CS 204 Lecture Notes on Elementary Network Analysis

Mart Molle
Department of Computer Science and Engineering
University of California, Riverside CA 92521
mart@cs.ucr.edu

October 18, 2006

1 First-Order "Fluid Flow" Models

1.1 Graph Theoretic Representation of a Network

Starting from the basic concepts from graph theory, we can create simple network models from which we can derive some fundamental properties of the system without having to work too hard. nor make too many restrictive assumptions.

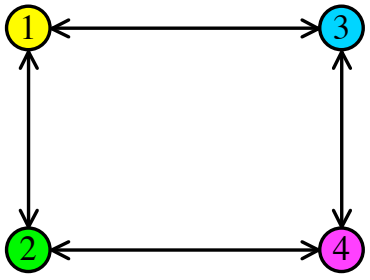
1.2 Network Topolgy

Consider a graph $G(N, E)$ with N nodes and E edges. Each node represents an external *network user* or an internal *packet switch* that is part of the network infrastructure. Similearly, each edge represents a low-level *communications link* that connects its source node to its destination node.

The network structure is represented by a (weighted) $N \times N$ adjacency matrix called the *topology matrix*, $\mathbf{C}$, whose (i, j)th entry $c_{i,j}$ is either zero, if there is no link from node n_i to node n_j, or the normalized capacity of that link, otherwise. The choice of units is arbitrary, as long as we are consistent: we could use bits/sec, packets/sec or something else.

1.3 Traffic Demand

From the perspective of the users, the internal structure of the network is not terribly important. Users rely on the network to transport their data from its source to the corresponding destination. However, the particular path through the subnet that is used for each end-to-end flow is irrelevant — as long as the quality of service provided by the network is adequate. Futhermore, the network service provider(s) may decide to change these paths at any time (without informing the affected users) in response to equipment failures, transient traffic congestion, or other factors.



Topology Matrix: C

	1	2	3	4
1	**0**	*10*	*10*	*0*
2	*10*	**0**	*0*	*10*
3	*10*	*0*	**0**	*10*
4	*0*	*10*	*10*	**0**

Figure 1: Sample 4-node network and its associated topology matrix

Traffic Demand Matrix: $\mathbf{\Gamma}$

	1	2	3	4
1	**0**	*1*	*2*	*2*
2	*2*	**0**	*3*	*7*
3	*1*	*1*	**0**	*2*
4	*4*	*3*	*1*	**0**
sum:	*7*	*5*	*6*	*11*

Figure 2: $N \times N$ traffic demand matrix, $\mathbf{\Gamma}$. Row i shows traffic generate rate source node i to every other destination node. At the bottom of column j shows the total traffic generated by all sources for destination node j.

For these reasons, the user traffic demand is described relative to an $N \times N$ *fully connected virtual overlay network*, $\mathbf{\Gamma}$, whose (i,k)th entry, $\gamma_{i,k}$ represents the normalized rate at which source node n_i generates traffic that must be delivered by the network to destination node n_k. Note that all diagonal entries must be zero, since nodes don't use the network to deliver traffic to themselves! However, every other entry $\gamma_{i,k}$ could be non-zero — even if the corresponding entry in the topology matrix $c_{i,k}$ is zero — because $\mathbf{\Gamma}$ is calculated as if all traffic follows one-hop paths through the virtual overlay network.

1.4 Transforming the Model into a Line Graph Representation

Let us now shift the focus of our discussion to the attributes of the communication links rather than the switches. For simplicity, let's assume unless otherwise stated that we have a wired network. The broadcast delivery property of wireless channels makes it messy to determine which nodes are directly connected to a given link. In addition, let's assume that all links are unidirectional and point-to-point. Even though most wired network links are bi-directional, some network architectures are specifically designed to use unidirectional links. Moreover, even if the links are bi-directional, the traffic on the link is usually quite different between the two directions. For notational clarity, I will follow the convention where each node is labelled by a distinct number (i.e., $n_i \in \{1, 2, \dots\} \equiv N$), and each edge is labelled by a distinct letter (i.e., $\ell_j \in \{a, b, \dots\} \equiv E$). We also define the following conversion functions between these two labelling conventions so that, if nodes 1 and 2 are connected by edge a, then we can say $a \equiv edge(1,2)$, $1 \equiv head(a)$ and $2 \equiv tail(a)$.

To accomodate this change of focus, we will now transform the graph G, which represents the network as a collection of switches, into its associated *line graph* G^*, which represents the network as a collection of communication links, by applying the following simple rules:

- every edge in G is a node in G^*, and
- two nodes in G^* are connected by an edge if and only if the corresponding edges in G meet at a common vertex.
- More generally, if G is a directed graph, then so is G^*, and there is a directed edge from node ℓ_i to node ℓ_j in G^* if and only if $tail(\ell_i) \equiv head(\ell_j)$ in G.

This line graph network representation offers several advantages over our original node-based model. First, link bandwidth is the most critical resource in a network because it is constrained by the available physical-layer technology and/or the subject of recurring monthly charges from an outside service provider. Conversely *switching* (i.e., deciding where next to send each arriving packet based on its input port and the information carried in its header) is essentially just a highly parallelizable table lookup problem that can be solved by a one-time hardware purchase. Therefore, it is probably best to visualize the operation of the

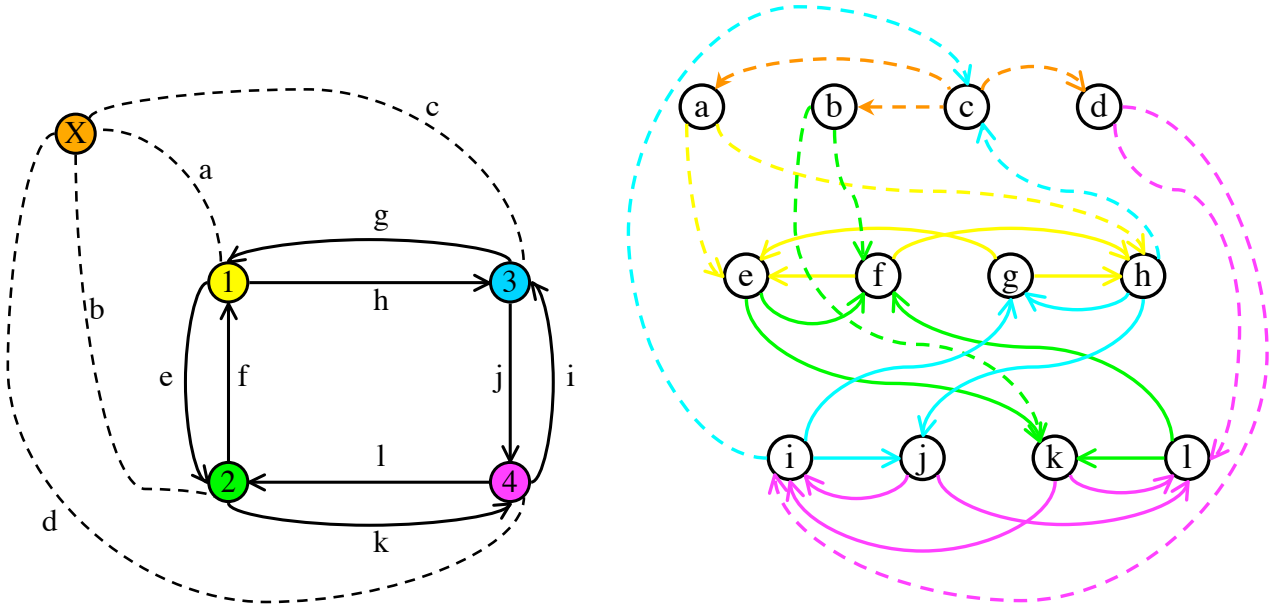


Figure 3: The simple network of Fig. 1 expanded into a directed graph representation shown at left. External node X has been added to serve as the source and sink of all end-to-end traffic; its link directions vary as a function of the destintion. The corresponding line graph is shown at right, assuming node 3 is the destination.

network as packets visiting a *sequence of communication links*, as opposed to visiting a *sequence of switching nodes*. While a packet is associated with a particular link ℓ, it may be:

- waiting in the corresponding output queue at node $head(\ell)$ until link ℓ is available,
- being transmitted over link ℓ at the rate of c_ℓ^* bits per unit time, or
- traveling at the propogation velocity of the channel from node $head(\ell)$ to node $tail(\ell)$.

Second, some of the important features in commonly-used routing algorithms can only be expressed by a switching function that depends on the packet's input port. For example, suppose a packet addressed to destination node n arrives on link ℓ_i of a standard layer-2 transparent bridge. If n is an unknown address, then the bridge must broadcast the packet to *all ports except* ℓ_i. Otherwise, the switch first finds the output port ℓ_j for address n in its forwarding database, and then sends one copy of the packet *only to port* ℓ_j — *unless* $\ell_i \equiv \ell_j$, in which case it must *discard the packet*. VLAN-capable bridges go even further, by partitioning the switching function to create multiple disjoint logical layer-2 networks on the basis of port numbers, protocol types, or explicit VLAN tags. Finally, many IP routers support very general control policies (using a set of pattern-matching rules called an *access-control list*, or ACL) to control the types of traffic (based on pattern-matching of IP addresses, protocols, port numbers, etc) that are permitted to enter or leave the router through each interface.

Third, the line graph representation makes it easy to apply the (vast) research literature about queueing network models and/or commputer systems performance analysis. In this case, complex systems are modelled as a collection of resources (or service centers), together with a set of tasks (or customers) who visit a sequence of resources to obtain service. In general, each resource corresponds to some individaul active processing element, such as a CPU or disk drive, and there is a directed edge connecting one resource to another if tasks leaving the first resource may proceed directly to the second one. It is assumed that each resource has a limited capacity to process tasks, whereas the time for a task to move from one resource to another is negligible. This is exactly the opposite of what we need for analyzing networks if we try to set up a model where resources represent the packet switches, but it fits perfectly with the line graph representation.

Label Conversion Function:

link	head	tail
a	X	1
b	X	2
c	X	3
d	X	4
e	1	2
f	2	1
g	3	1
h	1	3
i	4	3
j	3	4
k	2	4
l	4	2

Routing Tables at each Node (only info for destination 3 shown)
upper-left element not used; arrivals from destination node are bogus

Node 1:

	a	e	h
a	?	0.6	0.4
f	0	0	1
g	**0**	**0**	**0**

Node 2:

	b	f	k
b	?	1	0
e	0	0	1
l	0	1	0

Node 3:

	c	g	j
c	?	**0**	**0**
h	1	0	0
i	1	0	0

Node 4:

	d	i	l
d	?	1	0
j	**0**	**0**	**0**
k	0	1	0

Figure 4: Local routing tables for each node of the sample four-node network using the line graph representation. For compactness, only the layer corresponding to destination node 3 is shown.

Global Routing Table R (only info for destination 3 shown)
upper left quadrant represents routing table at node X

Also represents Adjacency Matrix for Line Graph

	a	b	c	d	e	f	g	h	i	j	k	l
a					0.6			0.4				
b						1					0	
c	0.33333	0.5	0	0.16667			0			0		
d									1			0
e		0				0					1	
f	0				0			1				
g	0				0			0				
h			1				0			0		
i			1				0			0		
j				0					0			0
k				0					1			0
l		0				1					0	

Figure 5: Global routing matrix $\mathbf{R}$ for the sample four-node network using the line graph representation. For compactness, only the layer corresponding to destination node 3 is shown.

1.5 Routing Matrix

In order to satisfy the user traffic generated by its customers, network service providers must map that traffic demand, $\mathbf{\Gamma}$, onto the physical network topology. Therefore, the network administrator must generate (either manually, or by enabling some automated route-update protocol) a *local routing table*, $\mathbf{R}^{(n)}$, for each node n in the physical network graph. This local routing table enables node n to select an appropriate "next hop" towards the final destination for each traffic flow. If we focus on the line-graph representation of the network, then $\mathbf{R}^{(n)}$ has one row per input link, one column per output link, and one layer per destination, as shown in Fig. 4. The (i, j, k)th entry of this three-dimensional matrix, $r_{i,j,k}$, gives the fraction of traffic arriving on link ℓ_i should be forwarded to link ℓ_j if its final destination is node k. Combining the local routing tables for each node gives us the global routing matrix $\mathbf{R}$ for the network, as shown in Fig. 5.

We say that the routing matrix $\mathbf{R}$ supports *sink tree routing* if for every destination k, $\mathbf{R}$ forces all traffic to follow a connected subtree of "inbound" edges directed toward node k at its root. Distance-vector IP routing algorithms (such as RIP) as well as the well-known transparent bridging algorithm used by local area network switches fit within this category. Bridges use the same tree for all traffic (up to the direction of each edge), whereas RIP in general will use different sink trees for each destination. Since all traffic that enters node n on its way to destination node k must follow the *same fixed path* from n to k, we see that:

- all rows of $\mathbf{R}^{(n)}$ must be identical, since the routing cannot depend on a packet's starting node nor

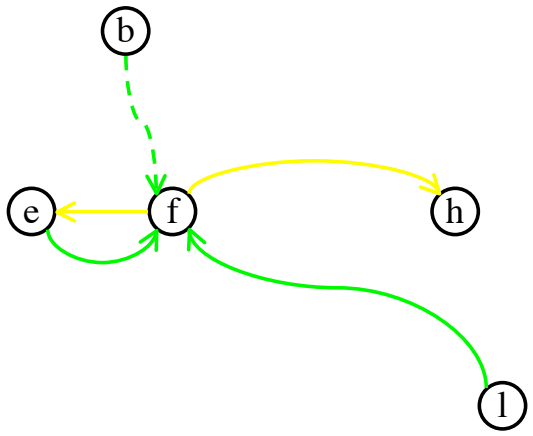


Figure 6: Graphical representation of the input/output terms that affect flow conservation at link f for traffic going to destination node 3.

through which incoming link it entered node n; and

- each row of $\mathbf{R}^{(n)}$ must contain the value one in a single column and zeros everywhere else, or multiple paths to node k will be created.

Among all the local routing tables shown in Fig. 4, only $\mathbf{R}^{(3)}$ has the required form to satisfy sink tree routing. However, one could argue that $\mathbf{R}^{(4)}$ should also be counted, since the only discrepancy to discard any packets that would otherwise be trapped in a non-sensical "ping-pong" loop between links i and j. Conversely, $\mathbf{R}^{(2)}$ clearly violates the sink tree rule because it sends all traffic that arrived on link e to one outgoing link and all other traffic to a different outgoing link.

We say that the routing matrix $\mathbf{R}$ supports *bifurcated routing* if some nodes, say n use multiple outgoing links to forward traffic to a single destination, say node k. For example, $\mathbf{R}^{(1)}$ in Fig. 4 shows that node 1 distributes locally-generated traffic going to destination 3 between its two available outgoing links. In such cases, we assume that node n makes a series of local decisions on how to distribute traffic for destination k between the available paths. Link-state IP routing algorithms (such as OSPF) fit within this category.

Clearly, line graph-based routing tables give us considerable flexibility in representing routing policies, allowing us to include the packet's current location n, its final destination k, and optionally the input link by which it reach n as factors in the local routing decision at node n. Although the routing decision could be further generalized to include the packet sender as another factor, at some point we exceed the "memory limit" of this approach. For example, a *completely memoryless* routing table that does not consider how a packet reached node n cannot prevent the "ping-pong" loop from link j to link i without blocking all packet deliverys to node 3. Line graph-based routing tables can prevent such 2-hop loops, but cannot detect longer cycles. For this reason, real networks add external mechanisms to avoid packet looping problems, such as carrying a hop-count (or "time-to-live") field in each packet header.

1.6 Calculating the Link Flows

Let $\mathbf{\Lambda}$ be the $E \times N$ matrix of link flows, whose (k, ℓ)th element $\lambda_{k,\ell}$ is the rate at which traffic for destination k traverses link ℓ. Since the flow calculation problem must be solved separately for each destination (since packets intended for one destination are not interchangeable with packets intended for another), let us also define the E-element row vector $\vec{\lambda}^{(k)} \equiv \{\lambda_{k,\ell_1}, \ldots, \lambda_{k,\ell_E}\}$ to simply the notation in the formulas given below.

Given the end-to-end traffic demand matrix, $\mathbf{\Gamma}$, and global routing matrix, $\mathbf{R}$, we can find the link flows by solving a set of simultaneous linear equations that enforce *conservation of flow* at every point in the network.[1] All the necessary information for deriving these flow conservation equations is contained in the

[1] This technique is widely used in circuit analysis, where it is called Kirchoff's First Law of conservation of the electric charge

$R^T - I$	a	b	c	d	e	f	g	h	i	j	k	l	flow balance constraint	Solution vector $\vec{\lambda}$
a	-1	0	0.33333	0	0	0	0	0	0	0	0	0	**0**	**2.0**
b	0	-1	0.5	0	0	0	0	0	0	0	0	0	**0**	**3.0**
c	0	0	-1	0	0	0	0	1	1	0	0	0	**0**	**6.0**
d	0	0	0.16667	-1	0	0	0	0	0	0	0	0	**0**	**1.0**
e	0.6	0	0	0	-1	0	0	0	0	0	0	0	**0**	**1.2**
f	0	1	0	0	0	-1	0	0	0	0	0	1	**0**	**3.0**
g	0	0	0	0	0	0	-1	0	0	0	0	0	**0**	**0.0**
h	0.4	0	0	0	0	1	0	-1	0	0	0	0	**0**	**3.8**
i	0	0	0	1	0	0	0	0	-1	0	1	0	**0**	**2.2**
j	0	0	0	0	0	0	0	0	0	-1	0	0	**0**	**0.0**
k	0	0	0	0	1	0	0	0	0	0	-1	0	**0**	**1.2**
l	0	0	0	0	0	0	0	0	0	0	0	-1	**0**	**0.0**
normalize condition	0	0	1	0	0	0	0	0	0	0	0	0	6	
													$= \sum \gamma_{*,3}$	

Figure 7: Matrix solution to link flows in the sample four-node network. For compactness, only the layer corresponding to destination node 3 is shown.

line graph representation of the network (e.g., Fig. 3) and the associated routing probabilities (e.g., Fig. 5).

In general, for any link $\ell \in E$ (i.e., a node in G^*) and any destination node k, the flow rate of destination-k traffic in and out of link ℓ must be balanced. For example, Fig. 6 shows the subgraph of direct inputs and outputs to link f. Notice that $head(f) = 2$ and $tail(f) = 1$ so all input links connect through node 2, and all outputs connect through node 1. Also, since neither endpoint of link f connects to the destination node, the directions of both links b and a are inbound from X and hence b is the only externl link that affects this flow balance condition.

Let us now consider what happens to the packets when they arrive at switch $tail(\ell)$. The hardware design of a packet switch can use either *input queueing* or *output queueing*. In an input queueing design, the packets *first wait* at their input port until their respective output link is available and *then move* to their respective output port. Conversely, in an output queueing design the packets *first move* to their respective output ports as quickly as possible, and *then wait* until the link is available. Output queueing is the overwhelming design choice because it eliminates a performance bottleneck called "head of the line" blocking — where one packet waiting for a busy output port can delay many packets behind it in the queue, even if their respective output ports are idle. Thus, if we assume the packet switches use output queueing, then all traffic arriving at node $tail(\ell)$ is immediately removed from link ℓ and either moved to the appropriate output queue or discarded because it violates some policy decision. Either way, the total output rate of destination-k traffic from the link is clearly $\lambda_{k,\ell}$. Thus:

$$\sum_{\substack{i \in E, \\ tail(i) \equiv head(\ell)}} \lambda_{k,i} \cdot r_{i,\ell,k} \;=\; \lambda_{k,\ell} \tag{1}$$

All of these flow-conservation equations can be combined to form a single matrix equation covering all destination-k traffic:

$$\vec{\lambda}^{(k)} \cdot \mathbf{R} = \vec{\lambda}^{(k)} \cdot \mathbf{I} \tag{2}$$

where $\mathbf{I}$ is the $E \times E$ identity matrix. It is easy to solve such systems of equations using Gaussian elimination, but first we need to put it into standard form $\mathbf{A}\vec{x} = \vec{b}$, where A is a square matrix of coefficients, $\vec{x}$ is the column vector of unknowns, and $\vec{b}$ is the column vector of right-hand sides. After some simple algebraic manipulations, Eq. 2 may be rewritten into the required form:

$$[\mathbf{R} \;-\; \mathbf{I}]^T \cdot \vec{\lambda}^{(k)^T} = \vec{0} \tag{3}$$

where $\vec{0}$ is the all-zero column vector. At this point, we are almost finished. Unfortunately, the system of equations shown here is *linearly dependent* because we can solve for any single flow given the others via

entering and leaving any junction point.

conservation of flow, and indeed a careful look at Eqs. 2 and 3 shows that $\vec{\lambda}^{(k)^T} \equiv \vec{0}$ is one of many possible solutions. Thus, the final step is to drop one of rows in Eq. 3 and replace it by the normalizing equation defined by the $\mathbf{\Gamma}$ matrix. In this case, the flow from destination node k to external node X must be equal to the kth column sum of $\mathbf{\Gamma}$.

Figure 7 shows the result of applying this solution methodology to the destination-3 traffic in our 4-node sample network.[2] The normalizing equation forms an extra row below the terms as defined by Eq. 3.

1.7 Average Path Length

We define the average path length for traffic carried by a network to be the ratio of the total distance covered (in hops) in delivering all traffic divided by the total amount of traffic delivered. Similar definitions can be made for traffic going to a specific destination node, and for traffic going from a specific source to a specific destination.

The denominator for this ratio (in the form of a rate) can be found from the end-to-end traffic demand matrix, $\mathbf{\Gamma}$, which includes all traffic but assumes one-hop delivery through some virtual overlay network. Similarly, the numerator for this ratio can be found from the link flow matrix, $\mathbf{\Lambda}$, which also includes all traffic but counts it towards the flow of every link traversed.

Thus, the global path length for all traffic carried by the network is simply Λ/Γ, the sum of all entries in the link flow matrix divided by the sum of all entries in the end-to-end demand matrix. Similarly, we can find the average path length for destination-k traffic by summing the entries in flow vector $\vec{\lambda}^{(k)}$ and dividing by the kth column sum from $\mathbf{\Gamma}$.

[2]Since MS Excel has a built-in function for carrying out matrix inversion but not for Gaussian elimination, the final solution shown as the right-most column vector in the Fig. 7 was computed through the seeminly equivalent, but numerically less-stable, formula $\vec{x}^T = \mathbf{A}^{-1}\vec{b}^T$.