Algorithms for Fair Multicast in Virtual Circuits and Stable Stochastic Parallel Queues

Davey Proctor

March 26, 2019

1 Multicast in Virtual Circuits

We have a network represented by a capacitated directed graph $G = (V, E)$ and a capacity function $C : E \to \mathbb{R}^+$. The network supports multicasting and broadcasting. A multicast session $n \in \{0, \ldots, N\}$ is described by (s_n, V_n) where s_n is the source node of the session where traffic is injected in the network and V_n is the set of destination nodes that should receive the traffic. For each session n there is a directed tree T_n associated with the session serving its traffic. T_n is rooted in s_n, its leafs are destination nodes in V_n while some nodes in V_n might be intermediary nodes of T_n. Traffic flows from s_n to all nodes in V_n through the tree T_n.

1.1 Same Multicast, Same Rate

Suppose all the nodes in the same multicast session n receive the same rate r_n. I present an algorithm to compute the lexicographically optimal multicast rates.

Definition 1.1. Let $T(e) \subseteq \{0, \ldots, N\}, e \in E$ be the trees that share edge e.

$$T(e) = \{n : e \in T_n, n \in \{0, \ldots, N\}\}$$

Algorithm 1.2. Initially, set all $r_n = 0$ and precompute all $T(e)$. Repeat until all sessions have flows ($r_n > 0$):

1. Determine the bottlenecking edge $\widehat{e}$, defined as

$$\widehat{e} = \arg\min_{e \in E} \frac{C(e)}{|T(e)|}$$

2. For all trees $n \in T(\widehat{e})$:

   (a) Set the flows on those trees to respect that bottleneck.

$$r_n \leftarrow \frac{C(\widehat{e})}{|T(\widehat{e})|}$$

(b) For all edges $e \in T_n$:

i. Compute the capacity that remains.

$$C(e) \leftarrow C(e) - r_n$$

ii. Remove the already-allocated trees from consideration.

$$T(e) \leftarrow T(e) \setminus \{n\}$$

Theorem 1.3. *The algorithm runs in $O(N * |E|)$ and produces feasible rates that are lexicographically optimal subject to the stated constraints.*

For the sake of analysis, it is helpful to introduce the following dummy variables.

Definition 1.4. Suppose $C_i(e), T_i(e), r_n^i, \widehat{e}_i$ are the values of the algorithm variables $C, T, r, \widehat{e}$ at the i^{th} iteration of the algorithm. For instance, upon initialization, we stated that $C_0(e) = C(e), T_0(e) = T(e)$ (C, T as in the problem statement, not the algorithm definition). We also introduce $\widehat{T}_i(e)$, tracking the already-solved sessions sharing a certain edge:

$$\widehat{T}_i(e) = \{n : r_n^i > 0\}$$

The following invariants hold for i up to and including the termination of the algorithm, not necessarily assuming that it does terminate. The first two invariants align the algorithm variables with what they are supposed to represent mathematically; the remaining are substantive ways in which the algorithm's behavior is desireable.

Invariant 1.5. Variable $T : (T_0, T_1, \dots)$ holds the sessions that have not yet been capacitated. That is, for all i, for all $e \in E$,

$$T(e) = T_i(e) \cup \widehat{T}_i(e).$$

Proof. Holds due to the update in step 2(b)ii, because the flows set in step 2a apply by definition to all edges of the corresponding tree. □

Invariant 1.6. Variable $C : (C_0, C_1 \dots)$ always holds the remaining capacity implied by flow values r. For all i, for all $e \in E$:

$$C(e) = C_i(e) + \sum_{n \in T(e)} r_n^i$$

Proof. By induction on i. Trivially true for $i = 0$, since all $r_n^0 = 0$. Suppose it holds for some $i \geq 0$. We wish to show it for $i+1$. In step 2(b)i, The algorithm decrements the capacity by the values of the newly-solved sessions. That is, for

all $e \in E$:

$$C_{i+1}(e) = C_i(e) - \sum_{n \in \widehat{T}_{i+1}(e) \setminus \widehat{T}_i(e)} r_n^{i+1} \tag{1}$$

$$= C(e) - [C(e) - C_i(e) + \sum_{n \in \widehat{T}_{i+1}(e) \setminus \widehat{T}_i(e)} r_n^{i+1}] \tag{2}$$

$$= C(e) - [\sum_{n \in T(e)} r_n^i + \sum_{n \in \widehat{T}_{i+1}(e) \setminus \widehat{T}_i(e)} r_n^{i+1}] \tag{3}$$

$$= C(e) - [\sum_{n \in \widehat{T}_i(e)} r_n^i + \sum_{n \in \widehat{T}_{i+1}(e) \setminus \widehat{T}_i(e)} r_n^{i+1}] \tag{4}$$

$$= C(e) - \sum_{n \in \widehat{T}_{i+1}(e)} r_n^i \tag{5}$$

$$= C(e) - \sum_{n \in T(e)} r_n^i \tag{6}$$

Equation (3) uses the inductive hypothesis; equations (4) and (6) use Invariant 1.5 relating trees to already-allocated vs. zero-valued flows. □

Invariant 1.7 (Feasible Flows)**.** For all $e \in E$:

$$\sum_{n \in T(e)} r_n \leq C(e).$$

Proof. Intuitively, the inequality holds because we decrement the remaining capacity by the amount already used at each iteration. Within one iteration, that the new flows sum to the bottlenecking edge implies that no edge elsewhere will have its capacity overflowed.

More formally, Invariant 1.6 gives us that it is equivalent to show that for every edge, newly allocated rates sum to less than or equal to the current capacity on that edge. That is, we seek to show:

$$\sum_{\widehat{T}_{i+1}(e) \setminus \widehat{T}_i(e)} r_n^{i+1} \leq C_i(e)$$

We start with the left hand side, noticing what the algorithm does in step 2a. For all $e \in E$:

$$\sum_{\widehat{T}_{i+1}(e) \setminus \widehat{T}_i(e)} r_n^{i+1} = |\widehat{T}_{i+1}(e) \setminus \widehat{T}_i(e)| * \frac{C_i(\widehat{e}_i)}{|T_i(\widehat{e}_i)|} \tag{7}$$

$$\leq |T_i(e)| * \frac{C_i(\widehat{e}_i)}{|T_i(\widehat{e}_i)|} \tag{8}$$

$$\leq C_i(e), \tag{9}$$

Given that $\widehat{e}_i$ is the bottleneck ($\widehat{e} = \arg\min_{e \in E} \frac{C(e)}{|T(e)|}$) from step 1. Equation (8) again uses Invariant 1.5. □

Invariant 1.8 (Bottleneck values never decrease)**.** While sessions are unsolved and a bottlenecking edge can be found, the value of the incremental allocations to the competing sessions are non-decreasing. That is, for all i,

$$\frac{C_i(\widehat{e}_i)}{|T_i(\widehat{e}_i)|} \leq \frac{C_{i+1}(\widehat{e}_{i+1})}{|T_{i+1}(\widehat{e}_{i+1})|}$$

Proof. Starting with the right hand side, the first equality reflects the actions of the algorithm on capacities in 2(b)i and trees in 2(b)ii:

$$\frac{C_{i+1}(\widehat{e}_{i+1})}{|T_{i+1}(\widehat{e}_{i+1})|} = \frac{C_i(\widehat{e}_{i+1}) - \frac{C_i(\widehat{e}_i)}{|T_i(\widehat{e}_i)|} * |\widehat{T}_{i+1}(\widehat{e}_{i+1}) \setminus \widehat{T}_i(\widehat{e}_{i+1})|}{|T_i(\widehat{e}_{i+1})| - |\widehat{T}_{i+1}(\widehat{e}_{i+1}) \setminus \widehat{T}_i(\widehat{e}_{i+1})|} \tag{10}$$

$$= \frac{\frac{C_i(\widehat{e}_{i+1})}{|T_i(\widehat{e}_{i+1})|} * |T_i(\widehat{e}_{i+1})| - \frac{C_i(\widehat{e}_i)}{|T_i(\widehat{e}_i)|} * |\widehat{T}_{i+1}(\widehat{e}_{i+1}) \setminus \widehat{T}_i(\widehat{e}_{i+1})|}{|T_i(\widehat{e}_{i+1})| - |\widehat{T}_{i+1}(\widehat{e}_{i+1}) \setminus \widehat{T}_i(\widehat{e}_{i+1})|} \tag{11}$$

$$\geq \frac{\frac{C_i(\widehat{e}_i)}{|T_i(\widehat{e}_i)|} * |T_i(\widehat{e}_{i+1})| - \frac{C_i(\widehat{e}_i)}{|T_i(\widehat{e}_i)|} * |\widehat{T}_{i+1}(\widehat{e}_{i+1}) \setminus \widehat{T}_i(\widehat{e}_{i+1})|}{|T_i(\widehat{e}_{i+1})| - |\widehat{T}_{i+1}(\widehat{e}_{i+1}) \setminus \widehat{T}_i(\widehat{e}_{i+1})|} \tag{12}$$

$$= \frac{C_i(\widehat{e}_i)}{|T_i(\widehat{e}_i)|} \tag{13}$$

The inequality in (12) reflects the fact that in step i, $\widehat{e}_i$ was in fact the bottleneck, achieving as low a bottleneck flow value as any other edge (including $\widehat{e}_{i+1}$). □

Proof of Theorem 1.3. Invariants 1.7 and 1.8 together imply that the algorithm terminates: we can't keep finding non-decreasing flow values that satisfy capacity constraints forever! Invariants 1.5 and 1.8 imply that all sessions have flows (lest trees remain in T_i and a bottleneck can be found).

Those invariants, together with Invariant 1.6 imply the session rates are lexicographically optimal upon termination: the algorithm is repeatedly solving the maxmin flow problem (within an iteration an even split of the bottlenecking flow is the best flow that any tree sharing the bottlenecking edge can obtain). We also decrease the remaining capacity appropriately. These facts imply lexicographical optimality.

The runtime is in $O(N*|E|)$, because each big iteration runs in $O(|T(\widehat{e})|*|E|)$ steps, solving $|T(\widehat{e})|$ sessions. □

1.2 Tree paths can have different rates

Each node k in the multicast group n may receive at its own rate r_k^n. The rate of the source then will be the maximum of the rates of all the nodes in the session and the rate at each link of the tree will be the maximum of all the nodes of the session in the subtree emanating from that link. Again, I describe an algorithm to compute the lexicographically optimal rates for all nodes and trees in the network.

Definition 1.9 (Subtrees from edges)**.** An edge of a tree can characterize a subtree in the following way. For all trees $n \in \{0, \ldots, N\}$ and edges $e \in T_n$, Let $T_{n,e}$ be the subtree of T_n below and including edge e. Formally, if P are the paths of tree n, an edge e' is in $T_{n,e}$ iff there exists a path $p = (v_0 = s_n, v_1, \ldots, v_k) \in P$ such that $e = (v_i, v_{i+1})$ for some $i : 0 \leq i < k$ and $e' = (v_j, v_{j+1})$ for some $j : i \leq j < k$. A vertex v is in $T_{n,e}$ if $v = v_l$ for some $l : i \leq l \leq k$. Let $V_{n,e} = V_n \cap T_{n,e}$ be the vertices of V_n that are in $T_{n,e}$.

Algorithm 1.10. Given similar initializations from the previous algorithm, again we repeat until all flows are set:

1. Determine the bottlenecking edge $\widehat{e}$, defined as

$$\widehat{e} = \arg\min_{e \in E} \frac{C(e)}{|T(e)|}$$

2. For all trees $n \in T(\widehat{e})$:
   (a) For all receiver nodes $k \in V_{n,e}$:
      i. Set the flows on those *paths* to respect that bottleneck.

$$r_k^n \leftarrow \frac{C(\widehat{e})}{|T(\widehat{e})|}$$

   (b) For all edges $e \in T_{n,\widehat{e}}$:
      i. Compute the capacity that remains.

$$C(e) \leftarrow C(e) - \frac{C(\widehat{e})}{|T(\widehat{e})|}$$

      ii. Remove the already-allocated subtrees from consideration.

$$T(e) \leftarrow T(e) \setminus \{n\}$$

The proof of the following is left to the reader:

Theorem 1.11. *The algorithm runs in $O(N * |E|)$ and produces feasible rates that are lexicographically optimal subject to the stated constraints.*

Next Directions. We treated the multicast trees as given, but in general there are many trees that would satisfy a multicast in a directed network from source s_n to receiver nodes V_n. The problem statement is the same as before, except that optimization additionally occurs over this space of possible trees. The challenge is to find an efficient algorithm for computing this new lexicographical optimum (not simply iterating the above on all possible sets of trees).

2 Work Conservation Stabilizes Parallel Queues

A server provides service to a set of N parallel queues. Queue n has an i.i.d. arrival stream $A_n(t), t = 1, 2, \dots$ and if the server is allocated to queue n at time slot t, service is successful according to a binary variable $M_n(t)$ where $M_n(t), t = 1, 2, \dots$ is an i.i.d. process. A service policy specifies the queue $R(t)$ that is receiving service at time t. A policy is work conserving if $R(t)$ is always one of the nonempty queues at slot t, if there is at least one. Let $a_n = E[A_n(t)], m_n = E[M_n(t)], n = 1, \dots, N$.

Theorem 2.1. *Under a work conserving policy the system will be stable if*

$$\sum_{n=1}^{N} \frac{a_n}{m_n} < 1.$$

Proof. Under a work conserving policy, the queue lengths $X(t) \in \mathbb{Z}_+^N, t \in \{1, 2, \dots\}$ describe a Markov chain over the state space $S = \mathbb{Z}_+^N$. A policy Π is represented as a function from the state space to the action space $R(t) \in \{0, \dots, N\}$. To show stability of a work conserving policy given $\sum_{n=1}^{N} \frac{a_n}{m_n} < 1$, it suffices to present a Lyapunov function with negative drift.

The Lyapunov function is

$$V(X_1, \dots, X_N) = \sum_{i=1}^{N} \frac{X_i}{m_i}$$

For all $\delta > 0$, we have that

$$|\{V(X) < \delta : X \in S\}| < \infty.$$

To show negative drift, we are allowed to ignore a finite subset of the state space. The subset will be the single state with all empty queues, $\langle 0, \dots, 0 \rangle$, in which the service is idle. Suppose $R(t) = i$ for some $i \in \{0, \dots, N\}$. Over $S \setminus \{\langle 0, \dots, 0 \rangle\}$, the drift is negative:

$$\begin{aligned}
\text{drift} &= \mathbb{E}[V(X(t+1)) - V(X(t)) \mid X(t) = \langle X_1, \dots, X_N \rangle] \\
&= \mathbb{E}[V(X_1 + A_1(t), \dots, X_{i-1} + A_{i-1}(t), X_i(t) + A_i(t) - M_i(t), \dots) \\
&\quad - V(X_1, \dots, X_N)] \\
&= \mathbb{E}[-M_i(t) * \frac{1}{m_i} + \sum_{n=1}^{n} A_n(t) * \frac{1}{m_n}] \\
&= -m_i * \frac{1}{m_i} + \sum_{n=1}^{N} \frac{a_n}{m_n} \\
&= -1 + \sum_{n=1}^{N} \frac{a_n}{m_n} \\
&< -\epsilon.
\end{aligned}$$

for some positive ϵ. $\square$

Intuition. The key insight is the weighted Lyapunov. While it's true that when working on a slow-to-empty queue, the overall lengths may be increasing, the other queues are together easy enough via $\sum_{n=1}^{N} \frac{a_n}{m_n} < 1$ such that taking more time for harder queues is justified.

One can view V as encoding our weighted duress should hard-to-empty queues have long lengths. $\frac{X_n}{m_n}$, by iterated first-success distributions, is the expected number of pulls on queue n before its current backlog is all serviced.

Finally, the result makes sense in that $\sum_{n=1}^{N} \frac{a_n}{m_n} < 1$ is necessary for stability under any policy; it reflects the principle that stability requires expected arrivals be less than expected service.

Next Directions. Suppose the condition is not satisfied, and the system is not stable under any policy. We seek lexicographically optimal rates $r_1, \ldots, r_n$ to service each queue constrained by $\sum r_i \leq 1$. The rates should be work conserving in the sense that in the long run, the service should not be idle (pulling on a queue that is empty) for any positive ϵ fraction of the time. Without this constraint, $r_i = \frac{1}{n}$ is trivially lexicographically optimal but fails insofar as easier queues are pulled on too frequently, often to the point of being empty.