Robert Gallager's Minimum Delay Routing Algorithm Using Distributed Computation

Timo Bingmann and Dimitar Yordanov

Abstract

One of the computer science papers most worth reading is Gallager's algorithm for minimum delay routing. The merit of Gallager's paper is its rigorous mathematical approach to a problem, which is more often taken care of using heuristics. The approach is founded on a well designed mathematical network model, which is custom-tailored to describe the minimum total delay routing problem. Mathematical observations on the model lead to two conditions for achieving global optimization, which are based on the marginal delay of links and neighbors. From these observations and conditions an iterative, distributed routing algorithm is naturally derived. Gallager finishes his analysis by proving in detail that the algorithm achieves total minimum delay routing. Algorithm and model are reviewed and illustrated in detail in this technical report.

1 Overview

In small computer networks all nodes are connected by a single medium, on which messages are transferred. But as networks grow larger, they have to be broken up into smaller subnetworks with router nodes mediating traffic between two or more transfer mediums. These router nodes require information on where packets need to be forwarded in order to arrive at their desired destination. Once the network grows sufficiently large, it is no longer possible to configure routes manually.

Research for good routing algorithms began in the early days of networking. This technical report studies the routing algorithm presented in the year 1977 by Robert Gallager in [Gall77]. Figure 1 shows how far the ARPANET, a predecessor of today's Internet, was evolved around the time Gallager published his routing algorithm. In 2006, Jim Kurose included Gallager's paper in his list of the ten most recommended computer science papers [Kuro06]. This overview section classifies routing algorithms by several key characteristics. The classification leads to the main goals achieved by Gallager's algorithm.

1.1 Goals of Routing Algorithms

A routing algorithm can have different design goals and depending on the goals different methods must be employed.

Most algorithms try to achieve "good" or even *optimal routing*. Here quality of routing is usually measured by a routing metric. Metrics are described in the next subsection. However this first optimization goal must be achieved within physical constraints and without losing sight of other important aims like:

- A routing algorithm should require only *little network overhead* to keep bandwidth available to the users.

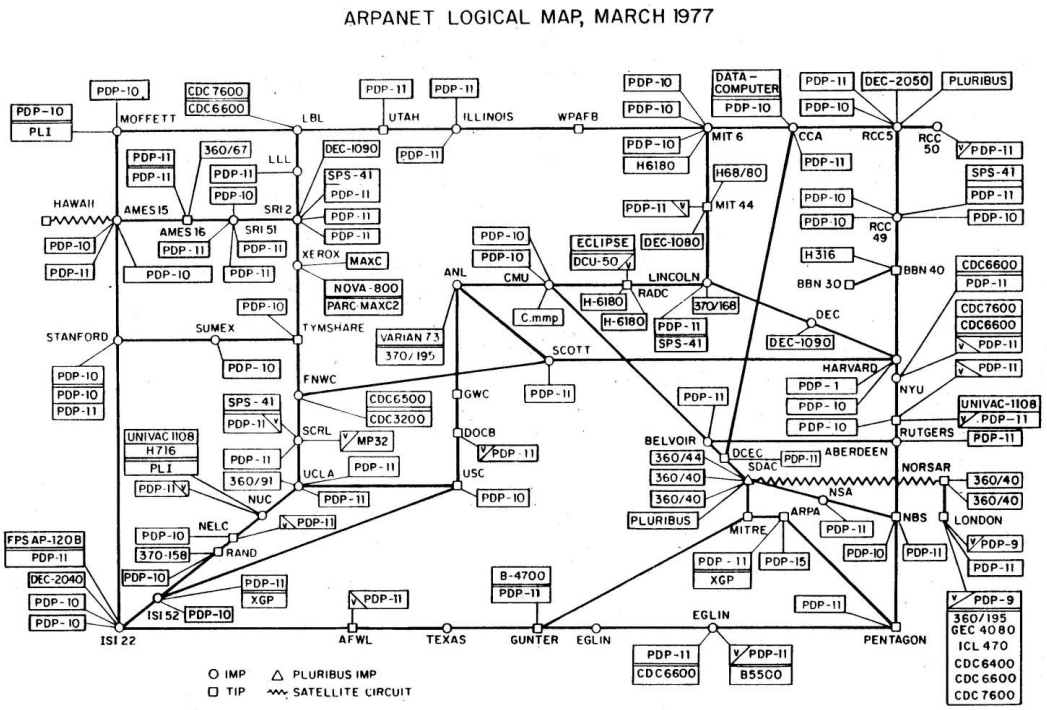

Figure 1: Topology of the ARPANET in March 1977 from [HMMW78]

- Furthermore today's Internet economy depends on *stability* and *reliability* of the underlying network routing calculations.
- Another aspect of quality for routing algorithms is how fast the network *adapts* to new conditions, especially in case of failures. The newly computed routes should *converge* to an optimal state quickly.
- The application of routing algorithms are not confined to small networks. For use in large networks or even the Internet, a routing algorithm's complexity must *scale well* relative to the number of nodes.

As often with a list of goals, it is almost impossible to achieve all at once. Each routing algorithm weighs the importance of above goals differently and thus achieves different optimizations.

1.1.1 Routing Metrics

By using routing metrics, the quality of a set of routing variables or routing tables can be measured. A metric defines the "distance" between two neighbors or two nodes. Only then can an optimal routing setting be defined.

A routing metric is usually a weighted combination of following base metrics; however a network administrator can make custom changes as required.

One of the base routing metrics is the imaginary *path length* of a link. If the length is set to 1 regardless of the actual link, then the resulting metric is called *hop count*. Beyond these base lengths, different weights representing link qualities like *reliability*, *load* and *bandwidth* can be used to sway the metric into a desired direction. External properties like *cost* or operator peering *policy* must also be regarded in actual network environments.

One widely used metric is *routing delay*, because it gives a good overall, policy-free measurement. It represents the delay a packet requires to travel from one node to another. This delay depends on many network properties such as bandwidth of travelled links, queue states at each router along the way, network congestion on intermediate links, and the physical distance to be travelled. Optimizing routing delay gives the user the fastest network experience.

1.2 Characteristics of Routing Algorithms

1.2.1 Route Calculation Over Time

Having investigated the primary goals of routing algorithms, this section tries to highlight some distinct design classes. A first classification of routing algorithms can be made by observing when routing tables and variables are changed. See [Sega77] for detailed definitions and analytic methods for the second two classes.

Static routing algorithms calculate routing tables and configure nodes *once*. During operation of the network no changes are made, unless an administrator requests reconfiguration because of link failure or addition. Good algorithms exist to calculate routing tables in fixed networks; however static routing has no mechanism to adapt to changing traffic requirements.

Dynamic routing algorithms allow much greater flexibility: when a packet is received by a routing node, an algorithm decides *ad-hoc* how to forward the packet. The decision may be based on the instantaneous states of outgoing queues and other variables. In extreme cases the routing tables are recalculated for each packet. This large amount of variables makes formal analysis of dynamic routing algorithms difficult.

Quasi-static routing algorithms are a trade-off between the other two strategies. At specific time points a routing algorithm may change routing tables to adapt to new requirements. Usually this is based on a *periodic* updating mechanism, in which nodes exchange variables needed to improve their routing tables. The nodes may need several update cycles to adapt to changed demands.

1.2.2 Other Characteristics

Single-Path vs. Multi-Path Algorithms
Traffic flow exiting a node towards its destination can either be sent over a single link or distributed to multiple neighbors. Single-path algorithms are much easier to evaluate and implement, but cannot take advantage of the network's full bandwidth. In larger networks this deficiency becomes intolerable. However determining the correct fractions of traffic routed to neighbors makes multi-path algorithms very complex.

Centralized vs. Distributed Algorithms
Routing algorithms can also be distinguished by the location where new routing information is calculated. Centralized algorithms depend on one special node which collects information and computes new routes. In the distributed method each node calculates new optimized routing variables from locally available information. Hence the calculation is done on each node and information is cooperatively exchanged.

The centralized method has some obvious draw-backs. It suffers from the chicken-and-egg problem: in order to be able to calculate new routes the central node needs information about the nodes on the network. However, this information has to be transmitted to the central node without having established transmission routes. Secondly special bridge links or cut points may fail so that no failure information can reach the central node. And lastly the

central node itself is prone for system failure and attacks. Because of these problems most modern approaches rely on distributed routing algorithms.

User vs. System Optimization
Another characteristic by which (distributed) routing algorithms can be divided into two groups is based on the scope of their optimization goals. A distributed algorithm can try to optimize routing variables locally and achieve local *user optimal* goals. For example it could balance the load on all its outgoing links. If, however, an algorithm tries to achieve global goals by having the nodes cooperate with each other, then this is called *system* optimization. The algorithm presented by Gallager optimizes total system delay.

1.3 Goals and Characteristics of Gallager's Algorithm

In this technical report the distributed, multi-path, quasi-static algorithm presented by Gallager in [Gall77] is reviewed. It minimizes total routing delay and achieves global system optimization. The routing delay metric is used in the algorithm, however any other convex, continuous valuation function will work as well. Then the custom defined metric function is minimized instead of total delay.

1.3.1 Knowledge at that Time

As depicted in figure 1 in 1977 the ARPANET was widely spread throughout the United States. The original routing algorithm implemented in the ARPANET was designed in 1969 (see [McFR78] for an overview). It tries to direct each packet along the path with smallest total estimated transfer time. These paths are periodically calculated by the IMPs (Internet Message Processors), the routers of that time. Their computations are based on the delay to their direct neighbors and routing information which is exchanged between them. Each IMP sends its newly computed routing table to each neighbor, so the neighbors in turn can update their own path tables. Thus each IMP has estimations of the total delay to each destination IMP and a best outgoing link for these packets.

1.3.2 Comparison to Today's Routing Protocols

Today the two most widely spread routing protocols in the Internet are OSPF and BGP. Both are routing *protocols* and thus define how routing information is configured, managed and exchanged, while the actual computation of routes is done using specific routing *algorithms*. See [KuRo03] for more details on the two protocols.

Open Shortest Path First (OSPF) is used as interior gateway protocol within autonomous systems to manage local routing. It uses Dijkstra's algorithm to calculate a *shortest path tree* regarding an custom-defined metric. Thus the whole network topology including metric information must be known to each node. Dijkstra's original algorithm computes only a single best path. However, due to the Internet's size, OSPF will find multiple routing paths with the same distance and use both equally.

Today the Border Gateway Protocol (BGP) is used between the autonomous systems of the Internet. BGP is a *path vector protocol*, which is derived from the basic *distance vector protocol*. This protocol's algorithm is similar to Gallager's algorithm. It also builds a distance table to each node and updates it by calculating the cost of the next hop plus the neighbor's currently known minimum cost to the destination. However BGP will typically install only a single route for each destination. Therefore BGP is a single-path, distributed, dynamic routing protocol with many further security and management features.

2 Model and Algorithm

2.1 Model

To grasp the routing challenge mathematically, Gallager formalizes the following network model. Figure 2 illustrates a small example network containing four nodes.

A network consists of n nodes, which are enumerated by the integers $1, \dots n$. A link from node i to node j is represented by the tuple (i, j). It is assumed that if (i, j) exists, then the reverse link (j, i) exists as well. This way neighbors can exchange routing information. Let $\mathcal{L} := \{(i, j) \text{ is existing link}\}$ the set of links.

An essential input to the algorithm is the amount of traffic, which enters the network at a specific source node i and exits it at the destination node, provided that a route exists. This input is defined by $r_i(j) \geq 0$ (e.g. in bits/s) as the traffic entering at node i destined to node j. Thus when the network is connected and routing works, $\sum_{j=1}^{n} r_i(j)$ traffic enters at node i and $\sum_{i=1}^{n} r_i(j)$ traffic exits at node j. All $r_i(j)$ are grouped into a set $\boldsymbol{r}$ named *input* set.

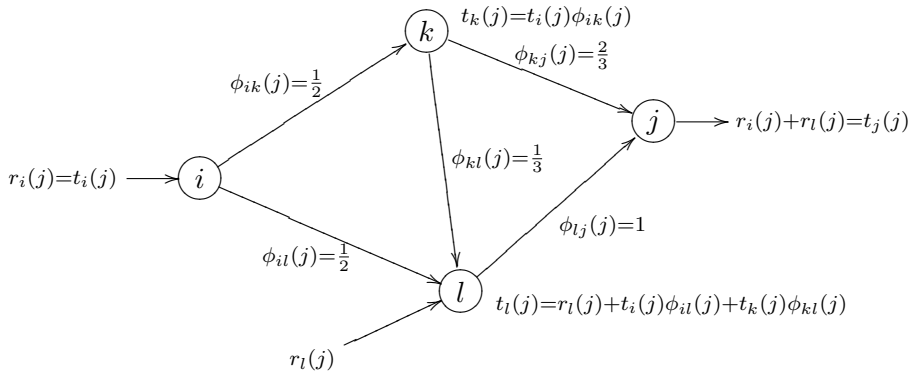

Figure 2: Example network of four nodes

The traffic $r_i(j)$ is supposed to flow from node i to node j. Usually intermediate nodes are required along the way and routing variables decide by which links the packets find their destination. At these intermediate nodes further traffic destined for node j can enter the network or be received from other sources. The sum over all traffic at a node i destined for node j is defined as $t_i(j)$. These values are not input values; a routing algorithm determines them by guiding network flow. The set of all $t_i(j)$ is called $\boldsymbol{t}$, the *node flow* set.

An algorithm determines with routing variables $\phi_{ik}(j) \geq 0$ the fraction of the traffic $t_i(j)$ which flows over the link (i, k) towards node j. So $t_i(j)\phi_{ik}(j)$ units of traffic flow over link (i, k). Name the set of all variables $\phi_{ik}(j)$ the *routing variable* set $\boldsymbol{\phi}$. Such a set must satisfy the following three constraints:

1. It is assumed that $\phi_{ik}(j) = 0$ for non-existing links ($(i, k) \notin \mathcal{L}$) and $\phi_{ik}(i) = 0$ as no traffic is send out again after arriving at destination node i.

$$\phi_{ik}(j) = 0 \quad \forall\, (i, j) \notin \mathcal{L} \text{ or } i = j \tag{1a}$$

2. All traffic aggregated at a node must be sent out over a link. No loss of traffic is allowed.

$$\sum_{k=1}^{n} \phi_{ik}(j) = 1 \quad \forall\, i, j \tag{1b}$$

3. All nodes are inter-connected: for each pair of nodes (i, j) a routing path from i to j exists.

$$\phi_{ik}(j) > 0, \phi_{kl}(j) > 0, \ldots, \phi_{mj}(j) > 0 \quad \exists\, i, k, l, \ldots, m, j \,\forall\, i, j \tag{1c}$$

Equation 2 shows a first dependency between the three sets of variables. It describes the aggregation of traffic at node i destined for node j.

$$t_i(j) = r_i(j) + \sum_{l=1}^{n} t_l(j)\phi_{li}(j) \tag{2}$$

The equation above describes only the flow of traffic towards node j, the whole network usually carries flows towards each node. This one traffic flow is regarded at node i and may be spread out and sent over different links towards node j as described by $\phi_{ik}(j)$. The link (i, k) carries $t_i(j)\phi_{ik}(j)$ units of traffic (e.g. bits/s) towards node j. The total traffic over all flows in the network carried by link (i, k) is then

$$f_{ik} = \sum_{j} t_i(j)\phi_{ik}(j) \tag{3}$$

For $(i, k) \notin \mathcal{L}$ set $f_{ik} = 0$.

The model does not specify a maximum link capacity directly. Instead the main focus is directed to the delay per unit of time D_{ik} which a link (i, k) introduces into the network. The delay is assumed to depend only on the amount of traffic flowing over the link: $D_{ik}(f_{ik})$. This delay includes processing time, send queuing delay, transfer time and arrival queues. Restricting the dependency of D_{ik} to the traffic amount simplifies analysis while simultaneously masking some complex properties of routers. For example packet sizes are disregarded, even though routers may require considerably more time to process many small packets than few large packets even though the total traffic rate is equal.

The actual delay function is only assumed to be convex and increasing. A reasonable example is $D_{ik}(f_{ik}) = \frac{f_{ik}}{C_{ik}-f_{ik}}$, where C_{ik} is the capacity of the link (i, k). Thus $D_{ik}(f_{ik}) \to \infty$ for $f_{ik} \to C_{ik}$. More complex analysis of analytic delay functions can be found in [Klei70].

Coming to the main objective of the routing algorithm let

$$D_T = \sum_{i,k} D_{ik}(f_{ik}) \tag{4}$$

This sum describes the total delay in the network and will be minimized by the algorithm by setting optimal routing variables $\boldsymbol{\phi}$. This is possible because D_T depends on f_{ik}, which again are functions of the sets $\boldsymbol{r}$ and $\boldsymbol{\phi}$ (equation 3). The sum D_T grows large if any of the $D_{ik}(f_{ik})$ grows large because the traffic f_{ik} exceeds some limitation like link capacity.

2.1.1 Routing Variables

To show that an algorithm which modifies only $\boldsymbol{\phi}$ will actually guide the network's flow, Gallager proves that a given input set $\boldsymbol{r}$ and a routing variable set $\boldsymbol{\phi}$ uniquely define the network flow set $\boldsymbol{t}$.

To prove this, first let $\Phi_j = (\phi_{ik}(j))_{1 \le i,k \le n}$ be the square $n \times n$ matrix containing the routing variable set. The two constraints $\phi_{ik}(j) \ge 0$ and equation 1b are exactly the defining properties of a stochastic matrix. By temporarily setting $\phi_{ji}(j) := \frac{r_i(j)}{\sum_k r_k(j)}$, let the appropriate fraction of all arriving traffic take an imaginary link from the destination back to the source

node. This creates steady traffic flow cycles in the network and the equation 2 can formally be contracted to

$$t_i(j) = \sum_{l=1}^{n} t_l(j)\phi_{li}(j) \tag{5}$$

This is the equation of a Markov chain (see [Behr00]) with an equilibrium distribution: $\bar{t}_j = \bar{t}_j \Phi_j$, with $\bar{t}_j = (t_i(j))_{1 \le i \le n}$ the vector of network flows. If it can be shown that Φ_j is a irreducible Markov transition matrix, then the Markov chain has exactly one solution $\bar{t}_j$ except for a scale factor. This scale factor corresponds to the absolute traffic arriving at node j. A Markov chain is irreducible if for all states i, j the state j is accessible from i (starting in i there is a non-zero probability that state j is reached). This is equivalent to the third constraint of $\boldsymbol{\phi}$ (see equation 1c). Thus given $\boldsymbol{r}$ and $\boldsymbol{\phi}$ the network flow $\boldsymbol{t}$ is uniquely defined by equation 2.

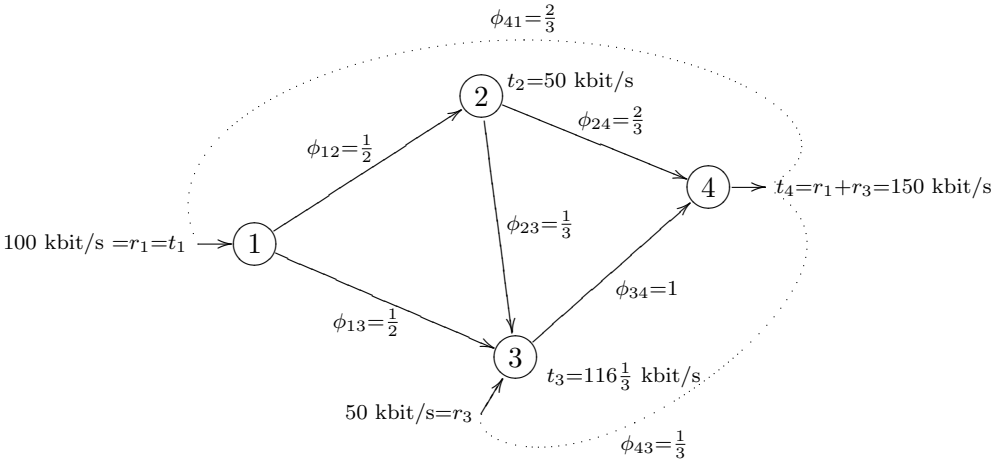

Figure 3: Example with calculated traffic flow and imaginary links

This result is illustrated by the following explicit calculation of $\bar{t}_j$ for the network in figure 3. The network is the same as in figure 2 but has explicit node labels and traffic values. The imaginary links carrying the traffic back from destination to source are also depicted. In the diagram and equations the target argument $j = 4$ is omitted.

$$\Phi = (\phi_{ik}(j))_{i,k} = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & 0 & 0 & 1 \\ \frac{2}{3} & 0 & \frac{1}{3} & 0 \end{pmatrix} \quad \lim_{n\to\infty} \Phi^n = \begin{pmatrix} \frac{6}{25} & \frac{3}{25} & \frac{7}{25} & \frac{9}{25} \\ \frac{6}{25} & \frac{3}{25} & \frac{7}{25} & \frac{9}{25} \\ \frac{6}{25} & \frac{3}{25} & \frac{7}{25} & \frac{9}{25} \\ \frac{6}{25} & \frac{3}{25} & \frac{7}{25} & \frac{9}{25} \end{pmatrix} \quad \Rightarrow \bar{t}' = \frac{1}{25} \begin{pmatrix} 6 \\ 3 \\ 7 \\ 9 \end{pmatrix}^\top$$

$$\Rightarrow \bar{t}' \cdot \Phi = \tfrac{1}{25} \begin{pmatrix} 6 & 3 & 7 & 9 \end{pmatrix} \cdot \Phi = \tfrac{1}{25} \begin{pmatrix} 9\cdot\frac{2}{3} \\ 6\cdot\frac{1}{2} \\ 6\cdot\frac{1}{2}+3\cdot\frac{1}{3}+9\cdot\frac{1}{3} \\ 3\cdot\frac{2}{3}+7\cdot 1 \end{pmatrix}^\top = \bar{t}' \quad \Rightarrow \bar{t} = \boxed{\begin{pmatrix} 100 \\ 50 \\ 116\frac{1}{3} \\ 150 \end{pmatrix}^\top \text{kbit/s}}$$

Notice that $\sum_k \bar{t}'_k = 1$. By requiring $\bar{t}_4 = r_1 + r_3 = 150$ kbit/s the scale factor for vector $\bar{t}'$ can be determined: $\bar{t} = \frac{25}{9} \cdot 150 \cdot \bar{t}'$.

2.2 Conditions for Minimum Delay

Having established a formal model for network traffic flow and a method to represent the total delay experienced by packets, the goal of minimizing delay can be expressed by minimizing D_T. The analysis in this section will establish two conditions for minimizing the total delay.

2.2.1 Marginal Delay

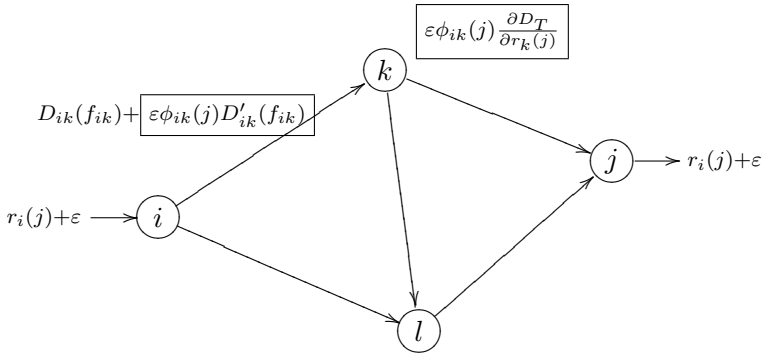

Figure 4: Example network with incremental delays framed

Begin with the following thought experiment. Suppose a given network (like the one in figure 4) is running and the input traffic $r_i(j)$ is increased by a small amount ε. The routing algorithm must determine how the outgoing links of i will carry this extra traffic towards node j and how it will effect the total delay D_T.

The straight-forward method is to choose the link with the smallest incremental delay $D'_{ik}(f_{ik})$. Thus the new traffic introduces the smallest delay in respect to outgoing links of i. However this straight-forward approach is only user optimal: it disregards that the increase may have other effects on existing traffic further down the path and may lead to non-optimal total performance of the network. Therefore the total effect of an increase of traffic on one of the outgoing links needs to be calculated. Analytically this is expressed by the partial derivative $\frac{\partial D_T}{\partial r_i(j)}$. It represents exactly the value needed: the increase of the total delay D_T regarding extra input traffic $r_i(j)$. An equation for the marginal delay can be deduced by letting $\varepsilon \to 0$.

Say the extra traffic is routed over an intermediate node k. Obviously the link (i,k) will then carry $\varepsilon\phi_{ik}(j)$ extra traffic. The delay on link (i,k) previously being $D_{ik}(f_{ik})$ increases by

$$\varepsilon\phi_{ik}(j)D'_{ik}(f_{ik}) \quad \text{where } D'_{ik}(f_{ik}) = \frac{\mathrm{d}D_{ik}(f_{ik})}{\mathrm{d}f_{ik}}$$

Then the additional traffic travels from node k towards node j and further increases the total delay on its path. Regarding node k this extra traffic can be viewed in the same manner as the new incoming traffic at node i: the increase is $\varepsilon\phi_{ik}(j)\frac{\partial D_T}{\partial r_k(j)}$. This leads to an implicit formula for the total delay increase:

$$\frac{\partial D_T}{\partial r_i(j)} = \sum_k \phi_{ik}(j)\left(D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_k(j)}\right) \tag{6}$$

Set $\frac{\partial D_T}{\partial r_i(j)} = 0$ for all $i = j$ or $(i,j) \notin \mathcal{L}$.

Now it's possible to select the best link (i,k), but a routing algorithm adjusts the network flow by changing routing variables in $\boldsymbol{\phi}$. The equations above only examine the change in delay regarding increased traffic. So now consider how changes to a $\phi_{ik}(j)$ will effect the total delay D_T: $\frac{\partial D_T}{\partial \phi_{ik}(j)}$. Figure 5 illustrates a link (i,k) with routing variable $\phi_{ik}(j)$.

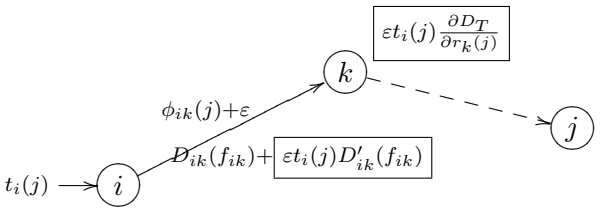

Figure 5: Example network with incremental delays in respect to $\phi_{ik}(j)$

The link (i,k) initially carries $t_i(j)\phi_{ik}(j)$ units of traffic. If $\phi_{ik}(j)$ is increased by a small amount ε, then the traffic on link (i,k) increases by $\varepsilon t_i(j)$ and the delay by $\varepsilon t_i(j)D'_{ik}(f_{ik})$. The additional traffic $\varepsilon t_i(j)$ arriving at node k again increases the total delay of network. Implicitly this can be expressed by

$$\frac{\partial D_T}{\partial \phi_{ik}(j)} = t_i(j)\left(D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_k(j)}\right) \tag{7}$$

Gallager shows that given sets $\boldsymbol{r}$, $\boldsymbol{\phi}$ and marginal delay functions $D'_{ik}(f_{ik})$ the equations above have a unique solutions for $\frac{\partial D_T}{\partial r_i(j)}$ and $\frac{\partial D_T}{\partial \phi_{ik}(j)}$. He constructs two explicit continuous equations depending only on the three given sets. However the implicit equations 6 and 7 are more useful. In the next section, the two equations will be analyzed and transformed into two conditions for minimum total delay.

2.2.2 Necessary and Sufficient Conditions

Having established the partial derivatives of D_T as a function of $\boldsymbol{\phi}$, a minimum can be found by calculating a stationary point in which all first-order derivatives are zero. Furthermore Lagrange multipliers are required, because these stationary points also have to satisfy the constraints $\sum_k \phi_{ik}(j) = 1$ and $\phi_{ik}(j) \geq 0$ $(\forall\, i,j,k)$. This method introduces the multipliers λ_{ij} and gives a necessary condition for finding a minimum in D_T:

$$\frac{\partial D_T}{\partial \phi_{ik}(j)} = t_i(j)\left(D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_k(j)}\right)\begin{cases} = \lambda_{ij}, & \phi_{ik}(j) > 0 \\ \geq \lambda_{ij}, & \phi_{ik}(j) = 0 \end{cases} \quad \forall\, i \neq j\, \forall\, (i,k) \in \mathcal{L} \tag{8}$$

The first observation is that the multipliers λ_{ij} do not depend on k. This leads to the insight, that for all links (i,k) having $\phi_{ik}(j) > 0$ the marginal delay $\frac{\partial D_T}{\partial \phi_{ik}(j)}$ must be the same value λ_{ij}. Furthermore links with $\phi_{ik}(j) = 0$ must have greater marginal delay and thus are less good links for new traffic. However this condition is not sufficient to find a set $\boldsymbol{\phi}$ with minimum delay.

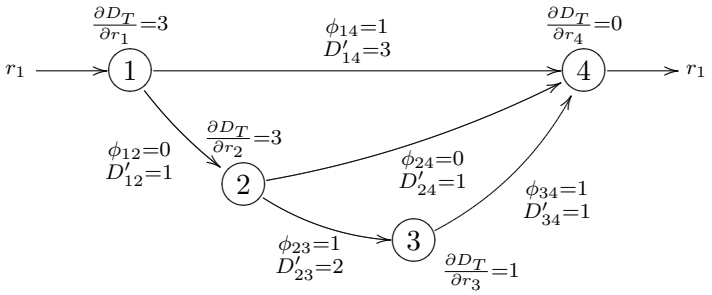

Figure 6: Network satisfying equation 8

Figure 6 shows a network fulfilling equation 8. To improve readability the target argument $j = 4$ is omitted in the diagram and following calculations. In the example the delay functions

$D_{ik}(f_{ik})$ are linear to f_{ik}: $D_{14}(f_{14}) = 3 \cdot f_{14}$ and thus the derivative $D'_{14}(f_{14}) = 3$. The partial derivatives of D_T in respect to ϕ_{ik} can be calculated explicitly using equations 6 and 7:

$$
\begin{aligned}
\frac{\partial D_T}{\partial \phi_{14}} &= t_1 \left(D'_{14} + \frac{\partial D_T}{\partial r_4} \right) = r_1 D'_{14} = \boxed{3r_1} = \lambda_{14} \\
\frac{\partial D_T}{\partial \phi_{12}} &= t_1 \left(D'_{12} + \frac{\partial D_T}{\partial r_2} \right) \\
&= t_1 \left(D'_{12} + \phi_{23} \left[D'_{23} + \frac{\partial D_T}{\partial r_3} \right] + \phi_{24} \left[D'_{24} + \frac{\partial D_T}{\partial r_4} \right] \right) \\
&= t_1 \left(D'_{12} + \phi_{23} \left[D'_{23} + \phi_{34} \left(D'_{34} + \frac{\partial D_T}{\partial r_4} \right) \right] + 0 \left[D'_{24} + \frac{\partial D_T}{\partial r_4} \right] \right) \\
&= r_1 \left(D'_{12} + \phi_{23} D'_{23} + \phi_{23} \phi_{34} D'_{34} \right) = r_1 \left(1 + 1 \cdot 2 + 1 \cdot 1 \cdot 1 \right) = \boxed{4r_1} \geq \lambda_{14} \\
\frac{\partial D_T}{\partial \phi_{24}} &= t_2 \left(D'_{24} + \frac{\partial D_T}{\partial r_4} \right) = r_1 \phi_{12} \left(\ldots \right) = r_1 0 \left(\ldots \right) = \boxed{0} \geq \lambda_{24} \\
&\text{for the same reason: } \frac{\partial D_T}{\partial \phi_{23}} = \boxed{0} = \lambda_{24} \text{ and } \frac{\partial D_T}{\partial \phi_{34}} = \boxed{0} = \lambda_{34}
\end{aligned}
$$

Thus equation 8 is satisfied with $\lambda_{14} = 3r_1$, $\lambda_{24} = 0$ and $\lambda_{34} = 0$. In total the delay $D_T = D_{14}(r_1) = 3r_1$. However by changing the routing variables as shown in figure 7 a smaller total delay D_T can be achieved: $D_T = D_{12}(r_1) + D_{24}(r_1) = 1r_1 + 1r_1 = 2r_2$.

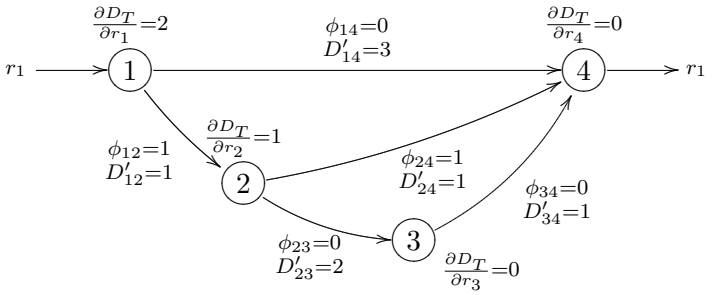

Figure 7: Network with better total delay D_T

Obviously equation 8 is not a sufficient condition for minimal delay. The trouble is that links with $\phi_{ik} = 0$ and therefore $t_i(j) = 0$ automatically fulfill the condition. Gallager approaches this unsatisfactory result with the brilliant idea of removing the factor $t_i(j)$ from equation 8. This leads to the following modified version of the equation, which is a sufficient condition for minimal delay:

$$
\frac{\partial D_T}{\partial r_i(j)} \leq D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_k(j)} \quad \forall\, i \neq j \,\forall\, (i,k) \in \mathcal{L} \tag{9}
$$

The equation depicts an intuitive condition for minimal delay: if the marginal delay increases along the link (in case of $>$), then the total delay on the network can be decreased by sending more traffic over link (i, k).

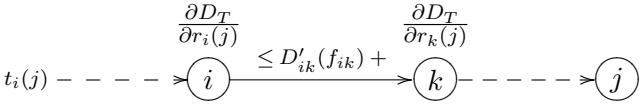

Figure 8: Sufficient condition for the link (i, k)

Gallager proves that equation 9 is a sufficient condition for minimal delay by taking advantage of the convexity of the functions $D_{ik}(f_{ik})$ regarding f_{ik}.

Furthermore with the objective of creating an algorithm based on the sufficient condition, Gallager deduces a more useful inequality: first regard the network as a whole by taking equation 9 times $\phi_{ik}(j)$ and summing over all k:

$$\underbrace{\sum_k \phi_{ik}(j)}_{\overset{(1b)}{=}1} \frac{\partial D_T}{\partial r_i(j)} \leq \sum_k \phi_{ik}(j) \left(D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_k(j)} \right) \overset{(6)}{=} \frac{\partial D_T}{\partial r_i(j)}$$

$$\implies \frac{\partial D_T}{\partial r_i(j)} = D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_k(j)} \tag{10}$$

This shows that the sufficient condition must actually be satisfied with equality. Therefore all outgoing links of i must have the same marginal delay. Now view the new equality 10 from a different angle: for all links (i,k) there is no link (i,m) with larger incremental delay:

$$\forall\,(i,k)\ \neg\exists\,(i,m) \qquad D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_k(j)} < D'_{im}(f_{im}) + \frac{\partial D_T}{\partial r_m(j)}$$

By moving $\neg\exists\,(i,m)$ into the equation and restricting the scope to the link with smallest delay, the following inequality can be deduced. It will be used by the algorithm to iteratively lower total delay.

$$\forall\,(i,k) \quad D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_k(j)} - \min_{(i,m)\in\mathcal{L}} \left(D'_{im}(f_{im}) + \frac{\partial D_T}{\partial r_m(j)} \right) \geq 0 \tag{11}$$

2.3 The Algorithm

The last section established an analytic sufficient condition for achieving minimum total delay routing in a network. This global condition was localized to equation 11. In this section Gallager's distributed minimum delay algorithm is presented on the basis of these considerations.

The general method of the algorithm is to iteratively optimize routing by increasing the routing variables $\phi_{ik}(j)$ for links, which have small marginal delay $D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_k(j)}$ and decrease them in the other case.

The algorithm's approach can be separated into two parts. In the first one the necessary variables $\frac{\partial D_{ik}}{\partial r_i(j)}$ and $D'_{ik}(f_{ik})$ are determined. The second part specifies how to calculate new routing variables $\boldsymbol{\phi}^1$ from the collected variables. Alongside these two calculation steps the algorithm must keep the routing variables loop-free to prevent deadlock.

2.3.1 Variables available to a specific node

One important design goal of Gallager is to calculate optimized variables using a distributed algorithm. Therefore only locally available or collected information may be used in the routing calculations. Figure 9 shows a complex network in which some variables of node i are labelled. In the example all traffic from s to j must flow over i.

Consider in general which variables are directly available to a node i.

- The node always knows its incoming and outgoing links and identifiers for its neighbors: in the example $(k_1, k_2, k_3, k_4, k_5, k_6)$.
- On each of the links the amount of traffic flow can be measured: f_{ik} and f_{ki} for all neighbors k.

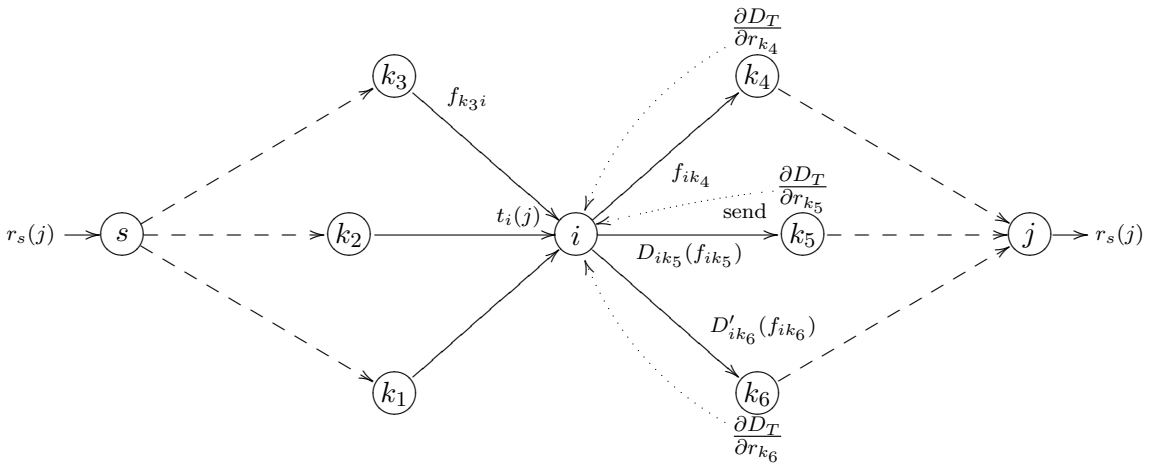

Figure 9: Some variables of node i

- Among the incoming traffic flow the node receives some traffic $t_i(j)$, which must be forwarded towards node j.
- To guide the traffic towards node j each link has a routing variable $\phi_{ik}(j)$.
- Each traffic flow and link introduces a delay into the network. This delay D_{ik} can either be measured directly on the link or be calculated from the traffic flow $D_{ik}(f_{ik})$ using appropriate functions.
- From each link's delay the node can calculate the incremental delay D'_{ik} by differentiating its functions $D_{ik}(f_{ik})$. More often $D'_{ik}(f_{ik})$ will be calculated directly using complex formulas like those found in [Klei70].

Almost all variables required to evaluate equation 11 are directly available to a node. Yet the most interesting variables are still missing: the marginal delay $\frac{\partial D_T}{\partial r_k(j)}$ of its neighbors. These variable must periodically be calculated by the neighbors and sent to i over their link. When node i has received these variables from all its *downstream* neighbors, then i can calculate its own marginal delay using equation 6 and must communicate the result to all its neighbors.

In figure 9 the directions *downstream* and *upstream* are intuitively distinguishable: k_1, k_2, k_3 are upstream and k_4, k_5, k_6 downstream of i. In general Gallager defines k to be *downstream* from i with respect to destination j, if there is a path from i to j through k and all routing variables on the way down to j are positive (i.e. $\phi_{il_1}(j) > 0$ ... $\phi_{l_n j}(j) > 0$). The opposite of downstream is defined as *upstream* (variables on the way up to s are positive).

The following important restriction has to be fulfilled in order to guarantee loop freedom: If k is downstream of i with respect to j, then i must *not* be downstream of k with respect to j (but may be downstream with respect to some other destination node).

By this restriction the downstream relation forms a partial order on the set of nodes. This way computation of marginal delay begins at the destination node and goes upstream towards the source. During this distributed computation a stable state is reached and no deadlock can occur, because a neighbor node cannot be up- and downstream at the same time.

2.3.2 Calculation of new optimized routing variables

The second part of the algorithm calculates new routing variables $\boldsymbol{\phi}^1$ from the variables collected in the last section. The algorithm's actual calculations are based on equation 11. The main challenge in this section is to keep the routing variables loop-free.

To achieve loop freedom, a set of blocked nodes $B_i(j)$ is defined for each node i in direction of j. This set restricts the node flow from node i by requiring $\phi_{ik}(j) = 0 \; \forall k \in B_i(j)$. For notational convenience the blocked set includes all nodes k, which do not have a link to the node i: $\{k : (i,k) \notin \mathcal{L}\} \subseteq B_i(j)$.

The second type of nodes in $B_i(j)$ are neighbors, which have downstream paths containing a loop. Formally $B_i(j)$ includes all nodes k, for which $\phi_{ik}(j) = 0$, $\phi^1_{lm}(j) > 0$ and k can route packets to j over a path that contains some link (l,m) with *improper* $\phi_{lm}(j)$.

A routing variable $\phi_{ik}(j)$ is defined as *improper* if it is positive and the marginal delay from i to j is smaller or equal to the marginal delay from k to j:

$$\phi_{ik}(j) > 0 \quad \text{and} \quad \frac{\partial D_T}{\partial r_i(j)} \leq \frac{\partial D_T}{\partial r_k(j)} \tag{12}$$

Intuitively a routing variable $\phi_{ik}(j)$ is improper if it is inefficient to route more traffic over the link. The algorithm is designed to reduce such $\phi_{ik}(j)$. As the algorithm progresses and optimizes routing, improper routing variables become less likely. The marginal delays $\frac{\partial D_T}{\partial r_i(j)}$ form another partial order on the set of nodes, if and only if there is no improper routing variable in the network. This marginal delay order, if it exists, is the same as the downstream order: marginal delays increase monotonically from destination to source node. These orders can be seen in the examples depicted in figures 6 and 10.

To better understand the blocked set regard the example in figure 10. The partial derivatives of D_T in respect to $r_i(j)$ can be calculated explicitly using equation 6:

$$\frac{\partial D_T}{\partial r_1(4)} = \underbrace{\phi_{12}(4)}_{=1}\left(D'_{12} + \frac{\partial D_T}{\partial r_2(4)}\right) + \underbrace{\boxed{\phi_{13}(4)}}_{=0}\left(D'_{13} + \frac{\partial D_T}{\partial r_3(4)}\right) = 1(2+2) + 0 = 4$$

$$\frac{\partial D_T}{\partial r_2(4)} = \underbrace{\phi_{24}(4)}_{=1}\left(D'_{24} + \frac{\partial D_T}{\partial r_4(4)}\right) = 1(2+0) = 2$$

$$\frac{\partial D_T}{\partial r_3(4)} = \phi_{34}(4)\left(D'_{34} + \frac{\partial D_T}{\partial r_4(4)}\right) + \boxed{\phi_{31}(4)}\left(D'_{31} + \frac{\partial D_T}{\partial r_1(4)}\right) = \tfrac{1}{2}(1+0) + \tfrac{1}{2}(1+4) = 3$$

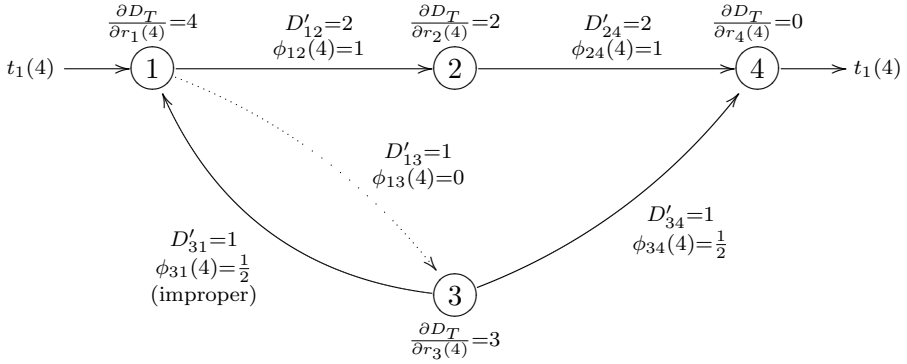

Figure 10: Example of an improper routing variable

From the equations above it can be seen that $\frac{\partial D_T}{\partial r_1(4)}$ and $\frac{\partial D_T}{\partial r_3(4)}$ would depend on each other if both $\phi_{13}(4)$ and $\phi_{31}(4)$ are larger than zero. Furthermore node 1 would simultaneously be up- and downstream to 3, thus violating the condition for loop freedom. The block set is defined to avoid this situation.

In figure 10 the variable $\phi_{13}(4)$ is zero. To keep the network loop-free, $\phi_{13}(4)$ must stay zero as long as $\phi_{31}(4)$ is positive. In this state $\phi_{31}(4)$ is improper, because the marginal delay from

node 3 to 4 is smaller than the marginal delay from 1 to 4. Therefore node 3 will be blocked from node 1 in respect to destination 4.

Note that the set $B_i(j)$ can change in each iteration of the algorithm. This way, nodes which were blocked in the past can become unblocked. Previously unused links can carry traffic after the algorithm breaks up possible loops by reducing improper routing variables to zero.

The algorithm always reduces improper routing values and must keep the routing variables loop-free. Based on equation 11 the actual calculation of the algorithm reduces the amount of traffic on non-optimal links and increases it on the best link.

An iteration of the calculation starts by determining the link (i, b) with the lowest marginal delay $D'_{ib}(f_{ib}) + \frac{\partial D_T}{\partial r_b(j)}$. The node b is the best unblocked downstream node and its routing variable $\phi_{ib}(j)$ will be increased.

Define $a_{ik}(j)$ as the difference between the marginal delay of link (i, k) and the best link (i, b):

$$a_{ik}(j) = D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_k(j)} - \left(D'_{ib}(f_{ib}) + \frac{\partial D_T}{\partial r_b(j)} \right) \tag{13}$$

For the best link $a_{ib}(j)$ will be zero. Calculating $a_{ik}(j)$ only makes sense for links that are not blocked (i.e. $k \notin B_i(j)$).

Now let $\Delta_{ik}(j)$ be the reduction for routing variable $\phi_{ik}(j)$:

$$\Delta_{ik}(j) = \min \left\{ \phi_{ik}(j),\ \frac{\eta}{t_i(j)} a_{ik}(j) \right\} \tag{14}$$

In equation 14 the difference $a_{ik}(j)$ is adjusted by a scale factor η and the traffic flow $t_i(j)$. Together these factors determine how fast the algorithm adapts to changes. The change is inversely proportional to $t_i(j)$, so routing variables on heavily used links are changed slowly. The scale factor η specifies how fast the algorithm changes variables and influences whether the algorithm converges to a stable state with minimum delay. The min construct is only used to ensure that $\phi^1_{ik}(j) \geq 0$ in equation 15.

As required above the new routing variables $\phi^1_{ik}(j)$ and the reduction $\Delta_{ik}(j)$ are defined as zero for all nodes k from the set $B_i(j)$.

For the rest of the nodes k, $\phi^1_{ik}(j)$ is defined as follows:

$$\phi^1_{ik}(j) = \begin{cases} \phi_{ik}(j) - \Delta_{ik}(j) & \text{if } (i,k) \text{ is not the best link} \\ \phi_{ib}(j) + \sum\limits_{\substack{(i,m)\in\mathcal{L} \\ m \neq b}} \Delta_{im}(j) & \text{if } (i,k) \text{ is the best link and therefore } k = b \end{cases} \tag{15}$$

Thus the new routing variables $\boldsymbol{\phi}^1$ are derived from the old ones by "shifting" an amount of traffic from all non-optimal links to the best one. The amount shifted is determined by the difference in marginal delays and two proportional factors.

Gallager finishes his paper by proving through a series of seven lemmas that the algorithm will achieve total minimum delay regardless of the starting routing variable set (see Appendix C in [Gall77]). The lemmas require $D_{ik}(f_{ik})$ to have a positive first derivative, a non-negative second derivative and a capacity C_{ik} with $\lim_{f_{ik} \nearrow C_{ik}} D_{ik}(f_{ik}) = \infty$.

$$\lim_{m \to \infty} D_T(\phi^m) = \min_{\phi} D_T(\phi) \tag{16}$$

During the proof a very small value for η is determined by upper-bounding all $D'_{ik}(f_{ik})$. Using this η equation 16 can be proven with some advanced calculus like the Taylor series and Cauchy-Schwarz's inequality. The algorithm is reduced to a non-increasing sequence in the compact space of routing variables $\boldsymbol{\phi}$, on which D_T is a continuous function.

3 Conclusion

3.1 Goals Achieved

The merit of Gallager's paper is its rigorous mathematical approach to a problem, which is more often taken care of using heuristics. The approach is founded on a well designed mathematical model of a network. This formal model is custom-tailored to describe the minimum total delay problem and leads to two conditions for achieving global optimization. The conditions are based on the marginal delay of links and neighbors.

These two conditions are directly used to derive an iterative, distributed routing algorithm. Extra attention is paid by the algorithm to keeping the routing paths loop-free at all times.

Using the model, equations for the algorithm's operation and some advanced calculus Gallager proves in minute detail, that the algorithm will always progress from an arbitrary start state into a network state with total minimum delay.

3.2 Problems

Unfortunately the algorithm, as it is presented by Gallager, is unsuitable for real networks or the Internet for several reasons. Due to the mathematical approach some issues of real networks had to be disregarded.

The first drawback is the required scale parameter η. This variable depends on the network and the input traffic, and therefore it is not possible to determine one η for all networks especially if input traffic changes greatly. For small η the algorithm will converge very slowly, but if η is too large it may never converge. In the formal proof a very small value can be used, but this value would lead to unsatisfactory results in real networks. A good parameter has to be gained experimentally.

The question of how the start state can be determined was left open. The algorithm expects a set of loop-free routing variables $\boldsymbol{\phi}$ to begin with. Gallager mentions one possibility: to determine shortest paths using a Dijkstra network flood. This method may lead to link flows which exceed capacity. Hence a good flow control has to be implemented, which limits the input traffic of the network during this initial period.

Another practical issue is not dealt with by the algorithm: when links or nodes are dropped or added, some higher level protocol must handle these network topology changes. The algorithm is able to adapt if one of the currently selected links fail. But when a node goes down, a complete routing update has to be started. A node i which noticed a broken link has to broadcast an infinite marginal delay with respect of all nodes j formerly reachable.

Lastly it remains to be investigated, how fast the algorithm adapts to changing statistics. This is crucial for practical implementation as network input traffic is a constantly changing value.

3.3 Impact

The strength of Gallager's investigations does not lie in practical realization: presented in 1977, there is no known implementation. Recently Gallager's mathematical results have been taken up and refined to an approximate, practically implementable algorithm in [VuGLA99]. The strong model and sound mathematical approach have put analysis of routing algorithms onto a new basis.

References

[Behr00] E. Behrends. *Introduction to Markov chains*. Vieweg. 2000.

[Gall77] R. G. Gallager. A Minimum Delay Routing Algorithm Using Distributed Computation. *IEEE Transactions on Communications* COM-25(1), January 1977, pages 73–85.

[HMMW78] F. Heart, A. McKenzie, J. McQuillian, and D. Walden. *ARPANET Completion Report*. January 1978. Map scanned by Larry Press and available at http://som.csudh.edu/cis/lpress/history/arpamaps/.

[Klei70] L. Kleinrock. Analytic and Simulation Methods in Computer Network Design. *Spring Joint Computer Conference, AFIPS Conference Proceedings* volume 36, May 1970, pages 569–579.

[KuRo03] J. Kurose and K. Ross. *Computer Networking*. Addison-Wesley. 2nd edition, 2003.

[Kuro06] J. Kurose. 10 Networking Papers: Recommended Reading. *ACM SIGCOMM Computer Communication Review* 36(1), 2006, pages 51–52.

[McFR78] J. McQuillan, G. Falk, and I. Richer. A Review of the Development and Performance of the ARPANET Routing Algorithm. *IEEE Transactions on Communications* COM-26(12), December 1978, pages 1802–1811.

[Sega77] A. Segall. The Modeling of Adaptive Routing in Data-Communication Networks. *IEEE Transactions on Communications* COM-25(1), January 1977, pages 85–95.

[VuGLA99] S. Vutukury and J. J. Garcia-Luna-Aceves. A Simple Approximation to Minimum-Delay Routing. *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, 1999, pages 227–238.