**Switch Algorithms**

**Switch**: A component that moves packets from one link to another link.

![Switch Diagram](image)

Input links | Output links

Packets arrive at input links and need to be transferred to corresponding output links.

**Popular switch architectures:**

1. **Output queued switch (OQ)**
2. **Input queued switch (IQ)**

**Output Queued Switch (OQ)**

All the packets arriving to input links are immediately transferred to corresponding output links. Each output link has a buffer/queue to store the packets that will be transmitted over the link subsequently.
let $\lambda_{ij}$ be the rate at which packets arrive at input link "i" destined for output link "j". Since each packet needs to be immediately moved to the corresponding output queue, the switch needs to operate at rate

$$\lambda = \sum_{j=1}^{N} \sum_{i=1}^{N} \lambda_{ij}.$$  

This means the switch memory that is used to process the incoming packets has to be at least as fast as $\lambda$. This memory requirement imposes a fundamental limit on the output queued switch architecture.

To analyze the performance, for simplicity assume the output link capacities are normalized to $1$ packet/unit-time, and each packet requires one unit of time for transmission. Then the required condition for stability at output queues is

$$\sum_{i=1}^{N} \lambda_{ij} < 1, \text{ for all } i < j \leq N.$$
Then the queue at each output link \( j \) is effectively a queue with arrival rate \( \sum_{i=1}^{N} \lambda_{ij} \) and deterministic service time at one for each packet. For example, if arrival processes are Poisson, then the arrival process to output queue "\( j \)" is also Poisson with rate \( \sum_{i=1}^{N} \lambda_{ij} \) (by merging property of Poisson processes), therefore output queue "\( j \)" behaves as a \( \text{M}/\text{D}/1 \) queue. Then average waiting/delay is followed from PK formula:

\[
\mathbb{E}[W_j] = \frac{\sum_{i=1}^{N} \lambda_{ij} \cdot 1}{2 \cdot \left( 1 - \sum_{i=1}^{N} \lambda_{ij} \right)}
\]

\[
\mathbb{E}[D_j] = \mathbb{E}[W_j] + 1
\]

By Little's Law:

\[
\mathbb{E}[Q_j] = \lambda_j \mathbb{E}[D_j] \quad \text{at each output queue} \quad 1 \leq j \leq N
\]
Input queued (IQ) switch:

Packets are stored at queues at input ports and are "switched" to corresponding output ports through a "crossbar switch fabric".

The figure shows a "3x3" switch fabric with $3^2 = 9$ crosspoints. To make a connection between Input 1 and Output 3, the red crosspoint has to be closed or on.

The physical constraint is that at most one crosspoint can be closed in each row and each column. Equivalently at any time, one input port can be connected to at most one output port and vice versa.
A simple way to represent a crossbar switch is by using a complete bipartite graph, as shown below:

```
Input 1 --- Output 1
Input 2 --- Output 2
Input 3 --- Output 3
```

"a 3x3 complete bipartite graph"

Hence, at any time, the set of connections from input ports to output ports should form a "matching" of the graph.

Matching: a set of edges that do not share any common vertex. In other words, each vertex (Input or output port) is associated with at most one edge.

We use the terms "matching" and "schedule" interchangeably.
Not that the memory required at the input queues to store and process packets is much lower than output queue switch (for example if line capacities are the same, the memory speed in IQ should be as fast as the link rate while in OQ it must be at least as fast as N times the link rate)

However there are two main issues with IQ switch

1. HOL (Head-of-line) blocking

2. Finding a "good" matching at each time.

HOL Blocking and Virtual Output Queues

1

2

3

black Pkts go to output 1
blue Pkts go to output 2
red Pkts go to output 3
Block Packet (HOL) blocks blue packet although the output port 2 (blue) is idle. HOL blocking can be resolved by using Virtual output Queues (VOQ)’s at input ports.

At each input port, one queue for each output port.

In N x N crossbar switch, need N^2 VOQ’s.

* Scheduling: finding a good matching at each time

For simplicity, we will assume:
- Time is slotted.
- Packets have unit size
- \( a_{i,j}(t) \): that packets arrive in time slot \( t \) at input port \( i \) destined for output port \( j \).

\[
\begin{align*}
\text{VQA} & 11 \quad \square \quad \square \quad \bullet \quad \square \quad \square \\
\text{VQA} & 12 \quad \square \quad \square \quad \bullet \quad \square \quad \square \\
\text{VQA} & 13 \quad \square \quad \square \\
\text{VQA} & 21 \quad \square \quad \square \quad \bullet \\
\text{VQA} & 22 \\
\text{VQA} & 31 \\
\text{VQA} & 32 \\
\text{VQA} & 33 \\
\end{align*}
\]

\[
a_{i,j}(t) = \begin{cases} 
1 \text{ with prob. } \lambda_{i,j} & 1 \leq i \leq N \\
0 \text{ with prob. } 1 - \lambda_{i,j} & 1 \leq j \leq N 
\end{cases}
\]
- \( M_{ij}(t) = 1 \) if a packet is transferred from input "i" to output "j"; otherwise \( M_{ij}(t) = 0 \).

- Thus \( a(t) = [a_{ij}(t)] \) is the arrival process and \( M(t) = [M_{ij}(t)] \) is the matching at time slot "t".

- Let \( M \) be the set of all matchings, then each \( M \in M \) satisfies
  \[
  \sum_{k=1}^{N} M_{ik} \leq 1 \quad \forall i \in \{1, \ldots, N\} \\
  \sum_{k=1}^{N} M_{kj} \leq 1 \quad \forall j \in \{1, \ldots, N\} \\
  M_{ij} \in \{0, 1\} \quad \forall i, j
  \]

- Let \( Q_{ij}(t) \) be the size of VOQ \( i, j \), then
  \[
  Q_{ij}(t+1) = (Q_{ij}(t) + a_{ij}(t) - M_{ij}(t))
  \]

Question: What is the capacity region of the IQ switch?

Recall that capacity region \( \Delta \) is the set of all arrival rates \( \lambda_{ij} \) that can be supported by the switch, i.e., queues remain stable.

Since at each time, at most one packet can be transferred from an input
port to an output port, a necessary condition for
stability is \( \sum_{k=1}^{N} \lambda_{i,k} < 1 \), \( 1 \leq i \leq N \).

Similarly, each output port can accept at most one packet
at each time slot, so \( \sum_{k=1}^{N} \lambda_{k,j} < 1 \), \( 1 \leq j \leq N \).

Let \( \Lambda = \left\{ \lambda : \lambda \geq 0, \sum_{i=1}^{N} \lambda_{i,k} < 1, \sum_{k=1}^{N} \lambda_{k,j} < 1 \right\} \), then

\( \Lambda_{\text{switch}} \subseteq \Lambda \).

Is \( \Lambda_{\text{switch}} \) smaller than \( \Lambda \) because of scheduling
(matching) constraints?

NO! In fact \( \Lambda_{\text{switch}} = \Lambda \) and scheduling constraints
do not affect it.

The reason can be explained using "Birkhoff–Von Neumann"
theorem. It states that given any \( \lambda \in \Lambda \), there exists a
\( M \in \text{Co}(\mathcal{M}) \) s.t. \( \lambda < M \) (component-wise),
where \( \text{Co}(\cdot) \) is the convex hull operator:

\[
\text{Co}(\mathcal{M}) = \left\{ X \in [0,1]^{N \times N} : X = \sum_{M \in \mathcal{M}} \alpha_{M} M ; \sum_{M \in \mathcal{M}} \alpha_{M} = 1 ; \alpha_{M} \geq 0 \right\}
\]
Now consider any scheduling algorithm \( \{ M(t), \ t=0, 1, 2, \ldots \} \) that chooses matching \( M(t) \) at time slot \( t \). Then effective service rate over \( [0, T] \) is \( r(t) \) which is

\[
    r(T) = \frac{1}{T} \sum_{t=0}^{T-1} M(t)
\]

so \( r(T) \in \text{co}(M) \) by definition.

Conversely, suppose \( \lambda \in \Lambda \), then \( \exists \mu \in \text{co}(M) \) such that \( \lambda < \mu \),

\[
    \mu = \sum_{M \in M} \alpha_M M, \quad \alpha_M > 0, \quad \sum_{M \in M} \alpha_M = 1
\]

now consider an algorithm that uses \( M \) for fraction \( \alpha_M \) of time, \( M \in M \); then each queue \( Q_{ij} \) is stable because \( \lambda_{ij} < \mu_{ij} \).

This shows that the set of supportable arrival rates is indeed \( \Sigma \). It also suggests an "offline" algorithm for supporting \( \lambda \in \Lambda \): if \( \lambda \) is known, we can find \( \alpha_M \)'s offline.

**Question:** Which schedule to choose if \( \lambda \) is not known?

**Answer:** At each time \( t \), choose a Max Weight Schedule.
\[ M^*(t) \in \arg \max_{M \in \mathcal{M}} \sum_{i,j} Q_{ij}(t) M_{ij} \]

In words, each edge \((i,j)\) has a weight equal to \(Q_{ij}\); and we choose the matching with the max weight.

**Why would Max Weight algorithm work?**

We prove that queues Markov chain is stable using Foster–Lyapunov theorem.

Consider the Lyapunov function:

\[ V(Q(t)) = \sum_{i,j} Q_{ij}(t)^2 \]

then calculate the drift:

\[ V(Q(t+1)) - V(Q(t)) = \sum_{i,j} \left( Q_{ij}(t+1)^2 - Q_{ij}(t)^2 \right) \]

bc. \((x)^2 \leq x^2\)

\[ \leq \sum_{i,j} \left( (Q_{ij}(t) + a_{ij}(t) - M^*_{ij}(t))^2 - Q_{ij}(t)^2 \right) \]

\[ = \sum_{i,j} 2 Q_{ij}(t) (a_{ij}(t) - M^*_{ij}(t)) + \sum_{i,j} (a_{ij}(t) - M^*_{ij}(t))^2 \]

\[ D = \mathbb{E}[V(Q(t+1)) - V(Q(t)) \mid Q(t) = Q] \]

\[ \leq \sum_{i,j} 2 Q_{ij}(t) (a_{ij}(t) - M^*_{ij}(t)) + \sum_{i,j} \mathbb{E}[a_{ij}(t)^2 + M^*_{ij}(t)^2] \]
\[ \leq 2 \sum_{ij} Q_{ij}(\lambda_{ij} - M^{*}_{ij(t)}) + \sum_{ij} (\lambda_{ij}) + N \]

because \{a_{ij(t)}\} was Bernoulli and independent of \(Q_{ij(t)}\).

Since \(\lambda \in \Lambda\), there exists a \(M \in \text{col}(M)\) s.t. \(\lambda \leq M\)
or equivalently \(\lambda(1 + \epsilon) \leq M\) for some \(\epsilon > 0\).

Hence,

\[ D \leq -2\epsilon \sum_{ij} \lambda_{ij} Q_{ij} + 2 \sum_{ij} Q_{ij}(\lambda_{ij} - M^{*}_{ij(t)}) + \text{const.} \]

\[ \leq -2\epsilon \sum_{ij} \lambda_{ij} Q_{ij} + 2 \sum_{ij} Q_{ij}(M_{ij} - M^{*}_{ij(t)}) + \text{const.} \]

this term is negative under Max Weight Scheduling. why ??!

\[ M \in \text{col}(M) \implies M = \sum_{M \in M} \alpha_{M} M, \quad \alpha_{M} > 0, \quad \sum_{M} \alpha_{M} = 1. \]

\[ M^{*}_{ij(t)} = \arg \max_{M \in M} \sum_{ij} Q_{ij} M_{ij} \]

\[ \implies \sum_{ij} Q_{ij} M^{*}_{ij(t)} \geq \sum_{ij} Q_{ij} M_{ij} \quad \forall M \in M \]

\[ \implies \sum_{ij} Q_{ij} M^{*}_{ij(t)} \geq \sum_{M \in M} \alpha_{M} \sum_{ij} Q_{ij} M_{ij} \]

\[ = \sum_{ij} Q_{ij} M_{ij} \]
\[ \sum_{ij} \lambda_{ij} (s_{ij} - \lambda_{ij}(t)) \leq 0 \]

In summary

\[ D \leq -2\varepsilon \sum_{ij} \lambda_{ij} s_{ij} + \text{const.} \]

which is negative for \( \{s_{ij}\} \) large enough. In fact, choose \( D > 0 \), then \( D \leq -\delta \) if

\[ -2\varepsilon \sum_{ij} \lambda_{ij} s_{ij} + \text{const.} \leq -\delta \]

\[ \Rightarrow \sum_{ij} \lambda_{ij} s_{ij} \geq \frac{\text{const.} + \delta}{2\varepsilon} \]

\[ \Rightarrow \{s_{ij}(t)\} \text{ is stable.} \]

* Algorithmic technique:

compute the drift and try to minimize the drift.
Complexity of Max Weight Scheduling

Max Weight Scheduling is an optimal algorithm, however, it requires to find a Max Weight matching of a bipartite graph at each time, i.e., it need to solve the following Integer program at each time:

\[
\begin{align*}
\text{max} & \quad \sum_{ij} z_{ij} x_{ij} \\
\text{s.t.} & \quad \sum_{k \in N} x_{ki} = 1 \quad \forall \ i \in N \\
& \quad \sum_{k \in N} x_{kj} = 1 \quad \forall \ j \in N \\
& \quad x_{ij} \in \{0, 1\}
\end{align*}
\]

The constraints above have a nice structure due to "Birkhoff–von Neumann" theorem. Specifically we can replace the above integer program with its "linear relaxation" without loss of optimality:

\[
\begin{align*}
\text{max} & \quad \sum_{ij} x_{ij} (2 + \delta_{ij}) \\
\text{s.t.} & \quad \sum_{k \in N} x_{ki} = 1 , \sum_{k \in N} x_{kj} = 1 \\
& \quad 0 \leq x_{ij} \leq 1
\end{align*}
\]
Finding an optimal solution of the linear program, (and hence a Max Weight Matching) needs \(O(N^3)\) runtime in a \(N \times N\) bipartite graph. Not practical for a high speed switch. One way to reduce the complexity is to use "maximal matching" at the expense of a factor \(1/2\) loss in the capacity region.

Maximal matching: is a matching such that if any other edge is added to it, it is not a matching anymore.

Maximal Matching Scheduling: At each time, first remove all edges with zero weights (\(\Omega_{ij}=0\)) from the complete bipartite graph, then find a maximal matching \(M_t\) in the remaining bipartite graph.
A maximal matching can be found in $O(n \log n)$ time. The following example shows that the achievable throughput under maximal matching algorithm may be smaller than the capacity region of the switch. (Example from Srikan- Yung book)

Example: Consider a 2x2 switch. Assume $\lambda_{12} = 0$ so $Q_{12} > 0$ for any $t > 0$. At most three edges need to be considered:

1. $Q_{11}$
2. $Q_{21}$
3. $Q_{22}$

Consider the following specific implementation:

(i) if $Q_{11}, Q_{22} > 0$: choose both edges $(1,1), (2,2)$

(ii) if $Q_{11} > 0, Q_{22} = 0$: choose $(1,1)$

(iii) if $Q_{11} = 0, Q_{22} > 0$: choose $(2,2)$

(iv) if $Q_{11} = Q_{22} = 0$ and $Q_{21} > 0$: choose $(2,1)$

This means that $Q_{11}$ and $Q_{22}$ are served independently of $Q_{21}$ and independently of each other, whenever non-empty.
Thus for stability of $Q_1$ and $Q_{22}$ trivially we need

\[ \lambda_{11} < 1 \quad \text{and} \quad \lambda_{22} < 1. \]

Then in steady state, \( P(\mathbf{Q}_{1} = \mathbf{0}) = 1 - \lambda_{11} \), \( P(\mathbf{Q}_{2} = \mathbf{0}) = 1 - \lambda_{22} \).

Hence $Q_{21}$ can be served at most \((1 - \lambda_{11})(1 - \lambda_{22})\) fraction of time.

So \( \lambda_{21} < (1 - \lambda_{11})(1 - \lambda_{22}) \) for stability of $Q_{21}$.

This is strictly less than what is achievable under max weight scheduling. Under max weight $\lambda_{21} + \lambda_{11} < 1$ and $\lambda_{21} + \lambda_{22} < 1$ thus $\lambda_{21} < \min\{(1 - \lambda_{11}), (1 - \lambda_{22})\}$ which is greater than $(1 - \lambda_{11})(1 - \lambda_{22})$, for the same values of $\lambda_{11}$ and $\lambda_{22}$.

**Question:** What is the capacity region at maximal matching scheduling?

**Answer:** It's at least 50% of capacity region of the switch.

**Proof:** We need to show that queues are stable for any \( \lambda \) s.t. \( 2\lambda \in \Lambda \). We will use the following two properties:

(i) \exists \delta > 0 s.t. \[ \sum_k 2(1+\delta) \lambda_{kj} < 1 \quad \forall j \]

(iii) \[ \sum_h 2(1+\delta) \lambda_{ih} < 1 \quad \forall i \]

which implies \[ \sum_h \lambda_{ih} + \sum_k \lambda_{kj} \leq \frac{1}{1+\delta} \quad \forall (i,j). \]
(iii) Since $\text{M}(\tau)$ is a maximal matching, if $\alpha_{ij}(t) \neq 0$, either port $i$ or port $j$ must be involved in $\text{M}(\tau)$, otherwise we can include edge $(i,j)$ in the matching which is a contradiction with $\text{M}(\tau)$ being maximal.

This implies $\sum_h M_{ih}(t) + \sum_k M_{kj}(t) \geq 1$ for all $(i,j)$ s.t. $\alpha_{ij}(t) \neq 0$.

The properties (i), (iii) show that the total arrival rate to the queues $\{Q_{ih} : 1 \leq h \leq N; Q_{kj} : 1 \leq k \leq N\}$ is less than $\frac{1}{\lambda + \epsilon}$ while the total service rate is at least $1$. This indicates that the collection of queues must be stable.

This can be rigorously proved using Foster-Lyapunov theorem.

First note that $\{Q_{ij}(t)\}$ is irreducible Markov chain. Next, consider the Lyapunov function:

$$V(Q_{ij}(t)) = \sum_{ij} \alpha_{ij}(t) \left( \sum_k \alpha_{kj}(t) + \sum_h \alpha_{ih}(t) \right)$$

Then calculate the difference

$$V(Q_{ij}(t+1)) - V(Q_{ij}(t)) = \sum_{ij} \alpha_{ij}(t+1) \left( \sum_k \alpha_{kj}(t+1) + \sum_h \alpha_{ih}(t+1) \right) - \sum_{ij} \alpha_{ij}(t) \left( \sum_k \alpha_{kj}(t) + \sum_h \alpha_{ih}(t) \right)$$
\[
\begin{align*}
\sum_{ij} Q_{ij}(t) \left( \sum_k a_{kj}(t) + \sum_h a_{ih}(t) - \sum_k M_{kj}(t) - \sum_h M_{ih}(t) \right) \\
+ \sum_{ij} \left( a_{ij}(t) - M_{ij}(t) \right) \left( \sum_k a_{kj}(t) + \sum_h a_{ih}(t) - \sum_k M_{kj}(t) - \sum_h M_{ih}(t) \right) \\
+ \sum_{ij} \left( a_{ij}(t) - M_{ij}(t) \right) \left( \sum_k a_{kj}(t) + \sum_h a_{ih}(t) - \sum_k M_{kj}(t) - \sum_h M_{ih}(t) \right)
\end{align*}
\]

**Term 1**

**Term 2**

**Term 3**

Term 1 and Term 2 are the same (by rearranging the summations).

Term 3 \(\leq 2N^3 = \text{constant} \).

Hence:

\[ V(Q(t+1)) - V(Q(t)) \leq 2(\text{Term 1}) + \text{const.} \]

The 0-th is given by

\[
D = \mathbb{E}\left[ V(Q(t+1)) - V(Q(t)) \mid Q(t) = Q \right]
\]

\[ \leq 2 \sum_{ij} Q_{ij} \left( \sum_k a_{kj} + \sum_h a_{ih} - \sum_k M_{kj}(t) - \sum_h M_{ih}(t) \right) + \text{const.} \]

\[ \leq 2 \sum_{ij} Q_{ij} \left( \frac{1}{1 + \varepsilon} - 1 \right) + \text{const.} \]

\[ = \frac{-2 \varepsilon}{1 + \varepsilon} \sum_{ij} Q_{ij} + \text{const.} \]
Given any $\delta > 0$, if $\sum Q_{ij} > \frac{\text{Const.} + \delta}{\epsilon'}$
then drift $D < -\delta$.

Hence $\{X(t)\}$ is positive recurrent.