Scheduling High Speed Data in (Adversarial) Wireless Networks

Matthew Andrews
Bell Labs

June 27, 2005
Standard single server scheduling problem

• Choose queue/user to serve
  – Serve at rate $C$
  – Service rate constant in time & user independent

• Many scheduling algorithms
  – Weighted Fair Queueing, Earliest Deadline First, First In First Out
Basic wireless data scheduling problem

• Choose queue/user
  – Serve distant user at **38.4kbps** or close user at **614.4kbps**
  – Service rates **user dependent!!!!**
  – Service rates **change over time!!!!**
    (user mobility, channel fading)
Talk outline

• Example wireless technology
  – CDMA1x-EVDO

• Models
  – Infinitely backlogged queues vs external arrival process
  – Stochastic channels vs adversarial channels

• Scheduling results
  – How to achieve fairness?
  – How to maintain quality-of-service?
  – How to keep queues and delays small?
Example Technology: CDMA1x-EVDO (Data Only)

- **Data only**
  - One user at a time.
  - Chosen user gets all system resources (power etc.)

Results apply to most infrastructure based wireless data systems.
EVDO continued

Time divided into 1.667ms slots
(600 slots per second)

• Mobile measures downlink signal-to-noise ratio (SNR)

• Calculates rate at which it can receive data

• Informs basestation in a Data Rate Control msg (DRC)

• Scheduler chooses user based on DRC values

Potential values for DRC

0kbps
38.4kbps
76.8kbps
153.6kbps
307.2kbps
614.4kbps
921.6kbps
1.2Mbps
1.8Mbps
2.4Mbps

Some issues:-

• For low DRC values, must reserve multiple slots

• Actual transmission rate may be higher than DRC (due to lucky decoding)

• EVDO Rev A
  – multiple users per slot
  – multiple flows per user
Wireless models
Wireless data scheduling model   (models EV-DO)

- Choose a queue/user to serve at time $t$
  - If we choose user $i$, serve at rate $DRC_i(t)$

- **Opportunistic** scheduling: Serve user when DRC is high
Scheduling research: should we bother?

- **Wireline:** gigantic literature on end-to-end user-level scheduling
  - Only the simplest schemes get implemented
- **Wireless:** is it any different?
  - Probably…

<table>
<thead>
<tr>
<th>Wireline Router</th>
<th>Wireless Basestation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deals with <strong>thousands</strong> of flows</td>
<td>Deals with <strong>tens</strong> of flows</td>
</tr>
<tr>
<td>Am I the <strong>bottleneck</strong>?</td>
<td>I’m the <strong>bottleneck</strong>!!</td>
</tr>
<tr>
<td>Easy to add capacity</td>
<td>Hard to add capacity</td>
</tr>
<tr>
<td>Scheduling interval ~1μs</td>
<td>Scheduling interval ~1ms</td>
</tr>
</tbody>
</table>
Arrival model 1: Infinitely backlogged queues

- Each user always has data to serve
- $R_i = $ long-term throughput of user $i$
- Want to optimize $F(R_1, \ldots, R_n)$
- Proportional Fair
  - Optimize $\sum_i \log R_i$
Arrival model 2: External arrival process

arrival vector \((a_1(t),...,a_n(t))\)

\[\text{service rate vector } (DRC_1(t),...,DRC_n(t))\]

- **Arrival vector** \((a_1(t),...,a_n(t))\)

- **Want stability**
  - Want bounded queues whenever possible

- **Want small queues**

- **Want small delays**
Channel model 1: Stationary stochastic process

- Service rate vector determined by state of ergodic Markov chain $M$

- $M(t) \rightarrow (DRC_1(t), \ldots, DRC_n(t))$
Mobility

- Mobility destroys stationarity
Channel model 2: Adversarial process

- Service rate vector determined by adversary
- Adversary wants scheduler to do bad things
- Adversary enables worst-case analysis
Arrival model 1: Infinitely backlogged queues

Channel model 1: Stationary stochastic process
Proportional Fair Metric

• Common objective
  – $R_i = \text{long-run throughput of user } i$

  – **Maximize** $\Sigma_i \log R_i$

  – “Doubling throughput of user $i$ has same effect on metric as doubling throughput of user $j$”
Scheduler used in practice…

- **EV-DO scheduler - Proportional Fair** (Tse)
  - \( r_i(t) = \) service to user \( i \) at time \( t \), (either 0 or \( DRC_i(t) \))

\[
R_i(t + 1) = \left( 1 - \frac{1}{\tau} \right) R_i(t) + \frac{r_i(t)}{\tau} \quad \text{(typically \( \tau = 1024 \))}
\]

At time \( t \) serve \( \arg \max_i \frac{DRC_i(t)}{R_i(t)} \)

- **Prop fair maximizes** \( \sum_i \log R_i(t) \)
Intuitive analysis for Proportional Fair

\[ F = \sum_i \log R_i(t) \]

\[ \nabla F = \left( \frac{1}{R_1(t)}, \ldots, \frac{1}{R_n(t)} \right) \]

Change in \( F \) \approx \nabla F \cdot \left( \frac{1}{\tau} (r_1(t), \ldots, r_n(t)) - \frac{1}{\tau} (R_1(t), \ldots, R_n(t)) \right) = \frac{DRC_i(t)}{\tau R_i(t)} - \frac{n}{\tau} \text{ if user } i \text{ served} \]

Prop Fair maximizes change in \( F \)

System converges to optimum solution as \( \tau \to 0 \)

(Rigorous analysis by Agrawal-Subramanian, Kushner-Whiting, Stolyar)
Drawback with Proportional Fair

- Maximizing $\Sigma_i \log R_i$ gives no control to rates of individual users

- Sometimes want to provide minimum/maximum rate constraints to individual users

- Objective becomes

  \[
  \text{Maximize } \Sigma_i \log R_i \\
  \text{Subject to } R_i \geq R_i^{\text{min}} \\
  R_i \leq R_i^{\text{max}}
  \]

\[R_1\]

unconstrained rate region

\[R_2\]

rate region with minrate constraints
Prop Fair with rate constraints

- **PFMR (Prop Fair with min/max rate constraints)**

\[
\text{serve } \arg \max_i \frac{DRC_i(t)}{R_i(t)} e^{\alpha_i T_i(t)}
\]

- Variables and Parameters
  - \( R_i(t) \) – average service rate (exponentially filtered)
  - \( T_i(t) \) – "token counter"
    - decremented by pckt size whenever pckt served
    - incremented at rate \( R_i^{min} \) when \( T_i \geq 0 \)
    - incremented at rate \( R_i^{max} \) when \( T_i < 0 \)

A-Qian-Stolyar

Similar objectives considered by Liu-Chong-Shroff
Performance of PFMR

- **PFMR (Prop Fair with min/max rate constraints)**

\[
\text{serve} \quad \arg \max_i \frac{DRC_i(t)}{R_i(t)} e^{a_i T_i(t)}
\]

- If \( R_i^{\text{min}} \leq R_i(t) \leq R_i^{\text{max}} \)
  - Token counter stays near zero, sched metric reverts to Prop Fair

- If \( R_i(t) < R_i^{\text{min}} \)
  - Token counter drifts positive until user \( i \) satisfies min rate constraint

- If \( R_i(t) > R_i^{\text{max}} \)
  - Token counter drifts negative until user \( i \) satisfies max rate constraint

- If \( \tau \to 0, a_i \propto \tau \) and system converges, then it converges to optimal soln
Prop Fair

Rmin 9.6kbps for all users

(In practice set token counter to $1.2 \times R_{\text{min}}$)
28 users have Rmin 9.6kbps

2 users have Rmin 48kbps
Comparison with “obvious” solution

- **PFMR (Prop Fair with min/max rate constraints)**

  \[
  \text{serve} \quad \arg \max_i \frac{DRC_i(t)}{R_i(t)} e^{a_iT_i(t)}
  \]

  - Variables and Parameters
  
    - \(R_i(t)\) – average service rate (exponentially filtered)
    - \(T_i(t)\) – “token counter”
      - decremented by pckt size whenever pckt served
      - incremented at rate \(R_i^{\text{min}}\) when \(T_i \geq 0\)
      - incremented at rate \(R_i^{\text{max}}\) when \(T_i < 0\)

- **Penalty function method**

  \[
  \text{serve} \quad \arg \max_i \frac{r_i}{R_i - R_{\text{min}}}
  \]

  PFMR provides min rate with less throughput loss !!

  e.g. If Prop Fair, by itself, provides min rate then PFMR acts same as Prop Fair

  Penalty function method has lower throughput than Prop Fair
\( R_{\text{min}} = 20 \text{kbps} \)

- Penalty Function
- PFMR with token rate \( R_{\text{min}} \)
- PFMR with token rate \( 1.2 \times R_{\text{min}} \)
Arrival model 1: Infinitely backlogged queues

Channel model 2: Adversarial process
Proportional Fair Metric

- Definition different for adversarial channels
  - \( R_i(t) = \) total throughput of user \( i \) by time \( t \)

- Objective: Maximize \( \sum_i \log R_i(t) \) for all \( t \)

- Recall: for stochastic channels, optimality possible
Results

• For adversarial channels with $n$ users:
  – For any algorithm (online or offline), there exists rate vector sequence and time $T$ s.t.
    \[ \sum_i \log R_i(T) \leq \sum_i \log R_i^*(T) - \Omega(n) \]
  
  – For any online algorithm, there exists rate vector sequence and time $T$ s.t.
    \[ \sum_i \log R_i(T) \leq \sum_i \log R_i^*(T) - O(n \log n) \]

  – There exists online algorithm s.t. for all $t$ and all rate vector sequences,
    \[ \sum_i \log R_i(t) \geq \sum_i \log R_i^*(t) - O(n \log n) \]
### Lower Bound (1)

**For adversarial channels with \( n \) users:**

- For any algorithm (online or offline), there exists rate vector sequence and time \( T \) s.t.

\[
\Sigma_i \log R_i(T) \leq \Sigma_i \log R^*_i(T) - \Omega(n)
\]

<table>
<thead>
<tr>
<th>Time</th>
<th>( DRC_0(t) )</th>
<th>( DRC_1(t) )</th>
<th>( DRC_{n/2-1}(t) )</th>
<th>( DRC_{n/2}(t) )</th>
<th>( DRC_{n/2+1}(t) )</th>
<th>( DRC_{n-1}(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( n )</td>
<td>( n )</td>
<td>...</td>
<td>( n )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>( n )</td>
<td>( n )</td>
<td>...</td>
<td>( n )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>( n )</td>
<td>( n )</td>
<td>...</td>
<td>( n )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( n^2/2-2 )</td>
<td>( n )</td>
<td>( n )</td>
<td>...</td>
<td>( n )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( n^2/2-1 )</td>
<td>( n )</td>
<td>( n )</td>
<td>...</td>
<td>( n )</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Optimal schedule for time \( n^2/2-1 \): assign each user \( n/2 \) slots

| \( n^2/2 \) | 0 | 0 | ... | 0 | \( n \) | \( n \) | ... | \( n \) |
| \( n^2/2+1 \) | 0 | 0 | ... | 0 | \( n \) | \( n \) | ... | \( n \) |
| \( n^2/2+2 \) | 0 | 0 | ... | 0 | \( n \) | \( n \) | ... | \( n \) |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| \( n^2-2 \) | 0 | 0 | ... | 0 | \( n \) | \( n \) | ... | \( n \) |
| \( n^2-1 \) | 0 | 0 | ... | 0 | \( n \) | \( n \) | ... | \( n \) |

Optimal schedule for time \( n^2-1 \):
- For first \( n^2/2 \) slots, assign service equally among users \( 0,1,\ldots,n/2-1 \)
- For remaining \( n^2/2 \) slots, assign service equally among users \( n/2,n/2+1,\ldots,n-1 \)
Lower Bound (2)

• For adversarial channels with \( n \) users:
  – For any online algorithm, there exists rate vector sequence and time \( T \) s.t.

\[
\sum_i \log R_i(T) \leq \sum_i \log R_i^*(T) - \Omega(n \log n)
\]

<table>
<thead>
<tr>
<th>Time</th>
<th>( DRC_0(t) )</th>
<th>( DRC_1(t) )</th>
<th>( DRC_2(t) )</th>
<th>( DRC_3(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
  
User 2 has received least amount of service from online alg
| 2    | 1              | 1              | 0              | 1              |
| 3    | 1              | 1              | 0              | 1              |
| 4    | 1              | 1              | 0              | 1              |
| 5    | 1              | 1              | 0              | 1              |
  
User 0 has received least amount of service from online alg
| 6    | 0              | 1              | 0              | 1              |
| 7    | 0              | 1              | 0              | 1              |
| 8    | 0              | 1              | 0              | 1              |
| 9    | 0              | 1              | 0              | 1              |
| 10   | 0              | 1              | 0              | 1              |
| 11   | 0              | 1              | 0              | 1              |
| 12   | 0              | 1              | 0              | 1              |
| 13   | 0              | 1              | 0              | 1              |
  
User 3 has received least amount of service from online alg
| 14   | 0              | 1              | 0              | 0              |
| 15   | 0              | 1              | 0              | 0              |

OPT serves
  • user 2 in time slots 0-1
  • user 0 in time slots 2-5
  • user 3 in time slots 6-13
Upper Bound

• **There exists** online algorithm s.t. for all $t$ and all rate vector sequences,
  \[
  \sum_i \log R_i(t) \geq \sum_i \log R^*_i(t) - O(n \log n)
  \]

  – Randomized algorithm: at each time step serve user chosen uniformly at random
    • $\mathbb{E}[R_i(T)] \geq \sum_i DRC_i(t) / n \geq R^*_i(t) / n$
    • $\log \mathbb{E}[R_i(T)] \geq \log R^*_i(t) - \log n$
    • $\sum_i \log \mathbb{E}[R_i(T)] \geq \sum_i \log R^*_i(t) - n \log n$

  – Algorithm can be derandomized
Arrival model 2: External arrival process
System Stability

• Want online algorithm that keeps system stable

• Proportional Fair is unstable!!!

– Basic problem - Prop Fair does not consider queue sizes!!!!
Instability of Proportional Fair

<table>
<thead>
<tr>
<th>$DRC_A(t)$</th>
<th>$DRC_B(t)$</th>
<th>$a_A(t)$</th>
<th>$a_B(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
<td>94</td>
<td>49</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>94</td>
<td>49</td>
</tr>
<tr>
<td>100</td>
<td>1000</td>
<td>94</td>
<td>49</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>94</td>
<td>49</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>94</td>
<td>49</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>94</td>
<td>49</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>94</td>
<td>49</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>94</td>
<td>49</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>94</td>
<td>49</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>94</td>
<td>49</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>94</td>
<td>49</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>94</td>
<td>49</td>
</tr>
</tbody>
</table>

Stable algorithm
- Assign $\frac{1}{2}$ of $B$-biased slots to $B$
- Assign remaining slots to $A$
- svc rate of $A$ → 95, svc rate of $B$ → 50

Proportional fair
- Assigns all $B$-biased slots to $B$
- Assigns remaining slots to $A$
- svc rate of $A$ → 90, svc rate of $B$ → 100
Arrival model 2: External arrival process

Channel model 1: Stationary stochastic process
Stable algorithm

- **Max Weight**  (See Tassiulas-Ephremides, Kahale-Wright, Neely-Modiano-Rohrs, Stolyar)
  - \( q_i(t) = \) queue size of user \( i \) at time \( t \)
  
  - Serve \( \arg\max_i q_i(t) \cdot DRC_i(t) \)

- Potential function \( \sum_i q_i(t)^2 \) has negative drift

- Queues remain stable!!!
Arrival model 2: External arrival process

Channel model 2: Adversarial process
The adversary

- At time $t$ adversary generates
  - DRC vector $(DRC_1(t), \ldots, DRC_n(t))$
  - Arrival vector $(a_1(t), \ldots, a_n(t))$
  - Its “own” schedule $X(t)$

- Admissibility:
  - For each time window $[\tau, \tau+w)$,
    $$(1-\varepsilon) \sum_{t \in [\tau, \tau+w) : X(t)=i} DRC_i(t) \geq \sum_{t \in [\tau, \tau+w)} a_i(t)$$
  - i.e. there exists a stable schedule
Instability

• No online algorithm is stable

• \( f(x) = \text{smooth, convex, increasing, unbounded function, e.g. } x^k \)

• Can force potential function

\[ \sum_i f(q_i(t)) \]

to be unbounded
Instability

• Two users, 0 and 1
  – DRC vector satisfies
    \[
    \frac{DRC_0(t)}{DRC_1(t)} = \frac{f'(q_1(t))}{f'(q_0(t))}
    \]
  – If online algorithm serves user 0
    \[a_0(t) = 0\quad a_1(t) = (1-\varepsilon)DRC_1(t)\]
  – If online algorithm serves user 1
    \[a_0(t) = (1-\varepsilon)DRC_0(t)\quad a_1(t) = 0\]
  – Online algorithm always serves “wrong” user
    \[f\] always increases

• Remark
  – Rate set = set of all DRC values is infinite
Stable algorithm

• Is stability possible for finite rate set?
  – Yes

• Bad situation:
  – Online alg. keeps serving $i$ but adv. keeps serving $j$

• Intuition
  – Track adversary’s schedule in past
    • Look at past DRCs and past arrivals

  – Try and “copy” the adversary
Tracking Algorithms

• Linear Tracking Algorithm
  – Counter $C_i(r)$ for each user $i$ and DRC value $r$
  
  – Serve $j = \text{argmax}_i C_i(DRC_i(t))$
  
  – Decrease $C_j(DRC_j(t))$ by 1
  
  – Calculate adversary’s schedule $X(s)$ for some $s \leq t$
  
  – Increase $C_{X(s)}(DRC_{X(s)}(t))$ by 1
  
  – Stable…? Don’t know…

<table>
<thead>
<tr>
<th>DRC process</th>
<th>Adv serves 1</th>
<th>Trk serves 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(5,3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3,10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4,6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1,0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(7,4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>time $s$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>time $t$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_0(7)$</td>
<td>- -</td>
<td></td>
</tr>
<tr>
<td>$C_1(10)$</td>
<td>+ +</td>
<td></td>
</tr>
</tbody>
</table>

• Quadratic Tracking Algorithm
  – Counters for **pairs** of users. Stable with polynomial queues
Stability

• How do we compute adversary’s schedule?

  – Need schedule \( X(t) \) such that admissibility condition holds

\[
(1-\varepsilon) \sum_{t \in [\tau, \tau+w]: X(t)=i} DRC_i(t) \geq \sum_{t \in [\tau, \tau+w]} a_i(t)
\]

  – Problem: this is NP-hard to compute

  – Fix: can run tracking algorithm against a fractional schedule

\[
x(t) = (x_1(t), \ldots, x_n(t))
\]

such that

\[
(1-\varepsilon) \sum_{t \in [\tau, \tau+w]} x_i(t) DRC_i(t) \geq \sum_{t \in [\tau, \tau+w]} a_i(t)
\]

\[
0 \leq x_i(t) \leq 1
\]

\[
\sum_i x_i(t) = 1
\]
Max Weight

• Max Weight
  – $q_i(t) = \text{queue size of user } i \text{ at time } t$

  – Serve $\arg\max_i q_i(t) \times DRC_i(t)$

  – Stable with finite rate set...? Don’t know...

  – Max Weight produces “exponential queues”

    – Queue size > $2^N$ for $N < U_{\text{rateset}}$

    – $U_{\text{EVDO-rateset}} = 2048$
## Max Weight

\[ a_0(t) = 64 \quad a_1(t) = 1 \]
\[ a_2(t) = 1 \quad a_3(t) = 1 \]

<table>
<thead>
<tr>
<th>( DRC_0(t) )</th>
<th>( DRC_1(t) )</th>
<th>( DRC_2(t) )</th>
<th>( DRC_3(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>64</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>\vdots</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>128</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>64</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>\vdots</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>128</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>64</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Build up user 1 queue

Build up user 2 queue

Build up user 3 queue

![Graph showing Max Weight over time for different users](image)
Summary

• The model matters
  – Do we assume users with large backlogs or do we want to serve an external arrival process?
  – Do we assume a smooth stationary channel process or do we assume an unpredictable adversarial channel process

• Infinite backlogs, stationary channels
  – Prop Fair and PFMR allocate throughputs fairly subject to bounds on user performance

• Infinite backlogs, adversarial channels
  – Cannot beat simple random algorithm

• External arrival process, stationary channels
  – Prop Fair not stable, MaxWeight maintains stability

• External arrival process, adversarial channels
  – Stability not always possible. Tracking algorithm gives stability when possible