Backpressure Scheduling

Tassiulas & Ephremides '1993

MaxWeight Algorithm: a policy to schedule links over time in such a way that any arrival rate vector in the stability region of the network

can be scheduled without causing any queue to become unstable.

Consider a given network, represented by a graph $G = (V, E)$ and a conflict graph $H = (V_H, E_H)$ where $V_H = E$, E_H is set of edges between "nodes" $\in V_H$ s.t the corresponding links have a conflict.

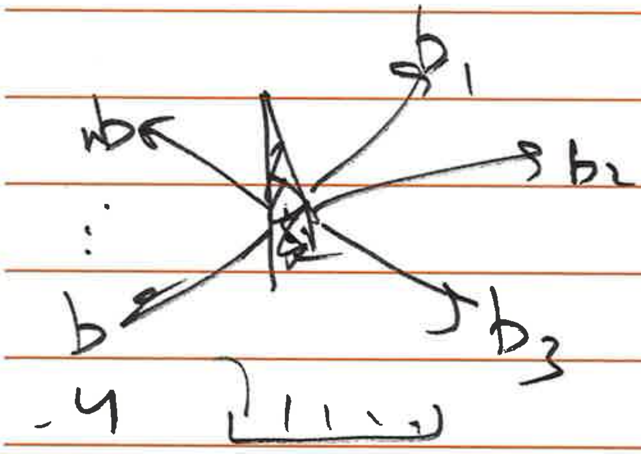
Scott Moeller (PhD student at USC)
developed BCP —
Backpressure Collection Protocol
Zarontiy protocol for data
collection in wireless sensor
networks, that is derived
from / inspired by backpressure
scheduling (MaxWeight).

—

BCP algorithm/protocol:

each node in a distributed
fashion, estimates ETX ij
& in every pkt, includes
in the header its own
(Qi) queue size & by logging neighbor'
pkts, figures out (Qj)
There is only 1 queue at each
node, ~~because~~ all nodes are
sending to the same sink

A special case where MaxWeight is easy: single transmitter.



e.g. downlink channel

for this setting: MaxWeight ⇒ serve the ~~~~ K-largest queues/users if K is the # of available channels.

if links are the same.

Else serve the K largest queues with respect to $Q_i \times R_i$

Q_i: queue

R_i: link rate

There are a set of (say unicast) flows in the network $(1 \ldots n)$ w/ corresponding sources & destinations:

$(s_1, t_1), (s_2, t_2) \ldots (s_n, t_n)$

The arrival rate vector is a set of rates for each flow:

$(r_1 \ldots r_n)$

$r_i \leftarrow$ how many pkts / sec. arrive (on average) at source s_i, to be sent to t_i.

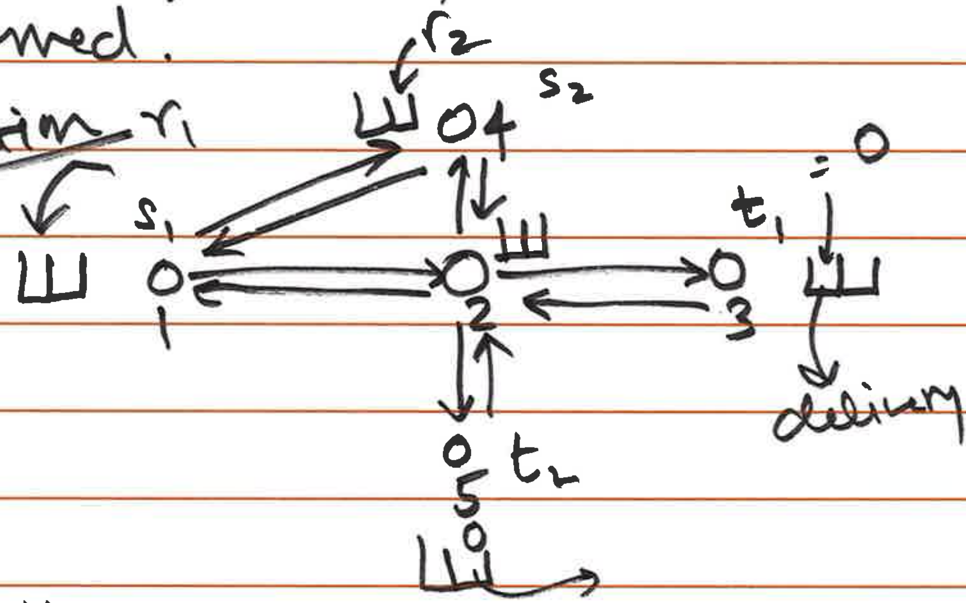
For simplicity assume exactly 1 pkt can be sent on each slot on any link.

A scheduling policy, at each time, can pick a set of non-interfering links, and move at most one pkt on each of those links.

At each node, we allow there to be n different queues: $Q_i^1, Q_i^2 \ldots Q_i^n$

The scheduling policy also gets to decide, on link i-j which flow's packet should be served.

Illustration

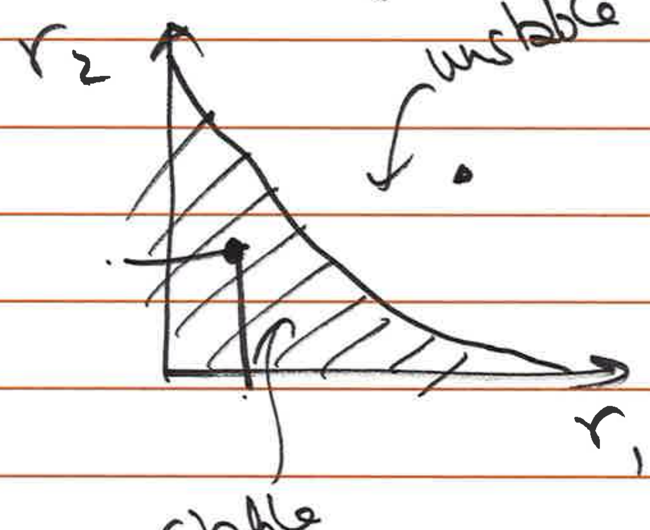


assume the conflict graph is a fully-connected / complete graph, i.e. every link interferes w/ every other link.

At the sink for each flow,
i.e at t_i, the queue
is assumed to be always 0.
i.e : $Q^i_{t_i} = 0$

Recall all queues are unlimited
in size (ideal assumption),
however we wish to avoid
queues growing without bound

(i.e. becoming unstable).



Network Stability region
$\underline{r} \in \Lambda$

MaxWeight can guarantee stability
$\forall \underline{r} \in \Lambda$. Sometimes called
a "Throughput-Optimal" policy.

The MaxWeight Scheduling Policy:

consider independent sets S — a set of edges that do not conflict w/ each other, i.e. can be co-scheduled.

pick the independent set S at each time, which maximizes the weight:

$$\max_{S} \sum_{e_{ij} \in S} w_{ij}^{*}$$

link rate of the $i-j$ edge

where

$$w_{ij}^{*} = \max_{f} \big(\underbrace{\max(Q_i^f - Q_j^f}_{\text{(queue backpressure or queue differential}}, 0) \big) \cdot R_{ij}$$

Repeat the above selection at each time.

Properties of Max-Weight:

1. guarantees stability $\forall \underline{r} \in \Lambda$

2. works for stochastic arrivals

3. works for any size network,
any conflict definition,
any set of flows.
(not necessarily unicast)

Requirements (Idealized):

- unlimited queue size
- Global time synchronization
- calculating the maximum weight independent set is NP-hard
- Centralized scheduling that needs global information about all queues at all times!

needed for analysis, not for implementation

Each node ~~i~~ i, computes for each ~~nbr~~ neighbor j, the weight

$$w_{ij} = \underbrace{Q_i - Q_j}_{\text{queue backpressure}} - V \cdot \underbrace{ETX_{ij}}_{\text{link measure}}$$

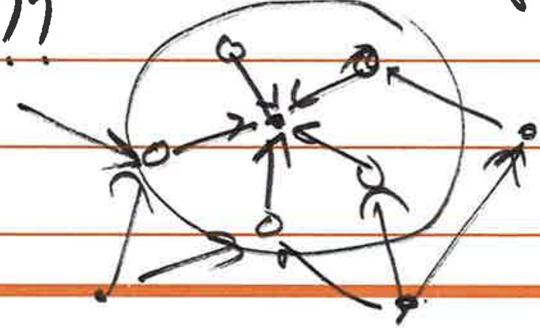
V: a tunable parameter

If $w_{ij^*} = \max_j w_{ij} > 0$

$j^* \xrightarrow{\text{argmax}} j$

then node i sends its next pkt to j^*, i.e. the nbr j that maximizes w_{ij}.

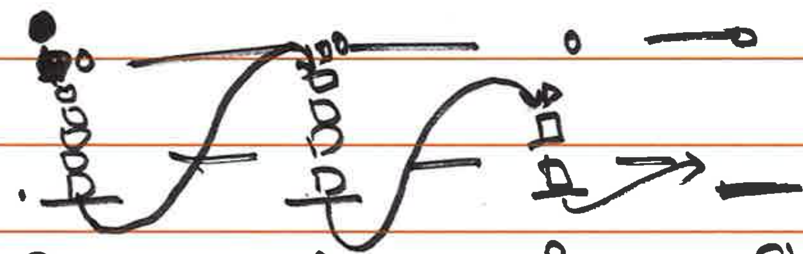
Note that the above description makes no reference to scheduling multiple links w/o interference — why??



(where)
BCP is a layer 3 protocol
— focuses just on which neighbor each node should forward a pkt to.

It lets L2 (MAC/link layer) use CSMA to take care of scheduling/avoiding interference. (when to send)

[Now, do not need global view or coordination or time sync.]
calculation at each node is trivial (max)



say $ETX_{ij} = 1 \ \forall$ links

$V = 2$

$w_{ij} = Q_i - Q_j - 2 \cdot 1$

$Q_i - Q_j - 2$

for sending a pkt on ij:

$w_{ij} > 0 \qquad Q_i - Q_j > 2$

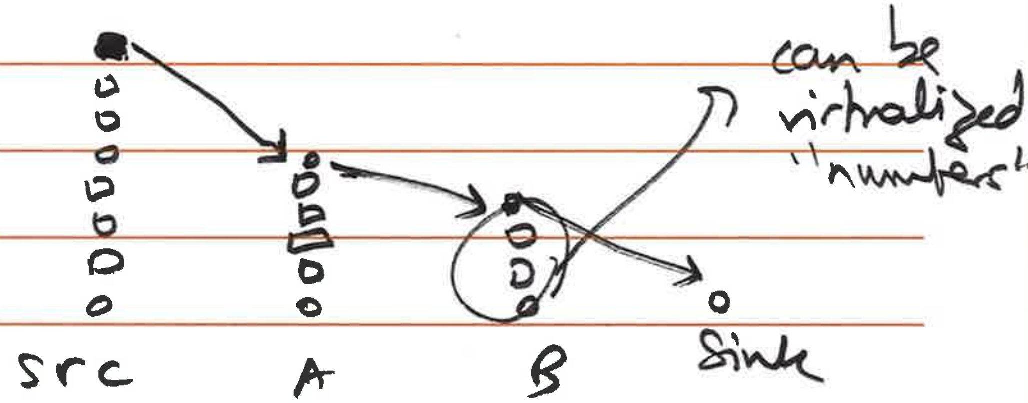
$Q_i > Q_j + 2$

FIFO Queuing is the standard queue service scheme in networks

w/ FIFO Queuing BCP has a high delay at low arrival rate!

Solution:

FIFO → LIFO.



completely solves the delay problem.

BUT ... a set of pkts stay stuck forever.

The early control pkts that are used to learn the gradients will be stuck.

Can show:

stuck pkts at node i

= V · ETX i to ~~dest~~ sink

shortest path ETX cost.

Problem w/ Generalizing BCP:

for any to any (unicast) traffic, we would need the number of queues at each node = # of flows in the network

worst case!

N nodes, # flows = $O(N^2)$

$$w_{ij} = Q_i - Q_j \quad \forall \in TX_{ij}$$

$$w_{ij}^f = Q_i^f - Q_j^f \quad - \quad \forall \in TX_{ij}$$

$$w_{ij}^* = \max_{j,f} w_{ij}^f$$

doesn't scale well for large networks

~~Open problem!~~ can this be solved?

- Shortest ETX routing
- Anypath routing
- Cooperative Broadcast
- Backpressure scheduling.

---

next class: routing in mobile networks
transport layer for wireless networks
is TCP good enough?

final exam is on August 8