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 $V_H$ such that the corresponding links have a conflict.
Scott Maeller (PhD student at USC) developed BCP - Backpressure Collection Protocol. It randomly selects a data collection in wireless sensor networks, that is derived from/inspired by backpressure scheduling (MaxWeight).

**BCP algorithm/principle:**

Each node in a distributed fashion, estimates ETX_{ij} and in every plot, includes in the header its own (Qi) queue size & by offig neighbors plots, figures out (Qi).

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., diamond channel.

For this setup:

MaxWeights:

gene the k-largest

Shene/curve if K is the # of available channels.

If two are the same:

Else share

The k largest mean with respect to Q_i x R_i

Shene link rate
There are a set of (say unicasts) flows in the network (1,...,n) with corresponding sources and 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:
$(r_1, \ldots, r_n)$

$r_i$ = how many bits/sec.
assuming (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 more at most one put on each of those links.

At each node, we allow time to be in 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.

![Diagram]

assume the conflict graph is a fully-connected Complete graph, i.e. every link interferes with every other link.
At the sink for each flow, i.e. at $t_i$, the queue is assumed to be always 0. i.e. $Q_{t_i} = 0$

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

(i.e. becoming unstable)

\[
\begin{align*}
\text{Unstable} & \quad \text{Stable} \\
\text{Network Stability Region} & \quad \text{MaxWeight can guarantee stability for } \forall X \in \Delta. \text{ Sometimes called a "Throughput-Optimal" policy.}
\end{align*}
\]
The MaxWeight Scheduling Policy:

Consider independent sets $S$—
a set of edges that do not conflict with each other, i.e., can be co-scheduled.

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

$$\max \sum_{e \in S} w^*_e$$

where $w^*_e = \max(\underleftarrow{Q}_i - \underleftarrow{Q}_j, 0)$

(queue backpressure
or queue differential)

Repeat the above selection at each time.
Properties of Max. Weight:

1. guarantees stability \( \forall r \in \Delta \)
2. works for stochastic arrivals
3. works for any size network, any conflict definition, any set of flows
   (not necessarily unique!)

Requirements (Idealized):

- unlimited queue size needed
- Global time synchronization for analysis
- Calculating the maximum weight independent set is NP-hard
- Centralized scheduling that needs global information at all queues at all times!
Each node $i$ computes for each neighbor $j$, the weight \( w_{ij} \) using:

\[
w_{ij} = o_i - o_j - V_{ETX_{ij}}
\]

where $o$ is a tunable parameter.

If \( w_{ij}^* = \max_{j \neq i} w_{ij} > 0 \), then node $i$ sends its next packet to $j^*$, i.e. the node $j$ that maximizes $w_{ij}$.

Note that the above description makes no reference to scheduling multiple links with interference — why??
BEP is a layer 3 protocol — frames just on which neighbor each node should forward a packet to.

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

[Now, do not need global view or coordination or time sync. Calculation at each node is trivial (max)]
say $ETX_{ij} = 1 + \text{links}$

$V = 2$

$wij = \phi_i - \phi_j - 2.1$

$\phi_i - \phi_j - 2$

for receipt a packet $ij$

$wij > 0 \quad \phi_i - \phi_j > 2$

$\phi_i > \phi_{j+2}$

FIFO queueing is the standard queue service scheme in networks.

FIFO queueing BCE has a high delay at low arrival rate!
FIFO $\rightarrow$ LIFO.

Can be virtualized number.

completely solves the delay problem.

But... a set of

plugs stay stuck forever.

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

Can show:

$\#$ stuck plugs at node $i$

$V \cdot ETX_{i to sink}$

Shortest path ETX cost.
Problem of Generalizing BCP: for any to any (unicast) traffic, we would need the number of queues at each node = # of flows in the network

\[ (N \text{ nodes, } \# \text{ flows} = O(N^2) \]

\[ w_{ij} = \Phi_i - \Phi_j - \sqrt{E_{X_{ij}}} \]
\[ w_{ij}^+ = \Phi_i^+ - \Phi_j^+ - \sqrt{E_{X_{ij}}} \]
\[ w_{ij}^* = \max_{j, f} w_{ij}^+ \]

doesn't scale well for large networks

Open problem: can this be solved?
- Short ETX routing
- Anycast 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