){
      return xy[user][0];
   }

   public float getY(int user){
      return xy[user][1];
   }

   public int getNofUsers(){
      return nofUsers;
   }
}
```

```
import java.lang.Math;
import java.util.*;

public class User{

    static final float SIZE_OF_PACKET = (float) 1;

    static final int MAX_NOF_FLOWS_FOR_SAME_USER=2;

    int no;
    float xPos;
    float yPos;

    Vector myBacklogs[][];
    float stohasticQueue[][];
    float myReceivedData[][];
    float lastArrival[][];
    float arrivalRates[][];

    float Xsl[][];

    float ai[][];

    public User(int no,int totalUsers,TrafficMatrix tm){
        this.no=no;
        xPos=(float) Math.random()*2;
        yPos=(float) Math.random()*2;

        myBacklogs=new Vector[totalUsers][MAX_NOF_FLOWS_FOR_SAME_USER];


        arrivalRates=new float[totalUsers][MAX_NOF_FLOWS_FOR_SAME_USER];
        lastArrival=new float[totalUsers][MAX_NOF_FLOWS_FOR_SAME_USER];
        myReceivedData=new float[totalUsers][MAX_NOF_FLOWS_FOR_SAME_USER];


        Xsl=new float[totalUsers][MAX_NOF_FLOWS_FOR_SAME_USER];
        ai=new float[totalUsers][MAX_NOF_FLOWS_FOR_SAME_USER];
        stohasticQueue=new float[totalUsers][MAX_NOF_FLOWS_FOR_SAME_USER];

        for (int i=0; i<totalUsers; i++){
            for (int j=0; j<MAX_NOF_FLOWS_FOR_SAME_USER; j++){
                arrivalRates[i][j]=tm.getTraffic(no,i,j);
                myBacklogs[i][j]=new Vector();
            }
        }
        initPriorities();
    }

    public User(int no,float x,float y,int totalUsers,TrafficMatrix tm){
        this.no=no;
        xPos=x;
        yPos=y;

        myBacklogs=new Vector[totalUsers][MAX_NOF_FLOWS_FOR_SAME_USER];
        arrivalRates=new float[totalUsers][MAX_NOF_FLOWS_FOR_SAME_USER];
        lastArrival=new float[totalUsers][MAX_NOF_FLOWS_FOR_SAME_USER];
        myReceivedData=new float[totalUsers][MAX_NOF_FLOWS_FOR_SAME_USER];

        Xsl=new float[totalUsers][MAX_NOF_FLOWS_FOR_SAME_USER];
        ai=new float[totalUsers][MAX_NOF_FLOWS_FOR_SAME_USER];
        stohasticQueue=new float[totalUsers][MAX_NOF_FLOWS_FOR_SAME_USER];

        for (int i=0; i<totalUsers; i++){
            for (int j=0; j<MAX_NOF_FLOWS_FOR_SAME_USER; j++){
                arrivalRates[i][j]=tm.getTraffic(no,i,j);
                myBacklogs[i][j]=new Vector();
            }
        }
```

```
    initPriorities();
  }

  public int getNo(){
    return no;
  }

  public float getXPos(){
    return xPos;
  }

  public float getYPos(){
    return yPos;
  }

  public float getW(int ofClass,int nofFlow){
    return myBacklogs[ofClass][nofFlow].size()*SIZE_OF_PACKET;
  }

  public float getTotalW(){
    float sum=0;
    for (int i=0; i<ai.length; i++){
      for (int j=0; j<MAX_NOF_FLOWS_FOR_SAME_USER; j++){
        sum+=getW(i,j);
      }
    }
    return sum;
  }


  public void addPacket(int ofClass,int flowId){
    stohasticQueue[ofClass][flowId]+=SIZE_OF_PACKET;
  }

  public void addPackets(int nofPackets,int ofClass,int flowId){
    stohasticQueue[ofClass][flowId]+=(nofPackets*SIZE_OF_PACKET);
  }

  public float getLastArrival(int forClass,int flowId){
    return lastArrival[forClass][flowId];
  }

  public void setNewArrival(float newInterArrivalTime,int forClass,int flowId){
    lastArrival[forClass][flowId]+=newInterArrivalTime;
  }

  public float getArrivalRate(int forClass,int flowId){
    return arrivalRates[forClass][flowId];
  }

  public void setArrivalRate(float lamda,int forClass,int flowId){
    arrivalRates[forClass][flowId]=lamda;
  }

  public Vector serveData(int t,float data,int forClass,int nofFlow){
    //System.out.println("before="+myBacklogs[3][0]);
    //System.out.println("before="+myBacklogs[2][0]);
    Vector dataToSend=new Vector();

    if (data>myBacklogs[forClass][nofFlow].size()*SIZE_OF_PACKET){

      for (int i=0; i<myBacklogs[forClass][nofFlow].size(); i++){
        Packet p = (Packet) myBacklogs[forClass][nofFlow].elementAt(0);
        p.setArrivalTime(t);
        dataToSend.add(p);
      }
      myBacklogs[forClass][nofFlow]=new Vector();
    }
    else {
```

```
        for (int i=0; i<=data/SIZE_OF_PACKET-1; i++){
            Packet p = (Packet) myBacklogs[forClass][nofFlow].elementAt(0);
            p.setArrivalTime(t);
            dataToSend.add(p);
            myBacklogs[forClass][nofFlow].remove(0);
        }
    }
    //System.out.println("after="+myBacklogs[3][0]);
    //System.out.println("after="+myBacklogs[2][0]);

    return dataToSend;
}

public void receiveData(float data,int fromClass,int nofFlow,Vector sentData){


    if (fromClass==no){


        for (int i=0; i<=data/SIZE_OF_PACKET-1; i++){
            myReceivedData[((Packet) sentData.elementAt(i)).getOrigin()][nofFlow]+=SIZE_OF_PACKET;
        }
//      System.out.println("receiving "+((int) (data/SIZE_OF_PACKET)));
    }
    else {

        for (int i=0; i<=data/SIZE_OF_PACKET-1; i++){

            myBacklogs[fromClass][nofFlow].add((Packet) sentData.elementAt(i));
        }
    }

}

public float getReceivedData(int origin,int nofFlow){
    return myReceivedData[origin][nofFlow];
}

public void setAi(float a,int dest,int nofFlow){
    ai[dest][nofFlow]=a;
}

public float getAi(int dest,int nofFlow){
    return ai[dest][nofFlow];
}

public float getXs(int dest,int nofFlow){
    return Xsl[dest][nofFlow];
}

public void setXs(float value,int dest,int nofFlow){
    Xsl[dest][nofFlow]=value;
}

public void setXsl(float Xs,int dest,int nofFlow){
    Xsl[dest][nofFlow]=Xs;
}

public float getXsl(int dest,int nofFlow){
    return Xsl[dest][nofFlow];
}

public float getSizeOfPacket(){
    return SIZE_OF_PACKET;
}

public void printQueues(){
    System.out.print(no+" has [");
```

```
    for (int i=0; i<myBacklogs.length; i++){
        System.out.print(myBacklogs[i]+" ");
    }
    System.out.println("]");
}

public float getSumStohasticQueueLength(){
    float sum=0;
    for (int i=0; i<ai.length; i++){
        for (int j=0; j<MAX_NOF_FLOWS_FOR_SAME_USER; j++){
            sum+=stohasticQueue[i][j];
        }
    }
    return sum;
}

public float getStohasticQueueLength(int dest,int flowId){
    return stohasticQueue[dest][flowId];
}

public void setStohasticQueueLength(float value,int dest,int flowId){
    stohasticQueue[dest][flowId]=value;
}

public void flowData(int t){

    if (no==0 ||no==1){
//      System.out.println("W="+getW(2,0)+" xs="+getXs(2,0));
//      System.out.println("W="+getW(3,0)+" xs="+getXs(3,0));
    }
    if (getNo()==0){
//      System.out.println("before B="+stohasticQueue[8][0]+" W="+myBacklogs[8][0].size());
    }

    for (int i=0; i<Xsl.length; i++){

        for (int flowNo=0; flowNo<MAX_NOF_FLOWS_FOR_SAME_USER; flowNo++){

            if (Xsl[i][flowNo]>stohasticQueue[i][flowNo]){

                int cnt=0;
                for (int j=0; j<=stohasticQueue[i][flowNo]/SIZE_OF_PACKET-1; j++){
                    Packet p = new Packet(SIZE_OF_PACKET,t,i,no,flowNo);
                    myBacklogs[i][flowNo].add(p);
                    cnt++;
        //          System.out.println("flow");
                }
                stohasticQueue[i][flowNo]-=cnt*SIZE_OF_PACKET;
            }
            else{
                for (int j=0; j<=Xsl[i][flowNo]/SIZE_OF_PACKET-1; j++){
                    Packet p = new Packet(SIZE_OF_PACKET,t,i,no,flowNo);
                    myBacklogs[i][flowNo].add(p);
                    stohasticQueue[i][flowNo]-=SIZE_OF_PACKET;
        //          System.out.println("flow");
                }
            }
        }
    }
    if (no==0 ||no==1){
//      System.out.println("W="+getW(2,0)+" xs="+getXs(2,0));
//      System.out.println("W="+getW(3,0)+" xs="+getXs(3,0));
    }

    if (getNo()==0){
    //  System.out.println("after B="+stohasticQueue[8][0]+" W="+myBacklogs[8][0].size());
    }
}
```

```
    public void initPriorities(){
        for (int i=0; i<ai.length; i++){
            for (int j=0; j<MAX_NOF_FLOWS_FOR_SAME_USER; j++){
                ai[i][j]=1;
            }
        }
    }

    public void printQueues2(){
        for (int i=0; i<ai.length; i++){
            for (int j=0; j<MAX_NOF_FLOWS_FOR_SAME_USER; j++){
            //  if (myBacklogs[i][j].size()!=0){
                    System.out.println("has i="+i+"j="+j+((myBacklogs[i][j]).size()));
            //  }
            }
        }
    }
}
```

```
public class Transmissions{

   int nofUsers;
   int nofCurrentTransmissions;
   int transmissions[][];

   public Transmissions(int nofUsers){
      this.nofUsers=nofUsers;
      nofCurrentTransmissions=0;
      transmissions=new int[nofUsers][nofUsers];
      clearAllTransmissions();
   }

   public void setTransmission(int from,int to){
      if (from!=to){
         transmissions[from][to]=nofCurrentTransmissions;
         nofCurrentTransmissions++;
      }
   }

   public void stopTransmission(int from,int to){
      transmissions[from][to]=-1;
   }

   public void clearAllTransmissions(){
      nofCurrentTransmissions=0;
      for (int from=0; from<nofUsers; from++){
         for (int to=0; to<nofUsers; to++){
            transmissions[from][to]=-1;
         }
      }
   }

   public boolean thereIsTransmission(int from,int to){
      return (transmissions[from][to]!=-1);
   }

   public int getTransmissionsNo(int from,int to){
      return transmissions[from][to];
   }

   public int getNofTransmissions(){
      return nofCurrentTransmissions;
   }

   public int getFromNode(int transmission){

      for (int from=0; from<nofUsers; from++){
         for (int to=0; to<nofUsers; to++){

            if (getTransmissionsNo(from,to)==transmission){
               return from;
            }
         }
      }
      return -1;
   }

   public int getToNode(int transmission){
      for (int from=0; from<nofUsers; from++){
         for (int to=0; to<nofUsers; to++){
            if (getTransmissionsNo(from,to)==transmission){
               return to;
            }
         }
      }
      return -1;
   }
```

```
    public void setFromNode(int transmission,int newFrom){

        int from = getFromNode(transmission);
        int to = getToNode(transmission);
        transmissions[from][to]=-1;
        transmissions[newFrom][to]=transmission;
    }

    public void setToNode(int transmission,int newTo){

        int from = getFromNode(transmission);
        int to = getToNode(transmission);
        transmissions[from][to]=-1;
        transmissions[from][newTo]=transmission;
    }

    public void printTransmissions(){
        System.out.println("TRANSMISSIONS");
        for (int i=0; i<nofUsers; i++){
            for (int j=0; j<nofUsers; j++){
                if (transmissions[i][j]!=-1){
                    System.out.print(i+"->"+j+"="+transmissions[i][j]+"  ");
                }
            }
            //System.out.println();
        }
        //System.out.println();
    }
}
```

```
public class TrafficMatrix{

   static final int MAX_NOF_FLOWS_FOR_SAME_USER=2;

   float traffic[][][];


   public TrafficMatrix(int nofUsers){
      traffic = new float[nofUsers][nofUsers][MAX_NOF_FLOWS_FOR_SAME_USER];
   }

   public void setTraffic(int fromUser,int toUser,int flowId,float aij){
      if (fromUser!=toUser){
         traffic[fromUser][toUser][flowId]=aij;
      }
   }

   public float getTraffic(int fromUser,int toUser,int flowId){
      return traffic[fromUser][toUser][flowId];
   }
}
```

```
import java.util.Vector;

public class TrafficArranger{

   int END_OF_SIMULATION;
   float MAX_XS=100;

   Vector users;

   int t=0;
   boolean first=true;

   float k = (float) 100;


   public TrafficArranger(Vector users,int simulationTime){
      this.users=users;
      END_OF_SIMULATION=simulationTime;
   }

   public void makeAllArrivals(){

      for (int i=0; i<users.size(); i++){

         User ui=(User) users.elementAt(i);

         for (int j=0; j<ui.arrivalRates.length; j++){

            for (int k=0; k<ui.MAX_NOF_FLOWS_FOR_SAME_USER; k++){

               float lamda_ijk=ui.getArrivalRate(j,k);

               int cnt=0;

               if (ui.getLastArrival(j,k)<t+1 & ui.getLastArrival(j,k)!=0){
                  cnt++;
               }

               while (ui.getLastArrival(j,k)<t+1){

                  float uni=(float) Math.random();
                  float exp=(float) (-Math.log(uni)/lamda_ijk);



                  ui.setNewArrival(exp,j,k);

                  cnt++;

   //               System.out.println("t="+t+" #"+i+" -> "+ui.getLastArrival(j,k));
               }
               if (cnt!=0){
                  ui.addPackets(cnt-1,j,k);
                  if (cnt-1!=0){
   //                System.out.println("adding "+(cnt-1));
                  }
               }
            }
         }

         //ui.printQueues();
      }
   }

   public void makeBurstyArrivals(){

      /*NOT ADAPTED TO NxN
      if (t%10000==0){
         if (first==true){
```

```
            first=false;
            User ui=(User) users.elementAt(transmissions.getFromNode(1));
            ui.setNewArrival(t-ui.getLastArrival());
        }
        else {
            first=true;
            User ui=(User) users.elementAt(transmissions.getFromNode(0));
            ui.setNewArrival(t-ui.getLastArrival());
        }
    }



    User ui=(User) users.elementAt(transmissions.getFromNode((first==true)?0:1));
    float lamda_i=ui.getArrivalRate();
    int cnt2=0;

    if (ui.getLastArrival()<t+1 & ui.getLastArrival()!=0){
        cnt2++;
    }

    while (ui.getLastArrival()<t+1){
        float uni=(float) Math.random();
        float exp=(float) (-Math.log(uni)/lamda_i);


        ui.setNewArrival(exp);

        cnt2++;


        //System.out.println("t="+t+" #"+first+" -> "+ui.getLastArrival());
    }
    if (cnt2!=0){
        ui.addPackets(cnt2-1);
    }
    */
}

public void serveLinks(Links links,PowerPriceStrategy pps,MulticlassActivationVector mav){

    if (pps.nofTransmissionPairs==0){
        return;
    }

    for (int i=0; i<users.size(); i++){

        User ui=(User) users.elementAt(i);

        for (int j=0; j<users.size(); j++){

            User uj=(User) users.elementAt(j);

            if (mav.getActivatedClass(i,j)!=-1){

                int classNo=mav.getActivatedClass(i,j);
                int correspondingFlow=mav.getCorrespondingFlow(i,j);

                User toi=(User) users.elementAt(j);
                int transmission=links.transmissions.getTransmissionsNo(ui.getNo(),toi.getNo());

                if (transmission!=-1){

                    float SINRi = links.calculateSINRatReceiver(transmission,pps);
                    float C_i = (float) (Math.log(SINRi)/Math.log(2));
                    float diff=ui.getW(classNo,correspondingFlow)-uj.getW(classNo,correspondingFlow);

            //      System.out.println("i="+i+" SINR="+SINRi+" Ci="+C_i+" i="+i+" j="+j+"
ui="+ui.getW(classNo,correspondingFlow)+" j"+uj.getW(classNo,correspondingFlow));
```

```
                float servingData;
                if (diff<C_i){
                   servingData=diff;
                }
                else {
                   servingData=C_i;
                }


                if (C_i>=0 & ui.getW(classNo,correspondingFlow)>0){

                   Vector data=new Vector();
                   //System.out.println("C_i="+C_i);
                   //System.out.println("before W="+ui.getW(classNo,correspondingFlow)+" C_i="+C_i);
                   if (C_i<diff){
                      data=ui.serveData(t,C_i,classNo,correspondingFlow);
                   }
                   else {
                      data=ui.serveData(t,diff,classNo,correspondingFlow);
                   }
                //  System.out.println("DATA SIZE="+data.size());

                   if (C_i<diff){
                      toi.receiveData(C_i,classNo,correspondingFlow,data);
                   }
                   else {
                      toi.receiveData(servingData,classNo,correspondingFlow,data);
                   }
                   if (ui.getW(classNo,correspondingFlow)<0){
                      System.out.println("after W="+ui.getW(classNo,correspondingFlow)+" C_i="+C_i);
                   }

//                 System.out.println("t="+t+" #"+i+" serving "+C_i);
                }
                else {
        //         System.out.println("t="+t+" #"+i+" serving ---");
                }
             }
          }
        }

        //****
        //updateXS


        for (int dest=0 ; dest<users.size(); dest++){

        //IS IT THAT FOR MULTIHOP??
           for (int flowId=0 ; flowId<ui.MAX_NOF_FLOWS_FOR_SAME_USER; flowId++){

           //  float newXs=Float.MAX_VALUE;

              int testRun=0;
              //0:no flow control
              //1:pf
              //2:geo


              float newXs;

              if (testRun==0){
                 newXs=MAX_XS;
              }
              else if (testRun==1){

                 //PROPORTIONAL
                 float akj=ui.getAi(dest,flowId);
//               System.out.println("me="+ui.getNo()+" a="+akj+" d="+dest+" flowId="+flowId);
```

```
                float Wkj=ui.getTotalW();

                newXs=akj*k/Wkj;
                if (!(newXs<MAX_XS)){
                   newXs=MAX_XS;
                }
             }
             else {



                float akj=ui.getAi(dest,flowId);
          //    System.out.println("me="+ui.getNo()+" a="+akj+" d="+dest+" flowId="+flowId);

                float Wkj=ui.getTotalW();
          //    System.out.println("W="+Wkj);

                if (Wkj<akj*k){
                   newXs=ui.getSumStohasticQueueLength();
                }
                else {
                   newXs=0;
                }

                if (!(newXs<MAX_XS)){
                   newXs=MAX_XS;
                }

                if (ui.getSumStohasticQueueLength()!=0){
                // System.out.println(ui.getSumStohasticQueueLength());
                }

             }

             ui.setXs(newXs,dest,flowId);

             if (i==0 && dest==3 && flowId==0){
   //           System.out.println(i+" new Xs="+newXs+" W="+Wkj);
             }
             if (i==1 && dest==2 && flowId==0){
      //        System.out.println(i+" new Xs="+newXs+" W="+Wkj);
             }

          }
       }
    }
}

public void procceedT(Links links,PowerPriceStrategy pps,MulticlassActivationVector mav){

   //System.out.println();

   makeAllArrivals();
   serveLinks(links,pps,mav);


   for (int i=0; i<users.size(); i++){

      //****
      //updateValve
      User ui=(User) users.elementAt(i);
   // System.out.println(i+" "+ui.getStohasticQueueLength()+" "+ui.getW()+" "+ui.getXs());
      ui.flowData(t);
   // System.out.println(i+" "+ui.getStohasticQueueLength()+" "+ui.getW()+" "+ui.getXs());

      //System.out.println(i+"'s queue lenth= "+ui.getW());
      //System.out.println(i+"'received= "+ut.getReceivedData());
//    System.out.println(i+" "+ui.getW()+ui.getXs());
   }
```

```
        //makeBurstyArrivals();



        t++;
        if (t%1000==0){
        //  System.out.println("t="+t);
        }
    }

}
```

```
import java.io.*;

public class Simulation{

   float sizeOfPacket;
   TrafficMatrix tm;

   public Simulation(){

   }

   public void simulationA(String outputFile,int nofIterations,int utilityType,int subType,float Pmax,int
simulationTime,boolean fixedBacklogs,boolean simulatingExpo){

      UserPositions up2=new UserPositions();
      up2.addUser((float) 0.8,(float) 0.9);
      up2.addUser((float) 0.8,(float) 0.3);
      up2.addUser((float) 1.1,(float) 1.2);
      up2.addUser((float) 1.1,(float) 0.7);
      up2.addUser((float) 1.1,(float) 0.2);
      up2.addUser((float) 1.4,(float) 1.0);
      up2.addUser((float) 1.4,(float) 0.5);
      up2.addUser((float) 1.7,(float) 0.9);
      up2.addUser((float) 1.7,(float) 0.4);
      up2.addUser((float) 1.4,(float) 0.1);


      float lambda1=(float) 0;
      float lambda2=0;
      float lambda3=0;

      float step=(float) 1.5;
      float diff=(float) 0.5;
      float perc=(float) 0.4;

      int nofUsers=up2.getNofUsers();
      tm=new TrafficMatrix(nofUsers);
      tm.setTraffic(0,8,0,(float) 0.3);
      tm.setTraffic(1,7,0,(float) 0.3);
      tm.setTraffic(5,4,0,(float) 0.3);

      Ours test5 = new
Ours(nofUsers,simulationTime,tm,up2,1,utilityType,subType,Pmax,fixedBacklogs,simulatingExpo);

      sizeOfPacket=((User) test5.network.users.elementAt(0)).getSizeOfPacket();

   // System.out.println("m1:"+test5.getService1()/sizeOfPacket);
   // System.out.println("m2:"+test5.getService2()/sizeOfPacket);
      float C11=test5.getC11();


      FileWriter fw=null;
      try {
         fw = new FileWriter(outputFile);
      }
      catch (IOException ioe){
         System.err.println("ioexc"+ioe.getMessage());
      }

      while (true){


         if (lambda1==0 && lambda2==0 && lambda3!=0){
            break;
         }


         lambda1=0;
         lambda2=0;
```

```
        while (true){

            if (lambda2==0 && lambda1!=0){
//              System.out.println("C11="+test.getC11());
//              System.out.println("C01="+test.getC01());
                break;
            }


            lambda2=0;

            tm=new TrafficMatrix(nofUsers);
            tm.setTraffic(0,8,0,(float) lambda1);
            tm.setTraffic(1,7,0,(float) lambda2);
            tm.setTraffic(5,4,0,(float) lambda3);


            Ours test = new
Ours(nofUsers,simulationTime,tm,up2,nofIterations,utilityType,subType,Pmax,fixedBacklogs,simulatingExpo);

//          currentService1=((User)
test.network.users.elementAt(8)).getReceivedData(0,0)/(sizeOfPacket*simulationTime);

//          System.out.println("curr="+currentService1+" lastServ="+lastService1with2at0);
//          System.out.println("l1="+lambda1);

            while (true){

                tm.setTraffic(0,8,0,(float) lambda1);
                tm.setTraffic(1,7,0,(float) lambda2);
                tm.setTraffic(5,4,0,(float) lambda3);
                Ours h = new
Ours(nofUsers,simulationTime,tm,up2,nofIterations,utilityType,subType,Pmax,fixedBacklogs,simulatingExpo);


                float service1=((User) h.network.users.elementAt(8)).getReceivedData(0,0)/(sizeOfPacket*simulationTime);
                float service2=((User) h.network.users.elementAt(7)).getReceivedData(1,0)/(sizeOfPacket*simulationTime);
                float service3=((User) h.network.users.elementAt(4)).getReceivedData(5,0)/(sizeOfPacket*simulationTime);

                System.out.println("m1="+service1+" at l1="+lambda1);
                System.out.println("m2="+service2+" at l2="+lambda2);
                System.out.println("m3="+service3+" at l3="+lambda3);

//              newService2=service2;


//              System.out.println("serv1="+h.getService1());

                try {
                    if
(Math.abs(service1/sizeOfPacket-lambda1)>diff||Math.abs(service1/sizeOfPacket-lambda1)>lambda1*perc){
                        System.out.println("This is it1!");
                        fw.write(""+service1+" "+service2+" "+service3+"\n");
                        fw.flush();
//                      previousService2=-1;
                        break;
                    }

                    if
(Math.abs(service2/sizeOfPacket-lambda2)>diff||Math.abs(service2/sizeOfPacket-lambda2)>lambda2*perc){
                        System.out.println("This is it2!");
                        fw.write(""+service1+" "+service2+" "+service3+"\n");
                        fw.flush();
//                      previousService2=-1;
                        break;
                    }
```

```
                }
                catch (IOException ioe){
                    System.err.println("ioexc"+ioe.getMessage());
                }



//              previousService2=newService2;

                lambda2+=step;
            }
            lambda1+=step;
        }
        lambda3+=step;
    }
}

    public void simulationB(String outputFile,int nofIterations,int utilityType,int subType,float Pmax,int
simulationTime,boolean fixedBacklogs,boolean simulatingExpo){
        /*
        float lambda1=(float) 0;
        float lambda2=0;

        tm.setTraffic(0,3,0,(float) 0.3);
        tm.setTraffic(1,2,0,(float) 0.3);
        Ours test5 = new Ours(4,simulationTime,tm,nofIterations,utilityType,subType,Pmax,fixedBacklogs,simulatingExpo);

        sizeOfPacket=((User) test5.network.users.elementAt(0)).getSizeOfPacket();
    // System.out.println("m1:"+test5.getService1()/sizeOfPacket);
    // System.out.println("m2:"+test5.getService2()/sizeOfPacket);
        float C11=test5.getC11();

        FileWriter fw=null;
        try {
            fw = new FileWriter(outputFile);
        }
        catch (IOException ioe){
            System.err.println("ioexc"+ioe.getMessage());
        }

        lambda1=0;
        lambda2=0;
        for (int i=0; i<20; i++){

            tm.setTraffic(0,3,0,(float) lambda1);
            tm.setTraffic(1,2,0,(float) lambda2);
            Ours h = new Ours(4,simulationTime,tm,nofIterations,utilityType,subType,Pmax,fixedBacklogs,simulatingExpo);

            System.out.println("m1:"+h.getService1()/sizeOfPacket+" at l1="+lambda1);
            System.out.println("m2:"+h.getService2()/sizeOfPacket+" at l2="+lambda2);
            try {
                fw.write(""+h.getService1()/sizeOfPacket+" "+lambda1+"\n");
                fw.flush();
            }
            catch (IOException ioe){
                System.err.println("ioe"+ioe.getMessage());
            }
            if (i==19){
                System.out.println("C11="+h.getC11()+" C10="+h.getC01());
            }

            lambda1+=0.05;
            lambda2+=0.05;
        }
        */
}

    public void simulationC(String outputFile){
```

```
        FileWriter fw=null;
        try {
            fw = new FileWriter(outputFile);
        }
        catch (IOException ioe){
            System.err.println("ioexc"+ioe.getMessage());
        }
    }

    public void simulationD(String outputFileNOO,int nofIterations,int utilityType,int subType,float Pmax,int
simulationTime,boolean fixedBacklogs,boolean simulatingExpo){

        //FlowControl

        String outputFile="multihop - geo - k=1000.txt";

        float step=(float) 0.4;
        float lambda1=0;
        float lambdaSlope=((float) 1);
        float lambda2=0;

        UserPositions up2=new UserPositions();
        up2.addUser((float) 0.8,(float) 0.9);
        up2.addUser((float) 0.8,(float) 0.3);
        up2.addUser((float) 1.1,(float) 1.2);
        up2.addUser((float) 1.1,(float) 0.7);
        up2.addUser((float) 1.1,(float) 0.2);
        up2.addUser((float) 1.4,(float) 1.0);
        up2.addUser((float) 1.4,(float) 0.5);
        up2.addUser((float) 1.7,(float) 0.9);
        up2.addUser((float) 1.7,(float) 0.4);
        up2.addUser((float) 1.4,(float) 0.1);

        int nofUsers=up2.getNofUsers();

        FileWriter fw=null;
        try {
            fw = new FileWriter(outputFile);
        }
        catch (IOException ioe){
            System.err.println("ioexc"+ioe.getMessage());
        }

        for (int i=0; i<30/step; i++){
            lambda1+=step;
            lambda2=lambdaSlope*lambda1;

            tm=new TrafficMatrix(nofUsers);
            tm.setTraffic(0,8,0,(float) lambda1);
            tm.setTraffic(1,7,0,(float) lambda2);
            Ours h = new
Ours(nofUsers,simulationTime,tm,up2,nofIterations,utilityType,subType,Pmax,fixedBacklogs,simulatingExpo);
            sizeOfPacket=((User) h.network.users.elementAt(0)).getSizeOfPacket();
            float a1=((User) h.network.users.elementAt(0)).getAi(8,0);
            float a2=((User) h.network.users.elementAt(1)).getAi(7,0);
            System.out.println(a1+" "+a2);
            float m1=((User) h.network.users.elementAt(8)).getReceivedData(0,0)/(sizeOfPacket*simulationTime);
            float m2=((User) h.network.users.elementAt(7)).getReceivedData(1,0)/(sizeOfPacket*simulationTime);
            System.out.println("m1:"+m1+" at l1="+lambda1);
            System.out.println("m2:"+m2+" at l2="+lambda2);

            try {
                fw.write(a1+" "+a2+" "+m1+" "+m2+" "+lambda1+" "+lambda2+"\n");
                fw.flush();
            }
            catch (IOException ioe){
                System.err.println("ioexc"+ioe.getMessage());
            }
        }
```

```
    }

    public static void main(String args[]){

        int nofIterations=1;
        int utilityType=1;
        int subType=1;
        float Pmax=100;
        int simulationTime=2000;
        boolean backlogsAreFixed=false;
        boolean simulatingExpo=false;

        String path="";

        if (utilityType==1){
            path+="log(SINR) - ";
        }
        else {
            path+="log(SINR+1) - ";
            if (subType==1){
                path+="case1 - ";
            }
            else {
                path+="case2 - ";
            }
        }
        path+="Pmax="+((int) Pmax)+"";

        path+=!backlogsAreFixed?(""):(" - fixed");

        path+=!simulatingExpo?(""):(" - expo");

        String outputPath = "cr - "+path+".txt";

        Simulation s = new Simulation();

s.simulationA(outputPath,nofIterations,utilityType,subType,Pmax,simulationTime,backlogsAreFixed,simulatingExpo);
        //s.simulationB("results.txt",nofIterations,utilityType,subType,Pmax,simulationTime);
        //s.simulationC("results.txt",nofIterations,utilityType,,subType,Pmax,simulationTime);
    //
s.simulationD(outputPath,nofIterations,utilityType,subType,Pmax,simulationTime,backlogsAreFixed,simulatingExpo);
    }
}
```

```
import java.util.Vector;
import java.io.*;

public class PowerPriceStrategy{

   int nofTransmissionPairs;

   float p[];
   float prices[];

   public PowerPriceStrategy(int no){
      nofTransmissionPairs=no;

      p=new float[nofTransmissionPairs];
      prices=new float[nofTransmissionPairs];
   }

   public float getPower(int i){
      return p[i];
   }

   public float getPrice(int i){
      return prices[i];
   }

   public void setPower(int i,float power){
      p[i]=power;
   }

   public void setPrice(int i,float price){
      prices[i]=price;
   }

   public void printStrategy(){
      System.out.print("\nP: ");
      for (int i=0; i<nofTransmissionPairs; i++){
         System.out.print(p[i]+" ");
      }
      System.out.print("\nPrices: ");
      for (int i=0; i<nofTransmissionPairs; i++){
         System.out.print(prices[i]+" ");
      }
      System.out.println("\n");
   }

   public void printToFile(String path,int iteration){
      try {
         FileWriter fw = new FileWriter(path,true);
         fw.write("\ni="+iteration);
         fw.write("\nP: ");
         for (int i=0; i<nofTransmissionPairs; i++){
            fw.write(p[i]+" ");
         }
         fw.write("\nPrices: ");
         for (int i=0; i<nofTransmissionPairs; i++){
            fw.write(prices[i]+" ");
         }
         fw.write("\n");
         fw.flush();
      }
      catch (IOException ioe){
         System.err.println("ioe"+ioe.getMessage());
      }
   }
}
```

```
public class Packet{

   float sizeOfPacket;
   int arrivalTime;
   int classId;
   int flowId;
   int origin;

   public Packet(float sizeOfPacket,int arrivalTime,int classId,int origin,int flowId){
      this.sizeOfPacket=sizeOfPacket;
      this.arrivalTime=arrivalTime;
      this.classId=classId;
      this.origin=origin;
      this.flowId=flowId;
   }

   public int getArrivalTime(){
      return arrivalTime;
   }

   public void setArrivalTime(int t){
      arrivalTime=t;
   }

   public float getSizeOfPacket(){
      return sizeOfPacket;
   }

   public void setSizeOfPacket(float size){
      sizeOfPacket=size;
   }

   public void setClassId(int classNo){
      classId=classNo;
   }

   public int getClassId(){
      return classId;
   }

   public void setOrigin(int or){
      origin=or;
   }

   public int getOrigin(){
      return origin;
   }

   public void setFlowId(int id){
      flowId=id;
   }

   public int getFlowId(){
      return flowId;
   }
}
```

```
import java.lang.Math;
import java.io.*;

public class Ours{

    AdHocNetwork network;
    PowerPriceStrategy pps;
    TrafficArranger ta;

    boolean backlogsAreFixed=false;
    boolean simulatingExpo=false;

    int utilityType=1;
    //1-> ui~=log(SINR)
    //2-> ui~=log(1+SINR)
    int algorithmTypeForUtility2=1;

    float uDerivAtINFTY;
    float uDerivAtZERO;

    float INFINITY=Float.MAX_VALUE;

    float P_MAX;
    float noise;

    float service1,service2;
    float C11,C01;

    int No;

    FileWriter expoFile=null;

    MulticlassActivationVector slotMAV;
    Transmissions transmissions;

    public Ours(int nofUsers,int simulationTime,TrafficMatrix trafficMatrix,UserPositions up,int nofIterations,int
utilityType,int subType,float Pmax,boolean fixedBacklogs,boolean simulatingExpo){

        slotMAV=new MulticlassActivationVector(nofUsers);
        transmissions=new Transmissions(nofUsers);
        network = new AdHocNetwork(nofUsers,transmissions,trafficMatrix,up);
        ta=new TrafficArranger(network.users,simulationTime);



        for (int i=0; i<network.users.size(); i++){

            User ui=(User) network.users.elementAt(i);

            for (int j=0; j<network.users.size(); j++){
                User uj=(User) network.users.elementAt(j);

                if (network.areNeighbours(i,j)){

                    if (!transmissions.thereIsTransmission(j,i)){

                        transmissions.setTransmission(i,j);
                    }
                }
            }
        }
        pps = new PowerPriceStrategy(transmissions.getNofTransmissions());

//      transmissions.printTransmissions();
//      network.links.printLinks();

        findMulticlassActivationVectorAtThisSlot();

        network.updateTransmissions(transmissions);
```

```
        transmissions.printTransmissions();
        network.links.printLinks();
        No=nofIterations;
        this.utilityType=utilityType;
        algorithmTypeForUtility2=subType;
        P_MAX=Pmax;
        noise=network.links.getNoise();
        backlogsAreFixed=fixedBacklogs;
        this.simulatingExpo=simulatingExpo;

        try{
           expoFile = new FileWriter("expo.txt");
        }
        catch(IOException ioe){
           System.err.println("ioe"+ioe.getMessage());
        }



        initialization();
    //  network.printNetwork();


        /*
        float G00=network.links.getGfromTo(0,0);
        float G11=network.links.getGfromTo(1,1);
        float G10=network.links.getGfromTo(1,0);
        float G01=network.links.getGfromTo(0,1);

//      System.out.println("**SNIR="+G00*P_MAX/(noise));

        float C1_max_max= (float) (Math.log(G00*P_MAX/(G10*P_MAX+noise))/Math.log(2));
        float C2_max_max= (float) (Math.log(G11*P_MAX/(G01*P_MAX+noise))/Math.log(2));
        float C1_max_0= (float) (Math.log(G00*P_MAX/(noise))/Math.log(2));
        float C2_0_max= (float) (Math.log(G11*P_MAX/(noise))/Math.log(2));

    //  System.out.println("C1_max_max="+C1_max_max+" C2_max_max="+C2_max_max );
    //  System.out.println("C1_max_0="+C1_max_0+" C2_0_max="+C2_0_max );
        setC11(C1_max_max);
        setC01(C1_max_0);
        */




        while (ta.t<ta.END_OF_SIMULATION){
           //transmissions.printTransmissions();
           for (int i=0; i<No; i++){
           //  System.out.print("ITERATION: "+i);
              doACircle();
           //  pps.printStrategy();
        //    pps.printToFile("output2.txt",i);
           }

    //     pps.printStrategy();
    //     float SINR1=network.links.calculateSINRatReceiver(0,pps);
    //     float SINR2=network.links.calculateSINRatReceiver(1,pps);
           //System.out.println(transmissions.getNofTransmissions());
           ta.procceedT(network.links,pps,slotMAV);
    //     transmissions.printTransmissions();
        //  network.links.printLinks();
           findMulticlassActivationVectorAtThisSlot();

           for (int i=0; i<network.users.size(); i++){
//            System.out.println(i+" 8:"+((User) network.users.elementAt(i)).getW(8,0)+" - 7:"+((User)
network.users.elementAt(i)).getW(7,0));
           }
//         slotMAV.printMAV();
```

```
//          System.out.println();
        // transmissions.printTransmissions();
           //network.links.printLinks();
        }

        for (int i=0; i<transmissions.getNofTransmissions(); i++){
           User toi=(User) network.users.elementAt(transmissions.getToNode(i));
           //System.out.println("i->"+toi.getReceivedData()/ta.END_OF_SIMULATION);

           //FIX OUTPUT:
           /*
           if (i==0){
              setService1(toi.getReceivedData()/ta.END_OF_SIMULATION);
           }
           else if (i==1){
              setService2(toi.getReceivedData()/ta.END_OF_SIMULATION);
           }
           */
        }
     }

     public void initialization(){

        if (simulatingExpo){

           //***
           //Correct init expo values -->
           /*
           User ui1=(User) network.users.elementAt(transmissions.getFromNode(0));
           User ui2=(User) network.users.elementAt(transmissions.getFromNode(1));

           ui1.setAi(13);
           ui2.setAi(2);


           //FIX INIT VALUES:
           //ui1.setBacklog(4000,1);
           //ui2.setBacklog(100,2);

           */
        }
        User ui0=(User) network.users.elementAt(0);
        User ui1=(User) network.users.elementAt(1);
        ui0.setAi(3,8,0);
        ui1.setAi(1,7,0);

        for (int i=0; i<transmissions.getNofTransmissions(); i++){

           float pi0;
           float price0;

           if (i==0){
              pi0=P_MAX;
              price0=0;
           }
           else {
              pi0=0;
              price0=10;
           }

           pps.setPower(i,pi0);
           pps.setPrice(i,price0);

           float Pi=pps.getPower(i);
           float SINRi=network.links.calculateSINRatReceiver(i,pps);

           //System.out.println("Pi = "+Pi+" SINRi="+SINRi+"("+i+")\n");

        }
```

```
        if (utilityType==1){

            uDerivAtINFTY=0;
            uDerivAtZERO=INFINITY;

        }
        else if (utilityType==2){

            uDerivAtINFTY=0;
            //uDerivAtZERO=theta;

        }


    }

    public void findMulticlassActivationVectorAtThisSlot(){

        slotMAV.resetActivations();
        //transmissions.clearAllTransmissions();

        for (int i=0; i<network.users.size(); i++){

            User ui=(User) network.users.elementAt(i);

            for (int j=0; j<network.users.size(); j++){

                User uj=(User) network.users.elementAt(j);

                if (network.areNeighbours(i,j)){

                    float maxDiff=-Float.MAX_VALUE;

                    int correspondingClass=-1;
                    int correspondingFlow=-1;

                    boolean opposite=false;

                    if (!(slotMAV.getActivatedClass(i,j)!=-1 || slotMAV.getActivatedClass(j,i)!=-1)){

                        for (int examingClass=0; examingClass<network.users.size(); examingClass++){

                            for (int examingFlow=0; examingFlow<((User)
network.users.elementAt(0)).MAX_NOF_FLOWS_FOR_SAME_USER; examingFlow++){

                                float diff=ui.getW(examingClass,examingFlow)-uj.getW(examingClass,examingFlow);


                                if (diff>maxDiff && diff!=0){

                                    maxDiff=diff;

                                    correspondingClass=examingClass;
                                    correspondingFlow=examingFlow;
                        //          System.out.println("not OPP: i="+i+" j="+j+" DW="+maxDiff+" diff="+diff+"
Wi="+ui.getW(examingClass,examingFlow)+" Wj="+uj.getW(examingClass,examingFlow));
                                    opposite=false;
                                }

                                if (-diff>maxDiff && diff!=0){
                                    maxDiff=-diff;
                                    correspondingClass=examingClass;
                                    correspondingFlow=examingFlow;
                                    opposite=true;
                        //          System.out.println("OPP: i="+i+" j="+j+" DW="+maxDiff+" diff="+diff+"
Wi="+ui.getW(examingClass,examingFlow)+" Wj="+uj.getW(examingClass,examingFlow));
                                }
```

```
                }
            }

//          System.out.println("final: i="+i+" j="+j+" DW="+maxDiff);
            if (!opposite){

                slotMAV.activateForClass(i,j,correspondingClass,correspondingFlow);


                if (correspondingClass!=-1){

                    if (transmissions.thereIsTransmission(j,i)){
                        int transNo=transmissions.getTransmissionsNo(j,i);
                        //transmissions.setTransmission(i,j);
                        transmissions.setFromNode(transNo,i);
                        transmissions.setToNode(transNo,j);
                    }
                }

            }
            else {

                slotMAV.activateForClass(j,i,correspondingClass,correspondingFlow);

                if (correspondingClass!=-1){

                    if (transmissions.thereIsTransmission(i,j)){
                        int transNo=transmissions.getTransmissionsNo(i,j);
                        //transmissions.setTransmission(i,j);
                        transmissions.setFromNode(transNo,j);
                        transmissions.setToNode(transNo,i);
                    }
                }
            }
        }
    }
}

    network.links.updateLinks(transmissions);
}

public void doACircle(){

    //Transmissions remain the same (as long as I is complete)
    for (int i=0; i<transmissions.getNofTransmissions(); i++){

        float PowerUpdate;
        float priceUpdate;


        float Pi=pps.getPower(i);
        float SINRi=network.links.calculateSINRatReceiver(i,pps);
        float Gii=network.links.getGfromTo(i,i);
        float Ii=network.links.calculateIi(i,pps);


        //System.out.println("Pi = "+Pi+" SINRi="+SINRi+"("+i+")\n");

        User user_i = (User) network.users.elementAt(transmissions.getFromNode(i));
        User user_j = (User) network.users.elementAt(transmissions.getToNode(i));
        int classId = slotMAV.getActivatedClass(transmissions.getFromNode(i),transmissions.getToNode(i));
        int flowId = slotMAV.getCorrespondingFlow(transmissions.getFromNode(i),transmissions.getToNode(i));

        float W_i=0;

        //****
        //EXPO-Rule
```

```
        //Correct W=X or W=X_i-X_j?
        /*
        if (simulatingExpo){

           float ai=user_i.getAi();
           float Xi=user_i.getW(classId);

           float meanAX=0;

           try{
              for (int k=0; k<transmissions.getNofTransmissions(); k++){
                 User user_k = (User) network.users.elementAt(transmissions.getFromNode(k));
                 int classIdK = slotMAV.getActivatedClass(transmissions.getFromNode(k),transmissions.getToNode(k));

                 meanAX+=user_k.getAi()*user_k.getW(classIdK);

                 expoFile.write(user_k.getW(classIdK)+" ");
              }
              meanAX=meanAX/transmissions.getNofTransmissions();

              expoFile.write("\n");
              expoFile.flush();

           }
           catch (IOException ioe){
              System.err.println("ioexc"+ioe.getMessage());
           }


           W_i=(float) (Math.exp((ai*Xi-meanAX)/(1+Math.sqrt(meanAX))));

        */
        //*****


        if (!backlogsAreFixed){


           if (classId==-1 ||flowId==-1){
              //Idle transmission
              W_i = 0;
           }
           else{
              W_i = user_i.getW(classId,flowId)-user_j.getW(classId,flowId);
           }

           if (W_i<0){

              //Already moved
              //Not ideal activationVector
        //    System.out.println("from has"+user_i.getW(classId,flowId));
        //    System.out.println("to has"+user_j.getW(classId,flowId));
              W_i=0;
           }
        }
        else {
           W_i=1;
        }


        if (utilityType==2){
           uDerivAtZERO=W_i;
        }


        //**********
        //PRICES

        float PriceUpdate=0;
```

```
        if (utilityType==1){
            PriceUpdate=W_i/(Ii+noise);
        }
        else if (utilityType==2){
            PriceUpdate=W_i*SINRi/((SINRi+1)*(Ii+noise));
        }
        //System.out.println("gamma="+SINRi+" Ii="+Ii+"Wi="+W_i);
        pps.setPrice(i,PriceUpdate);

        //
        //****

        //*****
         //POWER

        float S_pi_jh_ij=0;
        for (int j=0; j<transmissions.getNofTransmissions(); j++){
            if (i!=j){
                float pi_j=pps.getPrice(j);
                float h_ij=network.links.getGfromTo(i,j);

                //System.out.println("pi_j = "+pi_j+" h_ij="+h_ij+" ("+i+")");

                S_pi_jh_ij+= pi_j*h_ij;
            }
        }
        //System.out.println("Sum="+S_pi_jh_ij+"\n");

        float x=S_pi_jh_ij;

        float gx=0;

        if (x<=uDerivAtINFTY){
//          System.out.println("1");
            gx=INFINITY;
        }
        else if((uDerivAtINFTY<x) & (x<uDerivAtZERO)){

//          System.out.println("2");
            if (utilityType==1){
                gx=W_i/x;
            }
            else if (utilityType==2){

                if (algorithmTypeForUtility2==1){
                    gx=W_i*SINRi/((SINRi+1)*x);
                }
                else if (algorithmTypeForUtility2==2){
                    gx=W_i/x - (Ii+noise)/Gii;
                }
            }
        }
        else if (uDerivAtZERO<x){
//          System.out.println("3");
            gx=0;
        }

        //System.out.println("x= "+x+" gx = "+gx);
        //System.out.println("gamma="+SINRi+" Ii="+Ii+"Wi="+W_i);
        float proposedValue = gx;

        if (W_i==0){
//          System.out.println("prop = "+proposedValue+" "+i);
        }

        PowerUpdate = Math.min(proposedValue,P_MAX);
        pps.setPower(i,PowerUpdate);
```

```
        //
        //*******

    }
}

public float getService1(){
    return service1;
}
public float getService2(){
    return service2;
}

public void setService1(float f){
    service1=f;
}
public void setService2(float f){
    service2=f;
}

public float getC11(){
    return C11;
}

public float getC01(){
    return C01;
}

public void setC11(float c){
    C11=c;
}

public void setC01(float c){
    C01=c;
}

public static void main(String args[]){

    int nofIterations=1;
    int utilityType=1;
    int subType=1;
    float Pmax=100;
    int simulationTime=10000;
    boolean backlogsAreFixed=false;
    boolean simulatingExpo=false;

    UserPositions up1=new UserPositions();

    up1.addUser((float) 0.8,(float) 1.2);
    up1.addUser((float) 1.1,(float) 0.1);
    up1.addUser((float) 0.8,(float) 0.1);
    up1.addUser((float) 1.1,(float) 1.2);

    UserPositions up2=new UserPositions();

    up2.addUser((float) 0.8,(float) 0.9);
    up2.addUser((float) 0.8,(float) 0.3);
    up2.addUser((float) 1.1,(float) 1.2);
    up2.addUser((float) 1.1,(float) 0.7);
    up2.addUser((float) 1.1,(float) 0.2);
    up2.addUser((float) 1.4,(float) 1.0);
    up2.addUser((float) 1.4,(float) 0.5);
    up2.addUser((float) 1.7,(float) 0.9);
    up2.addUser((float) 1.7,(float) 0.4);
    up2.addUser((float) 1.4,(float) 0.1);

    UserPositions up3=new UserPositions();
```

```
        up3.addUser((float) 0.8,(float) 0.9);
        up3.addUser((float) 1.2,(float) 0.9);
        up3.addUser((float) 1.6,(float) 0.9);
        up3.addUser((float) 2.0,(float) 0.9);


        int nofUsers1=up1.getNofUsers();
        int nofUsers2=up2.getNofUsers();
        int nofUsers3=up3.getNofUsers();

        TrafficMatrix tm1=new TrafficMatrix(nofUsers1);
        //tm1.setTraffic(0,3,0,(float) 20);
        //tm1.setTraffic(1,2,0,(float) 20);

        TrafficMatrix tm2=new TrafficMatrix(nofUsers2);
        tm2.setTraffic(0,8,0,(float) 20);
        tm2.setTraffic(1,7,0,(float) 20);

        TrafficMatrix tm3=new TrafficMatrix(nofUsers3);
        tm3.setTraffic(0,3,0,(float) 5);

        Ours o = new
Ours(nofUsers2,simulationTime,tm2,up2,nofIterations,utilityType,subType,Pmax,backlogsAreFixed,simulatingExpo);
        float sizeOfPacket=((User) o.network.users.elementAt(0)).getSizeOfPacket();
        //o.network.printNetwork();

        System.out.println("m1="+((User)
o.network.users.elementAt(7)).getReceivedData(1,0)/(sizeOfPacket*simulationTime));
        System.out.println("m2="+((User)
o.network.users.elementAt(8)).getReceivedData(0,0)/(sizeOfPacket*simulationTime));
        //System.out.println("m3="+((User)
o.network.users.elementAt(3)).getReceivedData(1)/(sizeOfPacket*simulationTime));
    }
}
```

```
public class MulticlassActivationVector{

   int activatedForClass[][];
   int correspondingFlow[][];

   public MulticlassActivationVector(int nofUsers){
      activatedForClass=new int[nofUsers][nofUsers];
      correspondingFlow=new int[nofUsers][nofUsers];
      resetActivations();
   }

   public void resetActivations(){
      for (int i=0; i<activatedForClass.length; i++){
         for (int j=0; j<activatedForClass.length; j++){
            activatedForClass[i][j]=-1;
            correspondingFlow[i][j]=-1;
         }
      }
   }

   public void activateForClass(int from,int to,int classId,int flowId){
      activatedForClass[from][to]=classId;
      correspondingFlow[from][to]=flowId;
   }

   public int getActivatedClass(int from,int to){
      return activatedForClass[from][to];
   }

   public int getCorrespondingFlow(int from,int to){
      return correspondingFlow[from][to];
   }

   public void printMAV(){
      System.out.println("MAV:");
      for (int i=0; i<activatedForClass.length; i++){
         for (int j=0; j<activatedForClass.length; j++){
            if (getActivatedClass(i,j)!=-1){
               System.out.println(i+"->"+j+" for "+getActivatedClass(i,j)+" flowId="+getCorrespondingFlow(i,j));
            }
         }
      }
   }
}
```

```
import java.util.Vector;
import java.lang.Math;

public class Links{

    int nofTransmissionPairs;
    float links[][];
    Transmissions transmissions;
    Vector users;
    float noise=1;

    float neighborhoodCreterium=15;

    public Links(Transmissions trans,Vector users){
        nofTransmissionPairs=trans.getNofTransmissions();
        links=new float[nofTransmissionPairs][nofTransmissionPairs];
        transmissions=trans;
        this.users=users;
        calculateLinks();
    }

    public void setLink(int from,int to,float GFromTo){
        links[from][to]=GFromTo;
    }

    public float getGfromTo(int from,int to){
        return links[from][to];
    }

    public float getNoise(){
        return noise;
    }

    public void calculateLinks(){

        float G1=200;
        float G2=200;

        for (int iTrans=0; iTrans<nofTransmissionPairs; iTrans++){
            for (int jTrans=0; jTrans<nofTransmissionPairs; jTrans++){

                int from=transmissions.getFromNode(iTrans);
                int to=transmissions.getToNode(jTrans);

                User fromU=(User) users.elementAt(from);
                User toU=(User) users.elementAt(to);

                float distanceSquared=(float)
(Math.pow((fromU.getXPos()-toU.getXPos()),2)+Math.pow((fromU.getYPos()-toU.getYPos()),2));
                float d4=(float) Math.pow(distanceSquared,2);

                //***???????? ->
                float Aij=1;


                float Gij;
                if (from==to){
                    Gij=0;
                }
                else {
                    if (iTrans==jTrans){
                        Gij=Aij/d4;
                    }
                    else {
                        Gij=Aij/(d4*G1);
                    }
                }

                links[iTrans][jTrans]=Gij;
```

```
        }
      }
    }

    public void printLinks(){
      for (int i=0; i<nofTransmissionPairs; i++){
        for (int j=0; j<nofTransmissionPairs; j++){
          System.out.print("G("+i+","+j+")="+links[i][j]+"  ");
        }
        System.out.println();
      }
      System.out.println();
    }

    public float calculateSINRatReceiver(int transmission,PowerPriceStrategy pps){

      int from=transmissions.getFromNode(transmission);
      float Gii = links[transmission][transmission];

      float Pii=pps.getPower(transmission);
      float receivedPower=Gii*Pii;
    //  System.out.println("Pi="+Pii+" Gii"+Gii);


      float interference=0;


      for (int i=0; i<nofTransmissionPairs; i++){
        if (i!=transmission){
          int fromTemp=transmissions.getFromNode(i);
          float Gij = links[i][transmission];
          float Pij=pps.getPower(i);

//          System.out.println("Pij="+Pij+" Gij"+Gij+" i="+i+"j="+transmission);

          float interferenceJ=Gij*Pij;
          interference+=interferenceJ;

        }
      }

      return receivedPower/(interference+noise);
    }

    public float calculateIi(int transmission,PowerPriceStrategy pps){

      float interference=0;

      //System.out.println("interference="+interference);

      for (int i=0; i<nofTransmissionPairs; i++){
        if (i!=transmission){
          int fromTemp=transmissions.getFromNode(i);
          float Gij = links[i][transmission];

          float Pij=pps.getPower(i);

          float interferenceJ=Gij*Pij;
          interference+=interferenceJ;
        }
      }

      return interference;
    }

    public void updateLinks(Transmissions trans){
      transmissions=trans;
      nofTransmissionPairs=trans.getNofTransmissions();
      links=new float[nofTransmissionPairs][nofTransmissionPairs];
```

```
        calculateLinks();
    }
}
```