Fair queuing

Fair queuing is a family of scheduling algorithms used in some process and network schedulers. The algorithm is designed to achieve fairness when a limited resource is shared, for example to prevent flows with large packets or processes that generate small jobs from consuming more throughput or CPU time than other flows or processes.

Fair queuing is implemented in some advanced network switches and routers.

History

The term *fair queuing* was coined by John Nagle in 1985 while proposing round-robin scheduling in the gateway between a local area network and the internet to reduce network disruption from badly-behaving hosts.[1][2][3]

A byte-weighted version was proposed by Alan Demers, Srinivasan Keshav and Scott Shenker in 1989, and was based on the earlier Nagle fair queuing algorithm.[4][5] The byte-weighted fair queuing algorithm aims to mimic a bit-per-bit multiplexing by computing theoretical departure date for each packet.

The concept has been further developed into weighted fair queuing, and the more general concept of traffic shaping, where queuing priorities are dynamically controlled to achieve desired flow quality of service goals or accelerate some flows.

Principle

Fair queuing uses one queue per packet flow and services them in rotation, such that each flow can "obtain an equal fraction of the resources".[1][2]

The advantage over conventional first in first out (FIFO) or priority queuing is that a high-data-rate flow, consisting of large packets or many data packets, cannot take more than its fair share of the link capacity.

Fair queuing is used in routers, switches, and statistical multiplexers that forward packets from a buffer. The buffer works as a queuing system, where the data packets are stored temporarily until they are transmitted.

With a link data-rate of *R*, at any given time the *N* active data flows (the ones with non-empty queues) are serviced each with an average data rate of *R/N*. In a short time interval the data rate may fluctuate around this value since the packets are delivered sequentially in turn.

Fairness

In the context of network scheduling, *fairness* has multiple definitions. Nagel's article uses round-robin scheduling of packets,[2] which is fair in terms of the number of packets, but not on the bandwidth use when packets have varying size. Several formal notions of fairness measure have been defined including max-min fairness, *worst-case fairness*,[6] and *fairness index*.[7]

Generalisation to weighted sharing

The initial idea gives to each flow the same rate. A natural extension consists in letting the user specify the portion of bandwidth allocated to each flow leading to weighted fair queuing and generalized processor sharing.

A byte-weighted fair queuing algorithm

This algorithm attempts to emulate the fairness of bitwise round-robin sharing of link resources among competing flows. Packet-based flows, however, must be transmitted packetwise and in sequence. The byte-weighted fair queuing algorithm selects transmission order for the packets by modeling the finish time for each packet as if they could be transmitted bitwise round robin. The packet with the earliest finish time according to this modeling is the next selected for transmission.

The complexity of the algorithm is *O(log(n))*, where *n* is the number of queues/flows.

Algorithm details

Modeling of actual finish time, while feasible, is computationally intensive. The model needs to be substantially recomputed every time a packet is selected for transmission and every time a new packet arrives into any queue.

To reduce computational load, the concept of *virtual time* is introduced. Finish time for each packet is computed on this alternate monotonically increasing virtual timescale. While virtual time does not accurately model the time packets complete their transmissions, it does accurately model the order in which the transmissions must occur to meet the objectives of the full-featured model. Using virtual time, it is unnecessary to recompute the finish time for previously queued packets. Although the finish time, in absolute terms, for existing packets is potentially affected by new arrivals, finish time on the virtual time line is unchanged - the virtual time line warps with respect to real time to accommodate any new transmission.

The virtual finish time for a newly queued packet is given by the sum of the virtual start time plus the packet's size. The virtual start time is the maximum between the previous virtual finish time of the same queue and the current instant.

With a virtual finishing time of all candidate packets (i.e., the packets at the head of all non-empty flow queues) computed, fair queuing compares the virtual finishing time and selects the minimum one. The packet with the minimum virtual finishing time is transmitted.

Pseudocode

```
Shared variables
    const N              // Nb of queues
    queues[1..N]         // queues
    lastVirFinish[1..N] // last virtual finish instant
```

```
receive(packet)
     queueNum :=
chooseQueue(packet)

queues[queueNum].enqueue(packet)
     updateTime(packet,
queueNum)
```

```
updateTime(packet, queueNum)
    // virStart is the virtual
start of service
    virStart := max(now(),
lastVirFinish[queueNum])
    packet.virFinish :=
packet.size + virStart
    lastVirFinish[queueNum] :=
packet.virFinish
```

```
send()
     queueNum := selectQueue()
     packet :=
queues[queueNum].dequeue()
     return packet
```

```
selectQueue()
     it := 1
     minVirFinish = ∞
     while it ≤ N do
         queue := queues[it]
         if not queue.empty and
queue.head.virFinish <
minVirFinish then
             minVirFinish =
queue.head.virFinish
             queueNum := it
         it := it + 1
     return queueNum
```

The function **receive**() is executed each time a packet is received, and **send**() is executed each time a packet to send must be selected, *i.e.* when the link is idle and the queues are not empty. This pseudo-code assumes there is a function **now**() that returns the current virtual time, and a function **chooseQueue**() that selects the queue where the packet is enqueued.

The function **selectQueue**() selects the queue with the minimal virtual finish time. For the sake of readability, the pseudo-code presented here does a linear search. But maintaining a sorted list can be implemented in logarithmic time, leading to a *O(log(n))* complexity, but with more complex code.

See also

- Network scheduler
- Weighted fair queuing
- Weighted round robin
- Generalized processor sharing
- Deficit round robin
- Bufferbloat
- Fairness measure
- Max-min fairness
- Statistical multiplexing
- Active queue management

References

1. *John Nagle: "On packet switches with infinite storage," (http://tools.ietf.org/html/rfc970) RFC 970, IETF, December 1985.*
2. *Nagle, J. B. (1987). "On Packet Switches with Infinite Storage". IEEE Transactions on Communications. **35** (4): 435–438. CiteSeerX 10.1.1.649.5380 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.649.5380) . doi:10.1109/TCOM.1987.1096782 (https://doi.org/10.1109%2FTCOM.1987.1096782) .*
3. *Phillip Gross (January 1986), Proceedings of the 16-17 January 1986 DARPA Gateway Algorithms and Data Structures Task Force (https://www.ietf.org/proceedings/01.pdf) (PDF), IETF, pp. 5, 98, retrieved 2015-03-04, "Nagle presented his "fair queuing" scheme, in which gateways maintain separate queues for each sending host. In this way, hosts with pathological implementations can not usurp more than their fair share of the gateway's resources. This invoked spirited and interested discussion."*
4. *Demers, Alan; Keshav, Srinivasan; Shenker, Scott (1989). "Analysis and simulation of a fair queueing algorithm". ACM SIGCOMM Computer Communication Review. **19** (4): 1–12. doi:10.1145/75247.75248 (https://doi.org/10.1145%2F75247.75248) .*
5. *Demers, Alan; Keshav, Srinivasan; Shenker, Scott (1990). "Analysis and Simulation of a Fair Queueing Algorithm" (http://people.csail.mit.edu/imcgraw/links/research/pubs/networks/WFQ.pdf) (PDF). Internetworking: Research and Experience. **1**: 3–26.*

6. *Bennett, J. C. R.; Hui Zhang (1996). "WF/sup 2/Q: Worst-case fair weighted fair queueing". Proceedings of IEEE INFOCOM '96. Conference on Computer Communications. Vol. 1. p. 120. doi:10.1109/INFCOM.1996.497885 (https://doi.org/10.1109%2FINFCOM.1996.497885) . ISBN 978-0-8186-7293-4.*
7. *Ito, Y.; Tasaka, S.; Ishibashi, Y. (2002). "Variably weighted round robin queueing for core IP routers". Conference Proceedings of the IEEE International Performance, Computing, and Communications Conference (Cat. No.02CH37326). p. 159. doi:10.1109/IPCCC.2002.995147 (https://doi.org/10.1109%2FIPCCC.2002.995147) . ISBN 978-0-7803-7371-6.*

Last edited 12 months ago by MarcBoyerONERA