Classes of Circuit-Switched Networks

Circuit-switched networks are classified based upon:

- the connection assignments they can realize and
- how they can change from satisfying one CA to satisfying another.

Types of Connection Assignments

Permutation CA: a set of requests in which each input and output appears exactly once.

The symbol $\Sigma_N$ will denote the set of all permutation connection assignments for $N$-input, $N$-output networks.

Note: $|\Sigma_N| = N!$

A network is called a permutation network if it can satisfy all permutation connection assignments.

$d$-limited generalized CA: a set of requests in which no input appears more than $d$ times and no output appears more than once.

A network is called a $d$-limited generalized connector if it can satisfy all $d$-limited generalized connection assignments.

Generalized CA: a set of requests in which no output appears more than once.

A network is called a generalized connector if it can satisfy all generalized connection assignments.
Ways in Which Networks Change Connection Assignments

Consider two CAs, $A$ and $B$.

Suppose a network is to satisfy $A$ and then $B$.

The following might occur:

- Paths are set up for $A$.
- Data for $A$ is transmitted.
- Paths for $A$ are torn down.
- Paths are set up for $B$.
- Data for $B$ is transmitted.
- Paths for $B$ are torn down.

In most cases this would be fine, but suppose:

$$A = C \cup \{(a, \alpha)\} \text{ and } B = C \cup \{(b, \beta)\} \text{ and } |C| = 99,999.$$  

In this case, 99,999 paths are being torn down and then being immediately rebuilt. Imagine the waste!

**Q:** Would it be possible to only tear down the paths that change?

**A:** It depends upon the type of network.

For banyans the answer is yes. But these aren’t permutation networks.

For inexpensive permutation networks the answer is no.
Network Types

A network is *non-blocking* if it can change from satisfying $A$ to satisfying $B$ without tearing down paths in $A \cap B$, where $A$ and $B$ are any two connection assignments the network can realize.

A network is *rearrangeably non-blocking* if when changing from satisfying $A$ to satisfying $B$ it may tear down and rebuild some paths in $A \cap B$, where $A$ and $B$ are any two connection assignments the network can realize. These networks are called *rearrangeable* for short.

A network is *strictly non-blocking* if it can change from satisfying $A$ to satisfying $B$ without tearing down paths in $A \cap B$ for any routing of $A$, where $A$ and $B$ are any two connection assignments the network can realize.

A network is *wide-sense non-blocking* if it can change from satisfying $A$ to satisfying $B$ without tearing down paths in $A \cap B$ if a proper routing procedure had been followed for $A$, where $A$ and $B$ are any two connection assignments the network can realize.
Generic Clos Network

One of several networks described by Clos in BSTJ 1953.

- First stage consists of $m \times m'$ cells.
- Middle stage starts with $\sigma_{k,m'}$ link pattern.
- Middle stage consists of $k \times k$ cells.
- Last stage starts with $\sigma_{m',k}$ link pattern.
- Last stage consists of $m' \times m$ cells.

Characteristics determined by $m'$; two to be considered:
- Non-blocking.
- Rearrangeable.
The non-blocking Clos network is a strictly non-blocking permutation network.

For non-blocking Clos networks $m' = 2m - 1$.

Example, $k = 4$, $m = 2$:

Why $2m - 1$?
Proof the Network is Strictly Non-Blocking

Plan: find route for request \((0, 0)\) under worst-case conditions.

In first stage \((0, 0)\) can be blocked by \(\leq m - 1\) requests.

In center stage \((0, 0)\) can be blocked by \(\leq m - 1\) requests.

Therefore, \(2(m - 1) + 1 = 2m - 1\) center-stage cells needed.
Cost of Strictly Non-Blocking Clos Network

Cost $C(m, k) = 4km^2 - 2km + 2mk^2 - k^2$ crosspoints.

Minimum cost for fixed $N$:

First, eliminate $k$ from equation.

$N = mk$, so, $k = N/m$.

$C(m, N) = 4Nm - 2N + \frac{2N^2}{m} - \left( \frac{N}{m} \right)^2$ crosspoints

Take the derivative with respect to $m$:

$$\frac{d}{dm} C(m, N) = 4N - \frac{2N^2}{m^2} + \frac{2N^2}{m^3}$$

Cost is minimal for values of $m$ that solve:

$$0 = \frac{2m^3}{N} - m + 1$$

$$m \approx \sqrt[3]{\frac{N}{2}}.$$ 

Cost of approx.-minimum-cost network $4\sqrt{2}N^{1.5} - 4N$ crosspoints.

Cost is better than a crossbar, but not nearly the $O(N \log N)$ of the banyan.
The *rearrangeable Clos network* is a permutation network. Usually just called a *Clos network*. A generic Clos network with $m' = m$.

Example, $k = 4$, $m = 2$:

```
<table>
<thead>
<tr>
<th>0</th>
<th>&lt;0,0&gt;</th>
<th>&lt;1,0&gt;</th>
<th>&lt;2,0&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>&lt;0,1&gt;</td>
<td>&lt;1,1&gt;</td>
<td>&lt;2,1&gt;</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>&lt;0,2&gt;</td>
<td>&lt;1,2&gt;</td>
<td>&lt;2,2&gt;</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>&lt;0,3&gt;</td>
<td>&lt;1,3&gt;</td>
<td>&lt;2,3&gt;</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Why $m$?

Answer not as simple as strictly non-blocking Clos.

Will be covered after routing.
Routing Rearrangeable Clos Networks

The Looping Algorithm

*Looping algorithm* used to route Clos networks in which $m = 2$.

It can also route Clos networks in which $m$ is a power of 2.

Developed by Opferman and Wu.¹

Definition

The *dual* of a $2 \times 2$ cell input is the other input to that cell.

The *dual* of a $2 \times 2$ cell output is the other output of that cell.

---

The Looping Algorithm, Informally

1: **Start loop**: If all inputs routed, then done. Otherwise, choose an unrouted request, set input-stage cell arbitrarily.

2: **Continue loop**: Set middle and output stage cells.

3: For dual of output just routed:

4: Set middle-stage cell (back towards inputs).

5: If input-stage cell already set, goto **Start loop**. Otherwise consider dual of input, goto **Continue loop**.

\[ P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 3 & 6 & 7 & 2 & 1 & 5 & 0 \end{pmatrix} \quad P^{-1} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 5 & 4 & 1 & 0 & 6 & 2 & 3 \end{pmatrix} \]

\[
\begin{align*}
\pi((0, 0)) &= \begin{pmatrix} 0 & 1 \end{pmatrix} & \pi((0, 1)) &= \begin{pmatrix} 0 & 1 \end{pmatrix} \\
\pi((0, 2)) &= \begin{pmatrix} 0 & 1 \end{pmatrix} & \pi((0, 3)) &= \begin{pmatrix} 0 & 1 \end{pmatrix} \\
\pi((1, 0)) &= \begin{pmatrix} 0 & 1 & 2 & 3 \end{pmatrix} & \pi((1, 1)) &= \begin{pmatrix} 0 & 1 & 2 & 3 \end{pmatrix} \\
\pi((2, 0)) &= \begin{pmatrix} 0 & 1 \end{pmatrix} & \pi((2, 1)) &= \begin{pmatrix} 0 & 1 \end{pmatrix} \\
\pi((2, 2)) &= \begin{pmatrix} 0 & 1 \end{pmatrix} & \pi((2, 3)) &= \begin{pmatrix} 0 & 1 \end{pmatrix}
\end{align*}
\]
The Looping Algorithm, Pseudocode

INPUT
INT N /* the number of inputs. */, P[N] /* the permutation */.

CONSTANTS
INT top=0, bottom=1, unset=N

INITIALIZE
INT unrouted=0, left=0, right, PI=Inverse(P)
INT LeftCell[i]=RightCell[i]=unset FOR i = 0 to N/2-1

BEGIN
DO{
  WHILE( LeftCell[unrouted] != unset ){unrouted++}
  IF unrouted >= N/2 THEN RETURN ELSE left=2*unrouted ENDIF

  DO{
    SWITCH
      CASE (left MOD 2 == top): LeftCell[left/2]=0 /* Identity */
      CASE (left MOD 2 == bottom): LeftCell[left/2]=1 /* Transpose */
    END SWITCH

    right=P[left]

    SWITCH
      CASE (right MOD 2 == top): RightCell[right/2]=0 /* Identity */
      CASE (right MOD 2 == bottom):RightCell[right/2]=1 /* Transpose */
    END SWITCH

    left=( PI[ right XOR 1 ] ) XOR 1
    IF LeftCell[left/2] != unset THEN QUITLOOP ENDIF
  }ENDDO
}ENDDO
Looping Algorithm Example

\[ P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 3 & 6 & 7 & 2 & 1 & 5 & 0 \end{pmatrix} \]

\[ P^{-1} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 5 & 4 & 1 & 0 & 6 & 2 & 3 \end{pmatrix} \]

\[ \pi(\langle 0, 0 \rangle) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \]

\[ \pi(\langle 0, 1 \rangle) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \]

\[ \pi(\langle 0, 2 \rangle) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \]

\[ \pi(\langle 0, 3 \rangle) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \]

\[ \pi(\langle 1, 0 \rangle) = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \end{pmatrix} \]

\[ \pi(\langle 1, 1 \rangle) = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \]

\[ \pi(\langle 1, 2 \rangle) = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \]

\[ \pi(\langle 1, 3 \rangle) = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \]

\[ \pi(\langle 2, 0 \rangle) = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \]

\[ \pi(\langle 2, 1 \rangle) = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \]

\[ \pi(\langle 2, 2 \rangle) = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \]

\[ \pi(\langle 2, 3 \rangle) = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \]
Looping Algorithm

Time Complexity

Initialization

Most of time spent computing permutation inverse: $O(N)$.

Number of iterations: $N/2$ (one for each input-stage cell).

Operations per iteration:

(Iteration includes inner DO loops.)

Several operations, each taking $O(1)$ time.

Time complexity: $O(N)$.

Irony

Time to traverse network, 3 crosspoints.

Time to find path through, $O(N)$.

There are parallel algorithms which can route Clos network $m = 2$ in $O(\log N)$ time.

There is no way that a permutation connection assignment could route itself, as in an omega network.
Clos Network Cost

\[ C(m, k) = 2km^2 + k^2m \quad \text{xp.} \]

Slightly Lower Cost Rearrangeable Clos Network

Replace any input- or output-stage cell with a link pattern.

This simplification due to Waksman\(^1\) and others.

Now how much do we pay?

\[ C(m, k) = (2k - 1)m^2 + k^2m \quad \text{xp.} \]

---

Proof of Rearrangeability of Clos Network

Due to Slepian (1952, unpublished) and Duguid (1959, just a technical report).

Called the Slepian-Duguid proof.

Proof outline:

I Show that a single center-stage cell can always be routed.

II Show that routing the remaining cells is equivalent to routing a smaller Clos network.

III Use induction on size.
Part I of Proof

Assertion: For any rearrangeable Clos network and any permutation connection assignment there is always a set of requests that can be routed through a middle-stage cell.

\[ P = \begin{pmatrix}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
7 & 3 & 6 & 5 & 2 & 1 & 0 & 11 & 8 & 4 & 9
\end{pmatrix} \]

Part I proof outline:

- Description of something called a set of distinct representatives (SoDR).
- Description of how a SoDR relates to routing a single middle-stage cell.
- Use of Hall’s Theorem\(^2\) to prove the existence of a SoDR, in general.
- Use of Hall’s Theorem to prove the existence of a SoDR, for Clos networks.

Theorem of Distinct Representatives (Hall’s Theorem)

Let $S$ be a set, $A_i \subseteq S$, and $a_i \in A_i$ for $0 \leq i < k$.

The elements $a_i$ are a set of distinct representatives (SoDR) of $A_i$ if $a_i \neq a_j$ when $i \neq j$.

The theorem: there exists a set of distinct representatives of $A_i$ if the union of any $\kappa \leq k$ subsets have at least $\kappa$ distinct elements.

Stated another way: there exists a set of distinct representatives of $A_i$ if

$$\forall K \subseteq \langle k \rangle, \quad \left| \bigcup_{i \in K} A_i \right| \geq |K|.$$
Stated Using Balls and Urns

Let $S$ be a set of balls, each of a different color.

$$S = \{r, w, b\}.$$ 

Let there be $k$ urns, denoted $A_i$, for $0 \leq i < k$.

Each urn has zero or more balls (the same kind as in $S$).

$$A_0 = \{r, w\}, \quad A_1 = \{r, b\}, \quad A_2 = \{w\}.$$ 

Remove one ball from each urn.

These are a SoDR if each ball is a different color.

$$a_0 = r, \quad a_1 = b, \text{ and } a_2 = w.$$ 

*It’s not always possible to find a SoDR.*

A SoDR exists iff there are $\kappa \leq k$ different color balls inside any combination of $\kappa$ urns.

In the example above:

For $\kappa = 1$: Urn 0, 2 colors; urn 1, 2 colors; urn 2, 1 color.

For $\kappa = 2$: Urn 0 & 1, 3 colors; urn 0 & 2, 2 colors; urn 1 & 2, 3 colors.

For $\kappa = 3$: Urn 0 & 1 & 2: 3 colors.

So there exists a SoDR. (But we already knew that.)
Hall’s Theorem and Clos’ Network

The set $S$ is a set of output-stage cell labels.

Consider request $(a, \alpha)$.

This request enters through cell $\langle 0, \lfloor a/m \rfloor \rangle$ and exits through cell $\langle 2, \lfloor \alpha/m \rfloor \rangle$.

Define $c((a, \alpha)) = \lfloor \alpha/m \rfloor$.

The subsets $A_i$ are the output-stage cells through which requests entering $\langle 0, i \rangle$ pass. That is,

$$A_i = \{ c((a, \alpha)) \mid (a, \alpha) \in P, \lfloor a/m \rfloor = i \},$$

where $P$ is a permutation connection assignment.

The SoDR are used to find the permutation to be realized by a middle-stage cell:

$$\pi(\langle 1, 0 \rangle) = \begin{pmatrix} 0 & 1 & \cdots & k-1 \\ a_0 & a_1 & \cdots & a_{k-1} \end{pmatrix}.$$ 

For permutation

$$P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 3 & 6 & 5 & 2 & 1 & 0 & 10 & 11 & 8 & 4 & 9 \end{pmatrix},$$

$A_0 = \{2, 1, 2\}$, $A_1 = \{1, 0, 0\}$, $A_2 = \{0, 3, 3\}$, and $A_3 = \{2, 1, 3\}$.

One possible SoDR: $a_0 = 2$, $a_1 = 1$, $a_2 = 0$, $a_3 = 3$. 
Proof That a SoDR Can Always be Found for a Clos Network

Consider the requests associated with input-stage cells in $K \subseteq \langle k \rangle$, $\kappa = |K|:

$$P' = \{ (a, \alpha) \mid (a, \alpha) \in P, \lfloor a/m \rfloor \in K \}.$$ 

Consider the output-stage cells that these requests pass through:

$$A = \{ c(A) \mid A \in P' \}$$

Obviously, $|P'| = m\kappa$.

Since each output-stage cell can appear at most $m$ times:

$$|A| \geq \frac{|P'|}{m} = \frac{m\kappa}{m} = \kappa$$

In other words, for any set of $\kappa \leq k$ input-stage cells there are requests to pass through at least $\kappa$ output-stage cells.

Therefore, by Hall’s Theorem, one request passing through each input-stage cell can be chosen that goes through a different output-stage cell.

These requests can be used to route a middle-stage cell.

This completes the proof of Part I.
Proof of Part II

Assertion: *Finishing the routing of a \((3, (m, k, m), (k, m, k), T, T)\) Clos network in which a single middle-stage cell is routed is equivalent to the problem of routing an entire \((3, (m - 1, k, m - 1), (k, m - 1, k), T, T)\) Clos network.*

This can easily be visualized:

Details will be omitted. (This would make a good homework or final-exam question.)
Part III: Denouement

Theorem: All of the \((3, (m, k, m), (k, m, k), T, T)\) Clos Networks are permutation networks.

Proof by induction on \(m\):

Basis: A Clos network with one center-stage cell (i.e., \(m = 1\)) can always be routed.

Proof: By definition of the crossbar, or using Hall’s Theorem as in Part I.

Inductive Hypothesis: All Clos Networks of size 
\((3, (m', k, m'), (k, m', k), T, T)\)

for, \(0 < m' < m\), can be routed.

Assertion: If the IH is true then a \((3, (m, k, m), (k, m, k), T, T)\) Clos Network can be routed.

Proof:

By Part I a single center-stage cell can be routed.

By Part II and the IH the remainder of the network can be routed by routing an appropriately constructed 
\((3, (m - 1, k, m - 1), (k, m - 1, k), T, T)\) network.

Thus, a \((3, (m, k, m), (k, m, k), T, T)\) Clos Network can be routed.