Fair Allocation and Scheduling in Bonded Channel Systems

Mike Westall

Jim Martin

School of Computing

Clemson University

westall@cs.clemson.edu

http://www.cs.clemson.edu/ westall/homepage.html

Acknowledgement

This authors gratefully acknowledge that this research was supported in part by Cisco Systems, Inc. via Cisco Research Award *DOCSIS 3.0 Scheduling Algorithms* in the amount of $98,000.

Bonded Channels

What are *bonded channels*?

- Two or more physical channels that comprise single logical channel
- May be constituted of different physical media (e.g. two Ethernet cables),
- or be created by multiplexing the bandwidth of a single broadband carrier.

Why use them?

- To increase capacity
- To partition workloads of differing QoS needs
- To obtain greater reliability

Our domain of interest is bonded channels created from frequency division mulitiplexing on digital cable TV (DOCSIS) networks.

Network Flows

What is a *network flow*?

- An identifiable end-to-end entity to which QoS attributes are assigned.
- May consist of one *or more* transport layer connections.
- The flow's identity may be:
  - Explicitly specified in Link Layer header fields
  - Inferred from network and/or transport header data

Fair allocation of capacity

- Suppose a channel has capacity C bps,
- N flows have nominal demands $D_i, i = 1, ..N$ bps,
- and $\sum_{i=0}^{N-1} D_i > C$.

Then it is not possible to satisfy all allocations. When this occurs, a common approach is to favor flows with small demands at the expense of those with large demands.

A common approach for doing this is known as *max-min fair allocation*.

Max-min fair allocation

The name reflects the objective of the technique which is to interatively maximize the allocation to flows with minimal demands.

- An allocation of rates is max-min fair if and only if any rate increase must must necessarily decrease the allocation of a flow having a smaller allocation than the flow whose rate was increased.
- For a single channel or bonding group a solution always exists and a simple $O(nlog(n))$ algorithm can find it.
- Depending upon the problem domain a max-min fair allocation may not exist and may or may not be unique.
- Allocation is performed in order of increasing demand. If a flow's demand is less than or equal to its fair share of the remaning capacity, it receives its demand.
- Otherwise it receives its fair share of the remaining capacity.

Max-min fair allocation algorithm

In a pre-processing step, the *demand* vector is sorted into increasing order. Then the algorithm procedes as follows:

```
left = CAPACITY;
for (i = 0; i < COUNT; i++)
{
   share = left / (COUNT - i);

   if (demands[i] < share)
      allocs[i] = demands[i];
   else
      allocs[i] = share;

   left = left - allocs[i];
   sum += allocs[i];
}
```

- Suppose capacity is 20 Mbps and demands are 2, 5, 9, 11 Mbps.
- Then max-min fair allocation is 2, 5, 6.5, 6.5.

Single bonding group

Suppose a broadband carrier is partitioned into m channels and that

- Each of these channels has capacity C_j $Mbps$, and
- the aggregate capacity of the system is $C = \sum_{j=0}^{m-1} C_j$.

Now suppose that the workload consists of n flows and that

- Each flow has a demand D_i also measured in $Mbps$.
- The system consists of a single bonding group so that,
- each flow is capable of using *any channel*.

Then the max-min fair allocation is simply the max-min fair allocation of the aggregate capacity C.

Multiple bonding groups

Now suppose there is an $n \times m$ binary-valued mapping table, $M = M_{i,j}$, that identifies the channels that each flow is allowed to use. $M_{i,j} = 1$ if flow i is permitted to use channel j, and $M_{i,j} = 0$ if it is not.

Our objective is to construct a capacity allocation table, $A = \{a_{i,j}\}$ where $a_{i,j}$ specifies in $Mbps$ the portion of the capacity of channel j that is allocated to flow i. We call this table a *flow allocation strategy*. Obvious feasibility constraints on a flow allocation strategy are:

- $a_{i,j} > 0$ only if $M_{i,j} = 1$; Use only assigned channels.
- $A_i = \sum_{j=0}^{m-1} a_{i,j} \leq D_i \ \ \forall i$; Row sums can't exceed demand.
- $\sum_{i=0}^{n-1} a_{i,j} \leq C_j \ \ \forall j$; Col sums can't exceed capacity.

Demand and mapping constraints

Depending upon D and M, it may or *may not* be possible allocate the entire aggregate capacity C.

- Suppose the capacity of channel $C_2 = 10Mbps$, and
- $M_{i,2} = 1$ only for flows 1 and 3, and
- $D_1 = 4Mbps$ and $D_3 = 3Mbps$.
- Then no allocation strategy can exceed $7Mbps$ on channel 2

So we seek to determine C_a, the maximum aggregate capacity that can be allocated and to find an allocation table A for which A_i is max-min fair with respect to aggregate capacity C_a.

It is always possible to compute C_a, but depending upon C_j, D_i and $M_{i,j}$ a max-min fair allocation of C_a may or may not exist.

Load sharing domains

The matrix $M_{i,j}$ can be viewed as an adjacency matrix for the graph G in which the vertices consist of flows and channels and an edge exists between $flow_i$ and $channel_j$ i.f.f. $M_{i,j} = 1$. If G is connected, we say that the mapping defines a single *load sharing domain*.

In a load sharing domain, it is *potentially* possible to transfer load among all constituent channels.

If G consists of disjoint subgraphs, each subgraph represents a load sharing domain consisting of flows that share no channels with flows in other load sharing domains. It is *never* possible to transfer load across load sharing domain boundaries.

When a problem consists of multiple load sharing domains, it should be recast as subproblems that are solved independently.

Example

The following mapping consisting of three flows and four channels should be re-cast as two independent systems: the first consisting of flows 0, 1 and channels 0, 1; and the second consisting of flow 2 and channels 2 and 3.

$$M_{i,j} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

If we change the problem so that flow 1 can use channel 2 then it becomes a single load sharing domain.

In the following discussion we assume that $M_{i,j}$ describes a load sharing domain.

Nevertheless, even within a load sharing domain, a solution that maximizes aggregate allocated capacity, C_a and is max-min fair *with respect to C_a may not exist*.

Problem statement

Given

- a demand vector $D_i, i = 0, ..., n-1$
- a channel capacity vector $C_j, j = 0, ..., m-1$, and
- a mapping $M_{i,j}$ that defines a single load sharing domain,

Find

- C_a, the maximum usable aggregate capacity of the domain, and
- $a_{i,j}$, an allocation matrix, the sum of whose values is C_a and such that flow allocations $A_i = \sum_{j=0}^{m-1} a_{i,j}$ are max-min fair with respect to C_a.

Example 1

Let $D_i = \begin{bmatrix} 24 & 8 & 12 & 7 \end{bmatrix}, C_j = \begin{bmatrix} 10 & 10 & 10 & 10 \end{bmatrix}$, and

$$M_{i,j} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}.$$

Then $A_i = \begin{bmatrix} 13 & 8 & 12 & 7 \end{bmatrix}$, is the unique max-min fair allocation of the aggregate 40 Mbps of capacity and

$$a_{i,j} = \begin{bmatrix} 10 & 3 & 0 & 0 \\ 0 & 7 & 1 & 0 \\ 0 & 0 & 9 & 3 \\ 0 & 0 & 0 & 7 \end{bmatrix} \text{ and } a_{i,j} = \begin{bmatrix} 9 & 4 & 0 & 0 \\ 0 & 6 & 2 & 0 \\ 0 & 0 & 8 & 4 \\ 1 & 0 & 0 & 6 \end{bmatrix}$$ are non-unique

realizations of the solution.

Example 2

Let $D_i = \begin{bmatrix} 22 & 3 & 6 & 12 \end{bmatrix}, C_j = \begin{bmatrix} 10 & 10 & 10 & 10 \end{bmatrix}$, and

$$M_{i,j} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

In this case C_a = 39.0 and $A_i = \begin{bmatrix} 20 & 3 & 6 & 10 \end{bmatrix}$, is the unique max-min fair allocation of the usable 39 Mbps of capacity and

$$a_{i,j} = \begin{bmatrix} 10 & 10 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$ is a non-unique realization of the solution.

Graph based approaches

Scott Moser has developed an extension to the Ford-Fulkerson Max Flow algorithm.

In his approach

- All channels are connected to t with edges of actual channel capacity
- Edges exist between flows and channels and have infinite capacity i.f.f. the flow can use the channel.
- Edges between s and flows are initially given capacity 0 which is incrementally increased until fairness breaks down.

The approach presented here is based upon linear optimization. It is believed that both approaches produce the same result.

The linear programming solution

Step 1: C_a is is found by using linear programming to maximize the objective function

$$P = \sum_{j=0}^{m-1} \sum_{i=0}^{n-1} M_{i,j} a_{i,j}.$$

subject to the constraints

$$\sum_{i=0}^{n-1} M_{i,j} a_{i,j} \leq C_j \;\; j = 0, \ldots, m-1$$

$$\sum_{j=0}^{m-1} M_{i,j} a_{i,j} \leq D_i \;\; i = 0, \ldots, n-1.$$

The solution to this linear programming problem will yield

- a feasible allocation strategy $a_{i,j}$ whose sum is C_a.
- but the solution will not be fair.

Fair allocation of capacity

Step 2:

- Sort the demand vector, D_i into increasing demand order
- Use the max-min fair allocation algorithm to compute the vector S_i which is the max-min fair allocation of C_a.

The linear programming solution - step 3

Step 3: Perform another linear programming operation maximizing the objective function

$$P = \sum_{j=0}^{m-1} \sum_{i=0}^{n-1} M_{i,j} a_{i,j}.$$

subject to the constraints

$$\sum_{i=0}^{n-1} M_{i,j} a_{i,j} \leq C_j \quad j = 0, \ldots, m-1$$

$$\sum_{j=0}^{m-1} M_{i,j} a_{i,j} \leq S_i \quad i = 0, \ldots, n-1.$$

If $P = C_a$,

- $a_{i,j}$ is a feasible allocation strategy and
- the solution is max-min fair with respect to C_a.

No solution exists

If the value P computed in step 3 is not equal to C_a, then no solution exists and the solution computed in step 3 will contain:

- One or more channels that are not fully utilized,
- one or more fully utilized channels hosting flows whose allocation A_i is less than the flow's fair share S_i, and
- flows using the underutilized channels will have no allocations on any channel hosting a flow that did not receive its fair share.

The easiest way to achieve both full allocation of capacity and maxmin fairness is to modify $M_{i,j}$ allowing flows not receiving their fair share to use the under utilized channels. Then repeat steps 1 - 3 until max-min fair allocation is observed.

No solution exists

Suppose $D_i = C_j = 10 \ \ \forall i, j$. Here $C_a = 40$, and so fair share is 4 Mbps per flow, but the "fairest" strategy possible under $M_{i,j}$ is:

$$M_{i,j} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_{i,j} = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 3.3 & 0 \\ 0 & 0 & 3.3 & 0 \\ 0 & 0 & 3.3 & 0 \\ 0 & 0 & 0 & 2.5 \\ 0 & 0 & 0 & 2.5 \\ 0 & 0 & 0 & 2.5 \\ 0 & 0 & 0 & 2.5 \end{bmatrix}$$

Approximating max-min fairness

If modifications to $M_{i,j}$ are not desired, then through a series of partitionings it is possible to obtain a solution that is as fair as possible in the sense of the previous example.

- Pick a channel, j, having unused capacity.
- Remove all flows, i, for which $M_{i,j} = 1$ and
- all channels k for which $a_{i,k} > 0$.
- Since the removed flows are guaranteed not to use channels that service underprovisioned flows, some flows and some channels are guaranteed to be in each partition.
- There will be no flows that use both the removed channel(s), k, and also use any channel used by flows that did not receive fair shares. Any such load would have been indirectly transferred back to the underutilized channel and the capacity assigned to the underprovisioned flows.

Now solve the partitioned problem and repeat as necessary until each subproblem has a a solution.

Part II - Scheduling

Tradtional IP scheduling

- Single FCFS queue per channel
- FCFS or Random dropping used to penalize heavy users
- No real QoS guarantees possible

On the positive side it is *simple* and *stateless*

Flow aware scheduling

- Each flow has its own FCFS queue
- The scheduler mission is to choose one of the backlogged queues.
- QoS guarantees become more easily managed

Complications include

- Recognizing when a flow commences and terminates
- Classifying arriving packets by flow.

Soft and hard state approaches have been used.

Round Robin

Round robin -

- Backlogged flows are serviced in circular order
- A flow that is not backlogged when its turn comes forfeits that turn
- Flows sending small packets receive unfairly small fractions of capacity

Deficit Round Robin

Deficit round robin -

- Each scheduling round, a quantum of bytes, commonly equal to the maxium packet size, is added to the *deficit counter* of each backlogged flow.
- The size of each packet sent is deducted from the flow's deficit counter
- While its deficit counter remains positive, a flow can continue to send packets.
- A flow whose queue becomes empty has its deficit counter reset to 0.
- The unfairness of pure round robin is eliminated
- In a weighted variant, WDRR, quantum size is dependent upon a flows "weight"

Both RR and DRR admit very efficient $O(1)$ scheduling costs. DRR and WDRR are presently the preferred algorithms for enforcing QoS guarantees. If all packets are the same size, then *DRR == RR*.

Bonding Groups

A *channel bonding group,* G_k, is a fixed subset of channels that service a defined subset of flows.

- Each *unique* row of the mapping matrix M defines a bonding group.
- All flows having identical mapping rows belong to the same bonding group.
- A mapping is said to define *non-intersecting bonding groups* i.f.f. each channel belongs to exactly one bonding group.
- In this case, each bonding group is a separate load sharing domain, and
- if there exist rows, i and k, of M for which $M_{i,j} = M_{k,j} = 1$ for some j, then rows i and k must be identical.

Scheduling on Bonded Channels

When bonding groups are non-intersecting, both the fair share allocation and scheduling problems are trivial because each bonding group defines a load sharing domain for which $M_{G_{k_{i,j}}} = 1 \forall i, j$.

Fair-share allocation

- For each bonding group G_k, $C_{a_{G_k}} = Min(\sum_{flow_i \in G_k} D_i, \sum_{chan_j \in G_k} C_j)$.
- Each flow's fair share, S_i can be computed with respect to $C_{a_{G_k}}$.
- For each flow i, bound to the group k, $a_{i,j} = S_i / |G_k|$.

Scheduling -

- A table of associated flows is maintained for each bonding group,
- When any channels within the group become idle, the flows are scheduled using the usual (deficit) round robin algorithm.
- From Cisco's perspective it seems that table locking may present some challenges.

Extension to intersecting bonding groups

Suppose we have three flows of equal weight mapped to three channels of equal capacity as:

- Flow 0 - Channels 0 and 1 - Bonding group 0
- Flow 1 - Channel 1 - Bonding group 1
- Flow 2 - Channels 1 and 2 - Bonding group 2

A fair algorithm would work as follows:

- All flows backlogged: Flow 0 uses only Channel 0, Flow 1 Channel 1, and Flow 2 Channel 2.
- Flow 1 idle: Flows 0 and 2 equally share Channel 1
- Flow 0 idle: Flow 1 uses Channel 1 and Flow 2 uses Channel 2
- Flow 2 idle: Flow 1 uses Channel 1 and Flow 0 uses Channel 0

Scheduling intersecting bonding groups

There are two obvious ways to extend (D)RR to intersecting bonding groups.

- There is a single RR list that contains all flows through the router
- There is a single RR list for each channel containing all the flows that the channel can serve.
- In the second case, a single flow can appear on multiple lists.

Unfortunately both of them are unfair. The first approach is considerably worse than the second and can produce complete starvation of flows.

The second approach will produce an unfair $4/3, 1/3, 4/3$ allocation of the total capacity under (D)RR when all three flows are backlogged.

Other more fair refinements to the single list approach exist, but require use of true linked list structures instead of circular tables.

Fluid based schedulers

Designed to approximate the behavior of (weighted) bit level RR

- (Worst case fair) Weighted Fair Queuing provides the best approximation (from a fairness perspective) to a fluid system, but has significant computational overhead.
- Self-Clocked Fair Queuing provides a good approximation at considerably less cost.
- Both have been shown (along with DRR) to *inherently* provide maxmin fair allocation of capacity of a single channel.
- Both extend naturally to intersecting bonding groups.

Implementing WFQ/SCFQ

Implementing true WFQ requires building a simulation of a fluid server and running it in parallel with the actual scheduler, but Golestani's SCFQ is quite simple.

- Each arriving packet is stamped with a virtual clock value sometimes called a *service tag*.
- The scheduler maintains a *system virtual clock* which is the service tag of the packet currently in service.
- The *cost* of an arriving packet is its transmission time divided by its flows (fixed) share of the channel.
- The service tag of an arriving packet is set to its cost + max(service tag of the flow's last arriving packet, and the system virtual clock).

Whenever no flows are backlogged, the system virtual clock and the last packet timestamp are reset to 0.

Implementing SCFQ on bonded channels

The following data structures are required

- Each flow maintains a FIFO queue of backlogged packets
- Each packet is carries a *a unique service tag for each channel* that its flow is allowed to use.
- Each flow maintains a "last service tag" for each channel that it is allowed to use.
- Each channel maintains a priority queue of eligible flows, sorted by service tag of the packet at the head of the flow's FIFO queue.
- Backlogged flows appear in the priority queues of all eligible channels.

Implementing SCFQ on bonded channels

In operation:

- When a channel becomes idle, the scheduler removes the head packet from the FIFO queue of the flow at the top of the priority queue initiates its transmission.
- The priority queues of all channels that the flow is eligble to use *must then be updated* :-(
- Unlike variants of RR, both WFQ and SCFQ appear to *automagically provide* max-min fair allocation in the sense described earlier.

Unfortunately, "real routers" still implement routing algorithms in firmware and implementation of priority queues and locking mechanisms for accessing flows may be a "bridge" too far.