WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

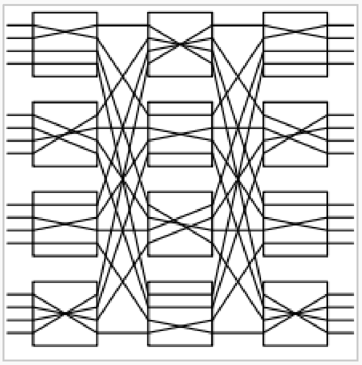
Print/export
Create a book
Download as PDF
Printable version

Languages
Српски / srpski
ไทย
Edit links

Nonblocking minimal spanning switch

From Wikipedia, the free encyclopedia
(Redirected from Nonblocking Minimal Spanning Switch)

A **nonblocking minimal spanning switch** is a device that can connect N inputs to N outputs in any combination. The most familiar use of switches of this type is in a telephone exchange. The term "non-blocking" means that if it is not defective, it can always make the connection. The term "minimal" means that it has the fewest possible components, and therefore the minimal expense.


A substitute for a 16x16 crossbar switch made from 12 4x4 crossbar switches.

Historically, in telephone switches, connections between callers were arranged with large, expensive banks of electromechanical relays, Strowger switches. The basic mathematical property of Strowger switches is that for each input to the switch, there is exactly one output. Much of the mathematical switching circuit theory attempts to use this property to reduce the total number of switches needed to connect a combination of inputs to a combination of outputs.

In the 1940s and 1950s, engineers in Bell Laboratories began an extended series of mathematical investigations into methods for reducing the size and expense of the "switched fabric" needed to implement a telephone exchange. One early, successful mathematical analysis was performed by Charles Clos (French pronunciation: [ʃaʁl Cləʊ]), and a switched fabric constructed of smaller switches is called a Clos network.[1]

Contents [hide]

Background: switching topologies [edit]

The crossbar switch [edit]

The crossbar switch has the property of being able to connect N inputs to N outputs in any one-to-one combination, so it can connect any caller to any non-busy receiver, a property given the technical term "nonblocking". Being nonblocking it could always complete a call (to a non-busy receiver), which would maximize service availability.

However, the crossbar switch does so at the expense of using N^2 (N squared) simple SPST switches. For large N (and the practical requirements of a phone switch are considered large) this growth was too expensive. Further, large crossbar switches had physical problems. Not only did the switch require too much space, but the metal bars containing the switch contacts would become so long that they would sag and become unreliable. Engineers also noticed that at any time, each bar of a crossbar switch was only making a single connection. The other contacts on the two bars were unused. This seemed to imply that most of the switching fabric of a crossbar switch was wasted.

The obvious way to emulate a crossbar switch was to find some way to build it from smaller crossbar switches. If a crossbar switch could be emulated by some arrangement of smaller crossbar switches, then these smaller crossbar switches could also, in turn be emulated by even smaller crossbar switches. The switching fabric could become very efficient, and possibly even be created from standardized parts. This is called a Clos network.

Completely connected 3-layer switches [edit]

The next approach was to break apart the crossbar switch into three layers of smaller crossbar switches. There would be an "input layer", a "middle layer" and an "output layer." The smaller switches are less massive, more reliable, and generally easier to build, and therefore less expensive.

A telephone system only has to make a one-to-one connection. Intuitively this seems to mean that the number of inputs and the number of outputs can always be equal in each subswitch, but intuition does not prove this can be done nor does it tell us how to do so. Suppose we want to synthesize a 16 by 16 crossbar switch. The design could have 4 subswitches on the input side, each with 4 inputs, for 16 total inputs. Further, on the output side, we could also have 4 output subswitches, each with 4 outputs, for a total of 16 outputs. It is desirable that the design use as few wires as possible, because wires cost real money. The least possible number of wires that can connect two subswitches is a single wire. So, each input subswitch will have a single wire to each middle subswitch. Also, each middle subswitch will have a single wire to each output subswitch.

The question is how many middle subswitches are needed, and therefore how many total wires should connect the input layer to the middle layer. Since telephone switches are symmetric (callers and callees are interchangeable), the same logic will apply to the output layer, and the middle subswitches will be "square", having the same number of inputs as outputs.

The number of middle subswitches depends on the algorithm used to allocate connection to them. The basic algorithm for managing a three-layer switch is to search the middle subswitches for a middle subswitch that has unused wires to the needed input and output switches. Once a connectible middle subswitch is found, connecting to the correct inputs and outputs in the input and output switches is trivial.

Theoretically, in the example, only four central switches are needed, each with exactly one connection to each input switch and one connection to each output switch. This is called a "minimal spanning switch," and managing it was the holy grail of the Bell Labs' investigations.

However, a bit of work with a pencil and paper will show that it is easy to get such a minimal switch into conditions in which no single middle switch has a connection to both the needed input switch and the needed output switch. It only takes four calls to partially block the switch. If an input switch is half-full, it has connections via two middle switches. If an output switch is also half full with connections from the other two middle switches, then there is no remaining middle switch which can provide a path between that input and output.

For this reason, a "simply connected nonblocking switch" 16x16 switch with four input subswitches and four output switches was thought to require 7 middle switches; in the worst case an almost-full input subswitch would use three middle switches, an almost-full output subswitch would use three different ones, and the seventh would be guaranteed to be free to make the last connection. For this reason, sometimes this switch arrangement is called a "$2n-1$ switch", where n is the number of input ports of the input subswitches.

The example is intentionally small, and in such a small example, the reorganization does not save many switches. A 16×16 crossbar has 256 contacts, while a 16×16 minimal spanning switch has 4×4×4×3 = 192 contacts.

As the numbers get larger, the savings increase. For example, a 10,000 line exchange would need 100 million contacts to implement a full crossbar. But three layers of 100 100×100 subswitches would use only 300 10,000 contact subswitches, or 3 million contacts.

Those subswitches could in turn each be made of 3×10 10×10 crossbars, a total of 3000 contacts, making 900,000 for the whole exchange; that is a far smaller number than 100 million.

Managing a minimal spanning switch [edit]

The crucial discovery was a way to reorganize connections in the middle switches to "trade wires" so that a new connection could be completed.

The first step in the nonblocking minimal spanning switch algorithm is just the naive earlier scheme (above): Search for a middle-layer subswitch that contains the needed in and out links. If a middle-layer subswitch is found that with unused links to both the needed input and output subswitches, then it is allocated, and the connection goes through.

However, since the number of middle subswitches is smaller in a minimal spanning switch than in a "2n−1" switch, sometimes this search fails. If a subswitch with the needed pair of unused links can't be found, a pair of middle subswitches must be found. One subswitch must have a link to the needed input subswitch; the other must have a link to the needed output subswitch. These subswitches are guaranteed to exist, because each input subswitch has as many outputs as inputs, and as there was one unused input (for the new connection to arrive on), there must be a corresponding unused output to some middle-layer switch. The same reasoning applies symmetrically to the output subswitch.

The problem is that these two available links to the middle layer are not to the *same* middle-layer switch, so they cannot make a working input-to-output connection. The algorithm finds a new arrangement of the connections through the two middle subswitches (call them A and B) that includes all of the existing connections, plus the desired new connection.

Make a list of all of the existing connections that pass through A or B. Include the new connection, although it is initially broken. The algorithm proper only cares about the internal connections from input to output switch, although a practical implementation also has to keep track of the correct input and output switch connections.

In particular, which middle subswitch a connection currently uses is not important; the algorithm will find a new assignment of connections to middle subswitches that accommodates all of the connections, including the new one.

In this list, each input subswitch can appear in at most two connections: one to subswitch A, and one to subswitch B. The options are zero, one, or two. Likewise, each output subswitch appears an at most two connections.

Next, assign a direction to each connection. Pick any connection which does not already have a direction assigned to it, and draw a "forward" arrow on it, from input to output. If its output subswitch has a second connection, draw a "backward" arrow on that, from output back to input, and follow that connection to its input subswitch. If that has a second connection, draw a forward arrow on it, and continue back and forth in this manner until one of two things must happen:

1. The chain terminates in a subswitch with only one connection, or
2. The chain loops back to the originally chosen connection.

In the first case, go back to the originally chosen connection and follow its chain backward, drawing additional direction arrows, until each link in the chain is assigned a direction, always head-to-tail.

Repeat until there are no more unassigned connections. When this is done, note that each input and output subswitch has at most one incoming arrow and at most one outgoing arrow. In other words, it participates in at most one forward and one backward connection.

Now, assign all of the forward connections to middle subswitch A, and all of the backward connections to middle subswitch B. Because each input and output subswitch has one link to each of A and B, and needs at most one link to each to accommodate the maximum of one forward and one backward connection it participates in, this assignment never produces conflicts.

After the connections are allocated in arrays in the software, then the electronics of the switch can actually be reprogrammed, physically moving the connections. The electronic switches are designed intentionally so that the new data can all be written into the electronics, and then take effect with a single logic pulse. The result is that the connection moves instantaneously, with an imperceptible interruption to the conversation. In older electromechanical switches, one occasionally heard a clank of "switching noise."

This algorithm is a form of topological sort, and is the guts of the algorithm that controls a minimal spanning switch.

Practical implementations of switches [edit]

As soon as the algorithm was discovered, Bell system engineers and managers began discussing it. After several years, Bell engineers began designing electromechanical switches that could be controlled by it. At the time, computers used tubes and were not reliable enough to control a phone system (phone system switches are safety-critical, and they are designed to have an unplanned failure about once per thirty years). Relay-based computers were too slow to implement the algorithm. However, the entire system could be designed so that when computers were reliable enough, they could be retrofitted to existing switching systems.

It's not difficult to make composite switches fault-tolerant. When a subswitch fails, the callers simply redial. So, on each new connection, the software tries the next free connection in each subswitch rather than reusing the most recently released one. The new connection is more likely to work because it uses different circuitry.

Therefore, in a busy switch, when a particular PCB lacks any connections, it is an excellent candidate for testing.

To test or remove a particular printed circuit card from service, there is a well-known algorithm. As fewer connections pass through the card's subswitch, the software routes more test signals through the subswitch to a measurement device, and then reads the measurement. This does not interrupt old calls, which remain working.

If a test fails, the software isolates the exact circuit board by reading the failure from several external switches. It then marks the free circuits in the failing circuitry as busy. As calls using the faulty circuitry are ended, those circuits are also marked busy. Some time later, when no calls pass through the faulty circuitry, the computer lights a light on the circuit board that needs replacement, and a technician can replace the circuit board. Shortly after replacement, the next test succeeds, the connections to the repaired subswitch are marked "not busy," and the switch returns to full operation.

The diagnostics on Bell's early electronic switches would actually light a green light on each good printed circuit board, and light a red light on each failed printed circuit board. The printed circuits were designed so that they could be removed and replaced without turning off the whole switch.

The eventual result was the Bell 1ESS switch (electronic switching system 1). This was controlled by a 3B20 computer, a lock-step dual computer using reliable diode transistor logic. In the 1ESS's 3B20, two computers performed each step, checking each other. When they disagreed, they would diagnose themselves, and the correctly running computer would take up switch operation while the other would disqualify itself and request repair. The 1ESS switch was still in limited use as of 2012, and had a verified reliability of less than one unscheduled hour of failure in each thirty years of operation, validating its design.

Initially it was installed on long distance trunks in major cities, the most heavily used parts of each telephone exchange. On the first Mother's Day that major cities operated with it, the Bell system set a record for total network capacity, both in calls completed, and total calls per second per switch. This resulted in a record for total revenue per trunk.

Modern switches [edit]

A practical implementation of a switch can be created from an **odd** number of layers of smaller subswitches. Conceptually, the crossbar switches of the three-stage switch can each be further decomposed into smaller crossbar switches. Although each subswitch has limited multiplexing capability, working together they synthesize the effect of a larger $N×N$ crossbar switch.

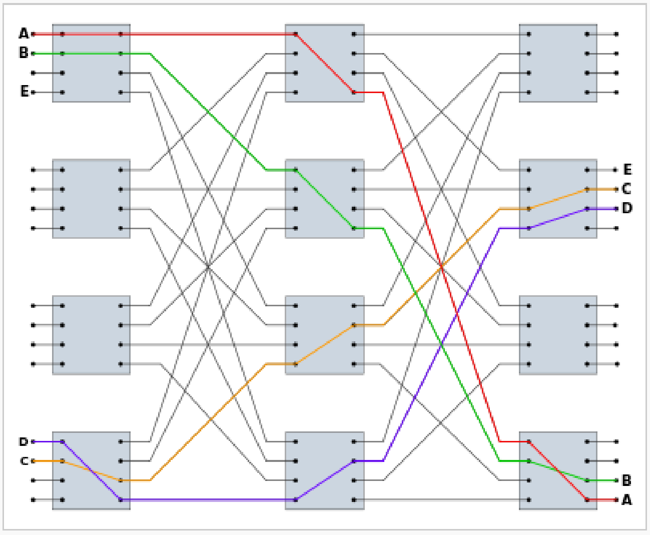
In a modern telephone switch, application of two different multiplexer approaches in alternate layers further reduces the cost of the switching fabric:

1. space-division multiplexers are something like the crossbar switches already described, or some arrangement of crossover switches or banyan switches. Any single output can select from any input. In digital switches, this is usually an arrangement of and gates. 8000 times per second, the connection is reprogrammed to connect particular wires for the duration of a time slot. **Design advantage:** In space-division systems the number of space-division connections is divided by the number of time slots in the time-division multiplexing system. This dramatically reduces the size and expense of the switching fabric. It also increases the reliability, because there are far fewer physical connections to fail.
2. time division multiplexers each have a memory which is read in a fixed order and written in a programmable order (or *vice versa*). This type of switch permutes time-slots in a time-division multiplexed signal that goes to the space-division multiplexers in its adjacent layers. **Design advantage:** Time-division switches have only one input and output wire. Since they have far fewer electrical connections to fail, they are far more reliable than space-division switches, and are therefore the preferred switches for the outer (input and output) layers of modern telephone switches.

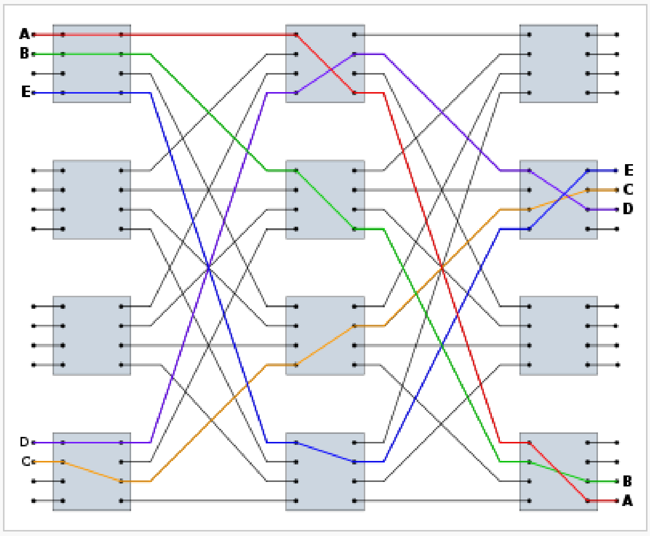
The scarce resources in a telephone switch are the connections between layers of subswitches. These connections can be either time slots or wires, depending on the type of multiplexing. The control logic has to allocate these connections, and the basic method is the algorithm already discussed. The subswitches are logically arranged so that they synthesize larger subswitches. Each subswitch, and synthesized subswitch is controlled (recursively) by the above algorithm.

If the recursion is taken to the limit, breaking down the crossbar to the minimum possible number of switching elements, the resulting device is sometimes called a crossover switch or a banyan switch depending on its topology.

Example of rerouting a switch [edit]



Signals A, B, C, D are routed but signal E is blocked, unless a signal, such as D shown in purple is rerouted



After D, in purple, is rerouted, Signal E can be routed and all the additional signals plus E are connected

See also [edit]

- Time Slot Interchange
- Clos Network
- Crossbar switch
- Banyan switch
- Fat tree
- Omega network

References [edit]

1. ^ Clos, Charles (Mar 1953). "A study of non-blocking switching networks" (PDF). *Bell System Technical Journal* **32** (2): 406–424. doi:10.1002/j.1538-7305.1953.tb01433.x. ISSN 0005-8580. Retrieved 22 March 2011.

Categories: Telephone exchanges | Graph algorithms