Report Number: WUCS-97-37

1997-01-01

Symmetrical Routes and Reverse Path Congestion Control

Rajib Ghosh and George Varghese

We describe new mechanisms to deal with asymmetries that arise in routing protocols. We show how to avoid route asymmetries (due to non-unique shortest paths) by adding random integer link costs. We show in detail how RIP can be modified to avoid route asymmetry with high probability, without affecting either its efficiency or performance metrics such as convergence time. Symmetrical intra-domain routing also makes possible a new form of congestion control that we call Reverse Path Congestion Control (RPCC). We show, using simulations, that RPCC can augment existing TCP congestion control mechanisms to improve start up behavior and to... **Read complete abstract on page 2.**

Symmetrical Routes and Reverse Path Congestion Control

Rajib Ghosh and George Varghese

Complete Abstract:

We describe new mechanisms to deal with asymmetries that arise in routing protocols. We show how to avoid route asymmetries (due to non-unique shortest paths) by adding random integer link costs. We show in detail how RIP can be modified to avoid route asymmetry with high probability, without affecting either its efficiency or performance metrics such as convergence time. Symmetrical intra-domain routing also makes possible a new form of congestion control that we call Reverse Path Congestion Control (RPCC). We show, using simulations, that RPCC can augment existing TCP congestion control mechanisms to improve start up behavior and to avoid losses at the boundary between domains and the backbone.

Symmetrical Routes and Reverse Path Congestion Control

Rajib Ghosh
George Varghese

WUCS-97-37

September 11, 1997

Department of Computer Science
Campus Box 1045
Washington University
One Brookings Drive
St. Louis, MO 63130-4899

Abstract

We describe new mechanisms to deal with asymmetries that arise in routing protocols. We show how to avoid route asymmetries (due to non-unique shortest paths) by adding random integer link costs. We show in detail how RIP can be modified to avoid route asymmetry with high probability, without affecting either its efficiency or performance metrics such as convergence time. Symmetrical intra-domain routing also makes possible a new form of congestion control that we call Reverse Path Congestion Control (RPCC). We show, using simulations, that RPCC can augment existing TCP congestion control mechanisms to improve start up behavior and to avoid losses at the boundary between domains and the backbone.

Keywords: Internet, Autonomous System, Route Asymmetry, Congestion Control, TCP, Bellman-Ford Algorithm, Distance-Vector, Link-State, RIP.

This work was supported by National Science Foundation under grant NCR-9628145 and the ONR Young Investigator Award.

Symmetrical Routes and Reverse Path Congestion Control

Rajib Ghosh
rajib@ccrc.wustl.edu
+1 (314) 935-4163

George Varghese
varghese@askew.wustl.edu
+1 (314) 935-4963

Abstract

We describe new mechanisms to deal with asymmetries that arise in routing protocols. We show how to avoid route asymmetries (due to non-unique shortest paths) by adding random integer link costs. We show in detail how RIP can be modified to avoid route asymmetry with high probability, without affecting either its efficiency or performance metrics such as convergence time. Symmetrical intra-domain routing also makes possible a new form of congestion control that we call Reverse Path Congestion Control (RPCC). We show, using simulations, that RPCC can augment existing TCP congestion control mechanisms to improve start up behavior and to avoid losses at the boundary between domains and the backbone.

Keywords: Internet, Autonomous System, Route Asymmetry, Congestion Control, TCP, Bellman-Ford Algorithm, Distance-Vector, Link-State, RIP.

1. Introduction

Computing routes between source and destination end-nodes is a fundamental task in any large computer network. This paper deals with the issue of possible *asymmetries* in routes. We describe possible problems caused by asymmetry, and mechanisms to deal with these problems. For concreteness, we will ground our discussion in the context of the Internet, using specific protocols such as RIP and TCP as examples. However, our ideas apply to other datagram routing protocols and even to route calculation in virtual circuit networks.

Symmetry is a great simplifying device in mathematics and physics because it reduces the number of cases to be considered. In some cases, a lack of symmetry came as a surprise, as in the case of Hamilton's discovery of a non-commutative algebra [Bel86], and Lee and Yang's discovery [Fer91] that parity is not always conserved. In a similar fashion, symmetry in routes and link costs is generally assumed to hold. However, Paxson's classic study of Internet routing [Pax96a, Pax96b] reveals that approximately half of the measured routes include asymmetric paths that visit at least one different city. Thus, a study of routing asymmetries appears to be timely.

The two most natural symmetry assumptions in routing are *symmetrical link costs* and *symmetrical routes.* It is common to assume that the cost of a link from node A to neighboring node B is the same as the cost of the link from node B to node A. Similarly, if A and B are not neighbors, and the route from A to B is $A, R_1, \ldots, R_n, B$, it is natural to assume that the route from B to A is the reverse route $B, R_n, \ldots, R_1, A$.

Neither of these assumptions is necessarily true. While managers can always enforce symmetrical link costs, such enforcement may not lead to optimal routes when the link costs are asymmetrical. Many links consist of two simplex links in each direction, with each simplex link having potentially different characteristics, such as bandwidth and latency. A common example is cable links to the home in which the bandwidth to the home is large but the bandwidth from the home is little or non-existent. Even if link costs were symmetrical, most routing protocols make no effort to compute symmetrical routes: if there are multiple shortest path routes, arbitrary tiebreakers can lead to asymmetrical routes. In some cases, routing protocols deliberately compute asymmetrical routes because of policy constraints [Pax96a].

Most standard routing protocols (i.e., Distance-Vector, Link-State) deal well with asymmetrical link costs for computing routes between a single source and a single destination. However many *Multicast* routing algorithms tacitly assume link asymmetry. A lack of link symmetry can lead to the calculation of suboptimal multicast trees. This can be avoided by modifying existing routing algorithms to calculate multicast "From Trees"[1] in addition to "To Trees".

Clearly link asymmetry leads to route asymmetry. In Section 2 we assume link symmetry, and isolate the impact of routing protocols on route asymmetry. We start by describing why symmetrical routes are desirable, and what are the major causes of route asymmetry. We then describe a new technique for avoiding asymmetries due to non-unique shortest paths. The main idea is to add an additional random cost component to each link so that, with high probability, there is only one shortest path between every source and destination. We validate our scheme by a theoretical analysis (Section 2.4) and by simulations on real Internet domains that have asymmetrical routes (Section 2.5). In Section 3, we discuss possible modifications to a common Interior Gateway Protocol, RIP, to provide symmetrical routes within a domain.

In Section 4, we assume symmetrical intra-domain routing and consider possible improvements to congestion control mechanisms based on this assumption. We propose *Reverse Path Congestion Control* (RPCC), a new bit-based scheme for passing congestion information from the network to source transport protocols. Unlike the classic DECbit [RJ88] scheme, RPCC does not require an end-to-end path of routers that can pass a congestion bit. Instead, congestion bits are passed directly from the congested router on reverse traffic flowing to the source. This allows faster response, and only requires that all routers from the source to the congested node be able to pass the congestion bit. Such a bit is useful if the source domain implements symmetrical routes, and as is the case today, congestion is common on the boundary between domains and backbones. We demonstrate potential benefits of RPCC by simulations in which we augment existing TCP congestion control mechanisms to provide faster reaction to impending congestion.

Our two ideas of Random Tiebreakers and RPCC are orthogonal. We conclude in Section 5 by discussing the independence of these ideas, and their application to other routing protocols.

2. Route Asymmetry

We begin this section by discussing why route symmetry is desirable, and the impact of asymmetry on routing protocols and measurements. We then analyze the two main causes for route asymmetry: non-unique shortest paths and policy routing. Finally, we propose a technique for creating unique shortest paths by modifying route computation algorithms.

[1] This idea was invented by Steve Deering and Christian Huitema

2.1. Importance of Routing Symmetry

Our discussion of the importance of route symmetry is entirely borrowed from Paxson's Ph.D. thesis [Pax96a]. While route asymmetry is not as pernicious as some of the other pathologies described in [Pax96a] (e.g., routing loops, route fluttering), it does affect several protocols.

For example, the Network Time Protocol, NTP [Mil85], approximates the one-way propagation time as half of the round-trip time between two hosts when synchronizing clocks between widely separated hosts. If the routes are asymmetric, this assumption breaks down. In such a case the two hosts can keep consistent time internally but not between each other.

A second example cited in [Pax96a] is protocols by which connection end-points infer network conditions from the pattern of packet arrivals they observe (e.g., by timing the arrival of acknowledgments [Kes91]). If routing is symmetric the bandwidth observed in the arrival of acks is the same as the bandwidth of the outgoing link. This could allow servers to determine the link bandwidth available for replying to client requests. If routing is not symmetric then the server cannot determine the correct value. Thus routing symmetry is helpful for servers to infer network conditions.

Routing symmetry is also necessary if routers are to set up *anticipatory flow state* for replies to requests which pass through them. For example, when A sends a connection request to B through router R, then R might set up flow state for the reply from B to A. However if the route is not symmetric and the reply path does not contain R, then the anticipatory flow to A is wasted.

In the next section, we describe a new congestion control scheme, *Reverse Path Congestion Control* (Figure 6), which depends completely on symmetrical routes, (at least symmetrical intra-domain routing, see Section 3). RPCC is a congestion avoidance [Jac88] scheme in which a router R, when it senses congestion building up on an outgoing link, sends feedback to all senders who are transmitting on that outgoing link by setting a bit(*congestion bit*) on packets/acknowledgments arriving through that congested link. If the reverse route does not contain this router R, then R will not be able to notify the senders of building congestion. Thus route symmetry is essential for RPCC.

2.2. Sources of Routing Asymmetry

The two most important causes of routing asymmetry are the absence of non-unique shortest paths and Policy Routing. Routers use Bellman-Ford or Link State routing [Per92] algorithms to calculate their routing tables. When multiple routes have the same cost, each router picks a route using an arbitrary tiebreaker. This leads to asymmetry in routes between two nodes A and B if the router closest to A and the router closest to B pick different routes. We describe an example below. We show how we can avoid this asymmetry by modifying route selection algorithms to include a random integer link cost which serves to arbitrate in case of ties.

Internet today consists of *domains* inter-connected by ISPs (*Internet Service Providers*). These ISPs are commercial organizations which charge for the service they provide. This structure of the Internet leads to two important classes of protocols: intra-domain, those that are used within a domain, e.g. RIP [Hed88], and inter-domain, like BGP [RL95] which are used only within backbone routers.

While the lack of a unique shortest path is probably an important source of route asymmetry within domains, the principal source of asymmetries in backbone routers is policy routing. One such example, well described in [Pax96a], is *Hot Potato* and *Cold Potato* routing. This is of growing importance with the growth of several private Internet Service Providers (ISPs). Suppose host A in ISP$_A$ wants to send a packet to host B in ISP$_B$ which is, say, at the other end of the US, and both ISP$_A$ and ISP$_B$ provide connectivity across the US. Then ISP$_A$ might like to "drop" the packet to ISP$_B$ as soon as possible because it might like ISP$_B$ to carry the packet along the costly trans-country link. This is "Hot Potato" routing. On the other hand, ISP$_A$ might like to "keep" the packet to itself as long as possible; this is "Cold Potato" routing. Such routing policies lead to grossly asymmetric routes across countries and continents.

A third cause of routing asymmetry is *Adaptive Routing* in which a router shifts traffic from a highly loaded link to a less loaded one, or load balances across multiple paths. Misconfiguration in routers can also cause route asymmetries.

2.3. Avoiding Routing Asymmetry

The only way to prevent asymmetries due to Policy Routes is to have policies that ensure symmetrical routes based on some agreement between ISPs. Thus, it is really implausible to assume that *inter-domain* routing will ever ensure symmetrical routes. We can, however, prevent asymmetry arising from non-unique shortest paths in *intra-domain* routing protocols by using random link costs (described later). This is useful for creating symmetrical intra-domain routes, which remove some of the problems mentioned above (Section 2.1), and also makes a new form of congestion control possible. RPCC (see Section 4 later) relies on symmetrical intra-domain routes between the source and the router to pass back a *congestion bit* indicating congestion at the router. A typical university or company domain today has Ethernet or FDDI links while the backbone is a *T1* or a *T3* link which is quite slow in comparison. Most congestion, thus, occurs at the boundary between the domain and the backbone. Intra-domain route symmetry and RPCC can thus enhance the throughput of most applications. Further, protocols like NTP [Mil85] will also benefit when used within a domain.

One way to ensure route symmetry is to use a symmetrical tiebreaker (when choosing among multiple shortest path routes) that remains the same when the roles of source and destination are reversed. One way to do this, due to Paul Koning, is to use the sorted list of node IDs in a path for choosing among several equal-cost paths. Clearly, this tiebreaker remains unchanged when the source and destination nodes are interchanged. We treat this tiebreaker as a string and choose the shortest path that has the smallest (in lexicographic order) string. Unfortunately, this method adds considerable complexity to route computation in order to also compute the sorted path list. Bellman-Ford protocols like RIP would have to pass the entire path; which would greatly increase the size of routing messages. Both Bellman-Ford and Link State protocols would have the additional overhead of sorting the path ID list. Sorting can increase the complexity of Dijkstra's algorithm from $O(N \log N)$ to $O(N^2 \log N)$, where N is the number of nodes.

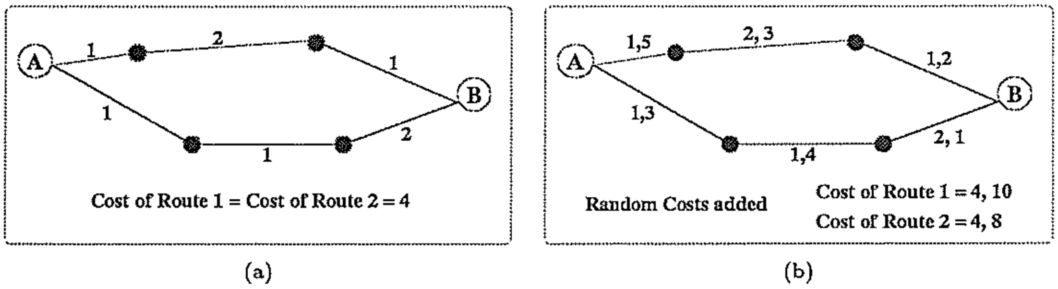


Figure 1: Random costs can break route asymmetry

We introduce a new technique that is simpler and much more efficient than the sorted list method. The idea is to use an additional (small) random number component along with the usual link cost

(Figure 1). Each link between two nodes gets a small integer random cost assigned either manually by a manager, or automatically by a leader node elected on the link. The sum of the random costs serves as a tiebreaker. When there are two or more paths with the same cost (between nodes A and B in Figure 1(a)), then the path with the least random cost (route 2, Figure 1(b)) is selected.

At this point, we would like to emphasize that adding random costs does not change the convergence behavior of the Bellman-Ford algorithm.[2] The random costs are used only for breaking ties between two equal cost paths. The actual and random costs are added and stored separately, and only the actual cost is compared to *infinity* when checking for convergence. We show in Section 3 later how to modify the Routing Information Protocol (RIP) [Hed88] to incorporate random costs. The simplest way is to split the original 32 bit cost metric into a 16 bit actual cost and a 16 bit random cost, and to redefine hopcount *infinity* from its current value of 16 to $16 * 2^{16}$. See Section 3 for details.

This method leaves the complexity of route computation unchanged and only requires that routing messages carry the extra random component along with the normal link cost. Conceptually routing is unchanged except that instead of using a single number for the cost of a link or path we need to use a tuple (c, r) to represent the cost (Figure 1(b)); c is the usual cost and r is the random cost. We compare tuples lexicographically: $(c, r) < (c', r')$ if $c < c'$ or $c = c'$ and $r < r'$. Except for this change, route computation in Bellman-Ford or Dijkstra algorithms remains unchanged.

Our new method of adding random costs raises an interesting question. How big should the random numbers be to make the probability of non-unique shortest paths sufficiently low? Below, we present a theoretical analysis followed by simulation results which show that 10 bit random numbers produce good results.

2.4. Probabilistic Analysis of the Random Cost Algorithm

Consider an arbitrary graph with arbitrary (but symmetric) link costs. Suppose we choose random link costs uniformly in the range $\{1, ..., c\}$. What is the probability that there is a non-unique shortest path? In its fullest generality this is a very hard problem, because even enumerating the set of shortest paths for a given graph is difficult.

Our key insight, which allows us to bound the required probability, is to use a powerful *Isolating Lemma* due to Mulmuley, Vazirani, and Vazirani [MR95]. Originally, the Isolating Lemma was invented to calculate the probability that a graph would contain a unique *perfect matching* given a random assignment of *node* weights. We, however, use it to calculate the probability that a graph will contain *all unique shortest paths* given a random assignment of *edge* weights.

Consider any set X of m elements. Suppose that each element is assigned an integer random cost chosen uniformly and independently in the range $\{1, ..., 2m\}$. Consider any family F of subsets of X. The cost of a subset $S \in F$ is the sum of the weights of the elements of X contained in S. The Isolating Lemma states that the probability that there is a unique minimum weight subset in F is at least $1/2$. This lemma is surprising and powerful because it works regardless of the way we choose the subsets F of X! A proof can be found in [MR95].

To use the Isolating Lemma, we let X be the set of all links. Thus m is the number of links in the graph. We take F to be the the set of all shortest paths between a certain pair of nodes. Our objective is to find the probability of a unique minimum cost(weight) path in F for the source-destination pair in question. It follows directly from the Isolating Lemma that if we choose random

[2] The convergence of Bellman-Ford often depends on the maximum cost of a route. Thus, changing the cost metric could, without care, affect convergence time.

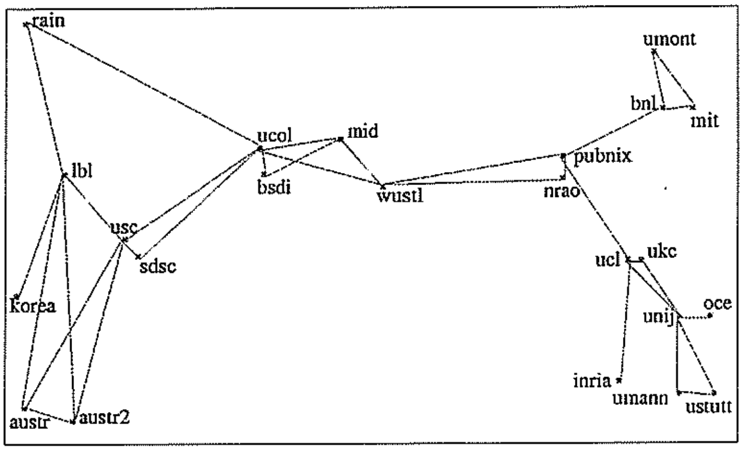


Figure 2: A Sample Internet topology consisting of 23 nodes.

Sl.	Name	Location	Sl.	Name	Location
1	rain	Portland, Oregon	13	umont	Montreal, Canada
2	lbl	Berkeley, CA	14	ucl	London, U.K.
3	usc	Los Angeles, CA	15	ukc	Canterbury, U.K.
4	sdsc	San Diego, CA	16	unij	Nijmegen, The Netherlands
5	ucol	Boulder, CO	17	oce	Venlo, The Netherlands
6	bsdi	Colorado Springs, CO	18	umann	Mannheim, Germany
7	mid	Lincoln, Nebraska	19	ustutt	Stuttgart, Germany
8	wustl	St. Louis, MO	20	inria	Sophia, France
9	pubnix	Fairfax, VA	21	korea	Pohang, South Korea
10	nrao	Charlottesville, VA	22	austr	Melbourne, Australia
11	mit	Cambridge, MA	23	austr2	Newcastle, Australia
12	bnl	Brookhaven, NY			

Table 1: List of Sites used in the above sample Internet topology

integer link costs for each link in the range $\{1, ..., 2m\}$, then the probability of a shortest path is at least $1/2$.

Since $1/2$ is too high a probability of failure, we generalized the Isolating Lemma slightly. The generalization shows that if the random link costs are chosen uniformly and independently in the range $\{1, ..., km\}$, the probability of a unique minimum weight shortest path is at least $1 - 1/k$. This shows that that we can make the probability of a unique shortest path as high as we like by increasing the number of bits allocated to the random cost. For example, the analysis indicates that in a 128 link network, we can get roughly 90 percent probability of a unique shortest path using 10 bit random costs.

While it is satisfying to obtain a theoretical estimate of the probability that is completely independent of the network topology, we note that the the theoretical result is a gross underestimate of the real success probability. For example, we are dealing only with a specific family of subsets (shortest cost paths between a particular source-destination pair) and not all possible subsets. Our simulation results, presented below, show much better results.

2.5. Simulation Results

We simulated *Bellman-Ford* route computation, as used by the *RIP* protocol, over a few sample Internet topologies. We used a subset of the topology of Internet sites used for routing measurement in [Pax96a, Pax96b]. This sample topology (Figure 2) consisted of 23 nodes, 13 from North America, 7 from Europe, 2 from Australia, and 1 from Asia. The random integer costs were chosen from a space of 1 through the number of nodes. We also used a subset of the Swiss Academic & Research Network to simulate a reasonably complex domain. The Swiss topology (Figure 3) consisted of 19 nodes. Table 1 lists the location of the sites in our sample Internet topology (Figure 2). By using real Internet topologies, we hope to make our analysis and results pertain to the Internet as closely as possible.

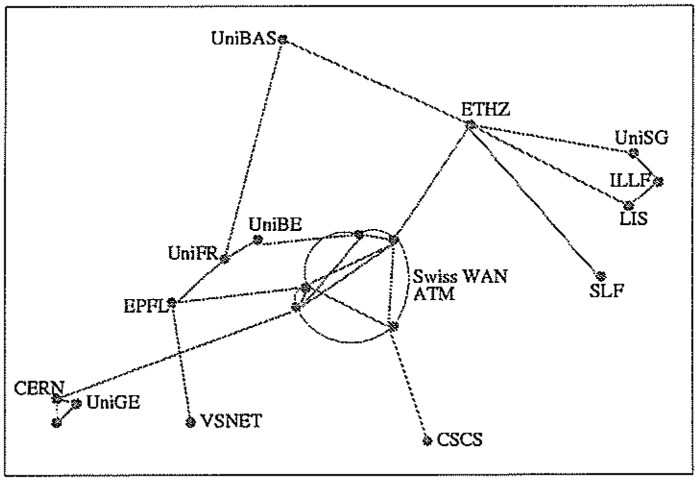


Figure 3: A Swiss Network topology consisting of 19 nodes selected from the Swiss Academic & Research Network.

The Distance-Vector algorithm was simulated several times over these two topologies, and we measured the number of asymmetrical routes in each topology. Table 2 shows the percentage of symmetrical routes obtained for the different topologies with random costs. With no random numbers, our algorithm yielded about 8 route asymmetries for the Internet topology and about 12 for the Swiss one. The use of 10 bit random numbers eliminated all route asymmetries.

Random Number Space	Topology	
	Sample Internet	Swiss Network
19	-	88.2
23	98.1	-
8 bits	99.95	98.95
10 bits	100	100

Table 2: Percentage of symmetrical routes obtained for different topologies. 10 bit random numbers appear to be sufficient for most topologies.

We also looked at a few local domain topologies but most of them had a *tree* topology, and hence no asymmetries. Thus, we believe that 10 bit random numbers should suffice for most topologies. The above link cost modification can be used to make any shortest path routing algorithm (like OSPF [Moy97]) symmetric. In the next section we describe in detail how we can modify a popular intra-domain protocol RIP [Hed88].

3. Modifications to Routing Information Protocol

Routing Information Protocol or RIP is a Bellman-Ford (or Distance-Vector) algorithm based protocol used for routing computation in the Internet. RIP is an *interior gateway protocol* - that is, RIP is used within domains or *Autonomous Systems.* It is intended for use in an IP based Internet. In RIP, a router calculates the shortest cost to other routers and then distributes the new costs to its neighbors. This process continues till the costs stabilize. In this section we discuss details of how we can modify RIP to incorporate random costs. From our experimental results (section 2.5) we see that 10 bit random numbers should suffice for most network topologies.

Figure 4 the shows the message format of RIP version 2 as described in RFC [Mal94]. The fields *Route Tag, Subnet Mask* and *Next Hop* were unused in RIPv1, but they were added in RIPv2 to enhance RIP features and to make RIP more robust. *Route Tag* was added to provide a method of separating "internal" RIP routes (routes for networks within the RIP routing domain) from "external" RIP routes, which may have been imported from an EGP or another IGP. *Subnet Mask* information makes RIP more useful in a variety of environments and allows the use of variable subnet masks on the network. *Next Hop* was added to prevent extra hops in routes.

<table>
<tr><td>Command(1)</td><td>Version(1)</td><td colspan="2">unused(2)</td></tr>
<tr><td colspan="2">Address Family Identifier(2)</td><td colspan="2">Route Tag (2)</td></tr>
<tr><td colspan="4">IP Address (4)</td></tr>
<tr><td colspan="4">Subnet Mask (4)</td></tr>
<tr><td colspan="4">Next Hop (4)</td></tr>
<tr><td colspan="4">Metric (4)</td></tr>
</table>

Figure 4: RIP Version 2 Message Format, each row is a 32 bit word

The simplest method of adding random costs is to modify the fields in the current message format rather than adding a new field. The cost metric for routes is a 32 bit field. However the value of *infinity* is only 16, which requires 5 bits. Therefore we propose reducing the value of the cost metric to 16 bits (see Figure 5). We can use the lower order 16 bits for the random costs. By reading the two fields as a single 32 bit integer, we can still make cost comparison in a single step. Since the actual cost is in the higher 16 bits, the actual cost will get preference when comparing costs; the lower 16 bits of random cost will be used to break ties.

<table>
<tr><td>Command(1)</td><td>Version(1)</td><td colspan="2">unused(2)</td></tr>
<tr><td colspan="2">Address Family Identifier(2)</td><td colspan="2">Route Tag (2)</td></tr>
<tr><td colspan="4">IP Address (4)</td></tr>
<tr><td colspan="4">Subnet Mask (4)</td></tr>
<tr><td colspan="4">Next Hop (4)</td></tr>
<tr><td colspan="2">Metric (2)</td><td colspan="2">Random Cost (2)</td></tr>
</table>

Figure 5: Proposed RIP Message Format; note the change in the last word. The value of *infinity* is left shifted by 16 bits to keep convergence unchanged. The random cost is also prevented from overflowing onto the actual cost fields.

However this scheme requires us to redefine the value of *infinity* to $16 * 2^{16}$ because we have left shifted the cost metric by 16 bits. It should be noted that the convergence behavior of RIP is unchanged. The link costs are represented in the high 16 bits; hence to ease comparison with *infinity*, the latter has to be left shifted by 16 bits too. This allows us to compare the 32-bit entire cost (actual + random) against *infinity.* Furthermore, to keep the actual and random cost fields separate, we have to ensure that the random cost does not overflow onto the actual cost metric. We do this as follows. The random cost of a link is chosen up to 10 bits. However the random cost of a route can be up to 16 bits. Thus we can have a maximum of 2^6 ie. 64 links. This seems fair enough since the value of hopcount *infinity* allows only 16 hops.

As far as protocol behavior is concerned there is no change from RFCs [Hed88] & [Mal94], except that hopcount *infinity* is redefined as $16 * 2^{16}$. The 32-bit cost is added and compared just as before. Confining the random cost to the low 16 bits breaks ties in case the actual costs are the same. Further, the protocol should ensure that the random cost does not overflow onto the actual cost metric. In practice such an event will not occur because before the random cost of a route reaches 16 bits, its actual cost will have reached *infinity.* According to RFC [Hed88], such routes are not propagated.

A simple way to ensure interoperability and backward compatibility is to break the original domain into two domains connected by (say) a BGP router. We gradually enlarge the domain containing the new RIP by reprogramming one router at a time in the domain containing the old RIP routers. When the transition is complete, we return to a single domain.

4. Reverse Path Congestion Control

If network routes are symmetrical, then a new form of congestion control becomes possible. We call our new scheme Reverse Path Congestion Control (RPCC). Since we have already seen that symmetrical inter-domain routing is infeasible, we will only assume that intra-domain routing is symmetrical. RPCC is similar to the DECbit [RJ88] scheme except in the way the congestion bit is passed. In the DECbit scheme, a congestion bit is passed at the routing layer from the point of congestion to the destination, and then back to the source via the transport header. In RPCC, the congestion bit is passed directly from the point of congestion back to the source.

It seems unlikely that a congestion bit can be passed on the complete path between a source and destination (as is required in the DECbit scheme) in the Internet because this would require cooperation from the various networks that comprise the Internet. However, it seems more plausible to hope that congestion bits can be passed within a single domain, and that routes can be made symmetrical within a domain (see Section 2). Thus, we would like to propose RPCC not as a modification to the DECbit scheme, but as another congestion control scheme which can take advantage of symmetry in routes. Unlike the DECbit scheme, RPCC does not require the congestion bit to be passed across the Internet, and hence can be used within a domain or an *Autonomous System*. In Section 4.2, we summarize how RPCC is different from the DECbit scheme.

We consider the use of RPCC as a congestion control mechanism that *augments* the normal congestion control mechanisms of TCP. If congestion occurs in the originating domain, or in any large region that includes the source in which the RPCC bit can be carried, then RPCC can potentially provide fast response to congestion. In all other cases, we rely on the normal TCP mechanisms. If congestion occurs at the boundary between the originating domain and the backbone, then RPCC can potentially provide faster detection of such congestion.

We proceed by describing the main idea of RPCC followed by algorithmic details. We then describe some simulations and comparisons with existing TCP mechanisms.

4.1. RPCC: The Idea

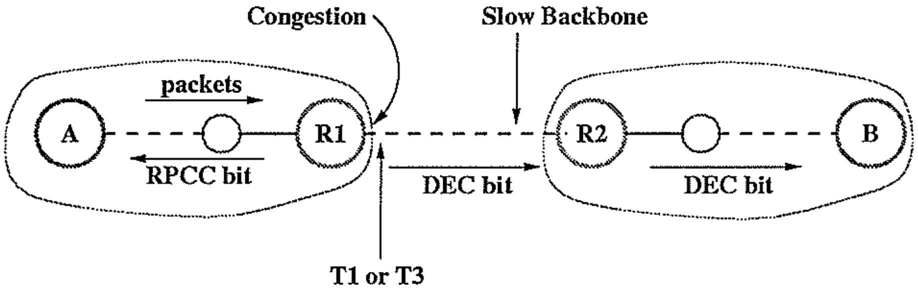


Figure 6: Reverse Path Congestion Control. Congestion occurs at $R1$; the *congestion bit* is set on all packets flowing to A from $R1$. Unlike DECbit, it is not passed to B and all the way back across the entire backbone. Typically, domains have outgoing *T1* or *T3* links, and most congestion occurs at the boundary between the domain and the backbone.

Suppose a host A sends a packet to host B through a router *R1* (Figure 6). If *R1* senses congestion building up on its outgoing link to B, then according to the DECbit [RJ88] scheme *R1* can send feedback to A by setting a *congestion bit* in the packet to B. However this feedback takes one round-trip delay. Our RPCC scheme allows router *R1* to send feedback to A, not through B, but by setting the congestion bit in packets/acknowledgments flowing to A from B (and all other hosts which are communicating to A). This reduces the feedback delay significantly, especially if the congestion build-up occurs at the point where packets from a LAN flow out onto a WAN (which is the usual case). More importantly, it avoids the need for an end-to-end bit path which is essential for the Internet.

We do this as follows. *R1* monitors its outbound queue length whenever a packet or acknowledgment is forwarded. When the average queue length of an outbound queue reaches some *feedback threshold*, *R1* detects congestion. *R1* then sets a bit in a table entry corresponding to the the outgoing congested link. Next, when *R1* routes a packet *arriving* through that congested link, *R1* sets the congestion bit on that packet. Thus all senders get to know of the impending congestion, and can reduce their transmission rates.

4.2. RPCC vs DECbit

Noticing the obvious similarities between RPCC and the DECbit scheme [RJ88], a reader might infer that RPCC is a straightforward enhancement of the DECbit scheme that works only in case of symmetrical routes. However, there are a few distinct differences between the two. Most importantly, the congestion bit in the DEC scheme is passed from the point of congestion to the destination, and then sent all the way back to the source. In RPCC, the congestion bit is directly passed back to the source (see Figure 6). This has many interesting consequences. It will potentially be faster because the congestion bit does not travel all the way to the destination, especially if congestion occurs near the source (which is the case most often). Further, unlike DECbit, which requires cooperation from all the routers and the destination to pass the congestion bit back to the source, RPCC requires only the routers (upto the point of congestion) to do so. Thus, RPCC can be used even within a domain. As stated before, most congestion occurs at the boundary between the domain and the backbone.

4.3. RPCC: Algorithm and Simulations

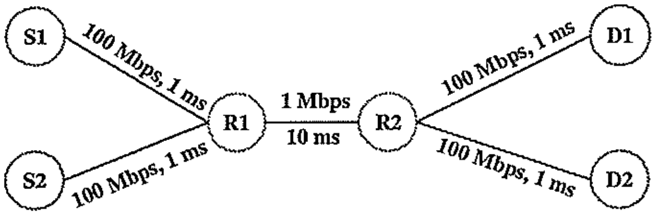


Figure 7: Topology used for testing RPCC; link *R1-R2* is a low bandwidth backbone link, all other links are high bandwidth ones.

We use the *Adaptive Averaging* technique used in the DECbit [RJ88] scheme to calculate the average queue length of a link. The feedback threshold value is selected as 1.0. We tested our RPCC scheme using the *ns* simulator [nLNS96], running a TCP-Reno [Ste94, WS94] implementation. We made one change to the sender TCP-Reno. On receipt of a packet or ack with the congestion bit set, the congestion window *cwnd* is reduced by 1/8th of its current value and the slow start threshold *ssthresh* is set to one-half of the current *cwnd*. The remainder of the TCP congestion control mechanism was left unchanged.

We observed a remarkable increase in the throughput obtained by the sender TCP, especially during the start-up phase (similar to the results in [Hoe96]). There were almost no packet drops. RPCC also provides fairness to new sources that start after an existing source has been transmitting at its full capacity.

Figure 7 shows the network topology we used for testing our RPCC scheme. The link from router *R1* to *R2* represents a typical low bandwidth *WAN* with a throughput of 1 Mbps and a latency of 10 ms. *S1*, *S2*, *D1* and *D2* are end-nodes connected to the *WAN* through high bandwidth LANs (100 Mbps, 1 ms latency).

Using *ns* we simulated an FTP (over TCP-Reno) from *S1* to *D1*. The output queue length at *R1* was set to 5.

Figure 8 shows the performance of RPCC TCP over TCP-Reno. The routers used the *RED* gateway scheme [FJ96]. The plot in Figure 8(a) depicts the sequence numbers sent out by the sender TCP with time. Note the startup behavior of TCP-Reno at 0.2 sec. The exponential growth of the congestion window (Figure 8(b)) results in packet losses and hence timeout (from 0.25 to 0.85 sec). Fast retransmit (at 0.20 and 0.25 sec) fails to recover from the congestion build up. The sender

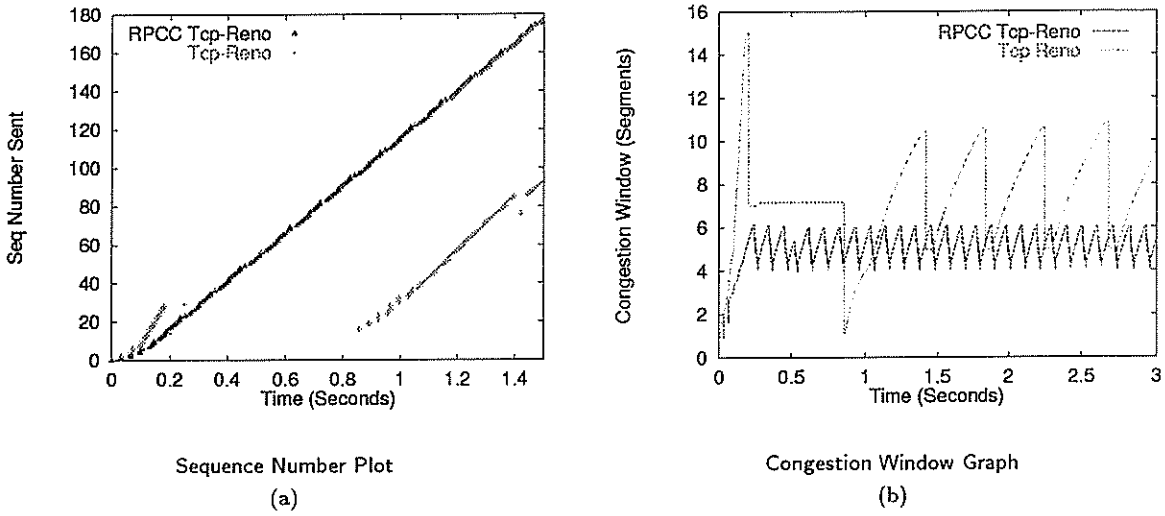


Figure 8: RPCC vs Original TCP-Reno; note the improved start up behavior and the reduced oscillation in the congestion window.

goes into slow-start (the congestion window drops to 1). Also note the packet drop at 1.4 sec (TCP needs an occasional packet drop to retard the window growth). The rest of the simulation upto 5 seconds was an extrapolation of the behavior seen here after 1 sec. The RPCC TCP, however, never loses a packet; this leads to much improved throughput. This is explained by the congestion window graph (Figure 8(b)). The sender's window oscillates less and hence avoids the build up of congestion and packet loss.

The startup behavior of TCP has been previously discussed in [Hoe96]. We also simulated the mechanisms in [Hoe96] somewhat crudely and found that they did nearly as well as RPCC for a single sender. However, for multiple concurrent senders our RPCC scheme did much better. The improved startup behavior of RPCC TCP makes it potentially worthwhile for bursty applications like the Web and other short duration connections.

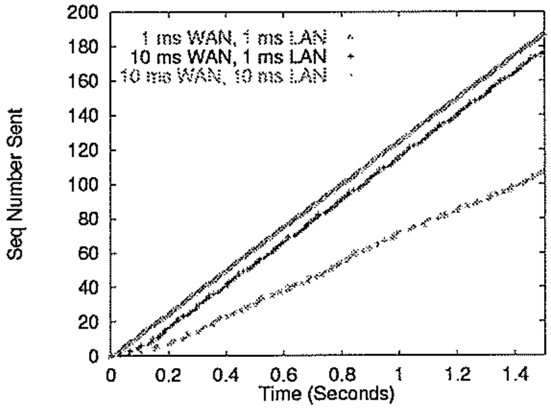
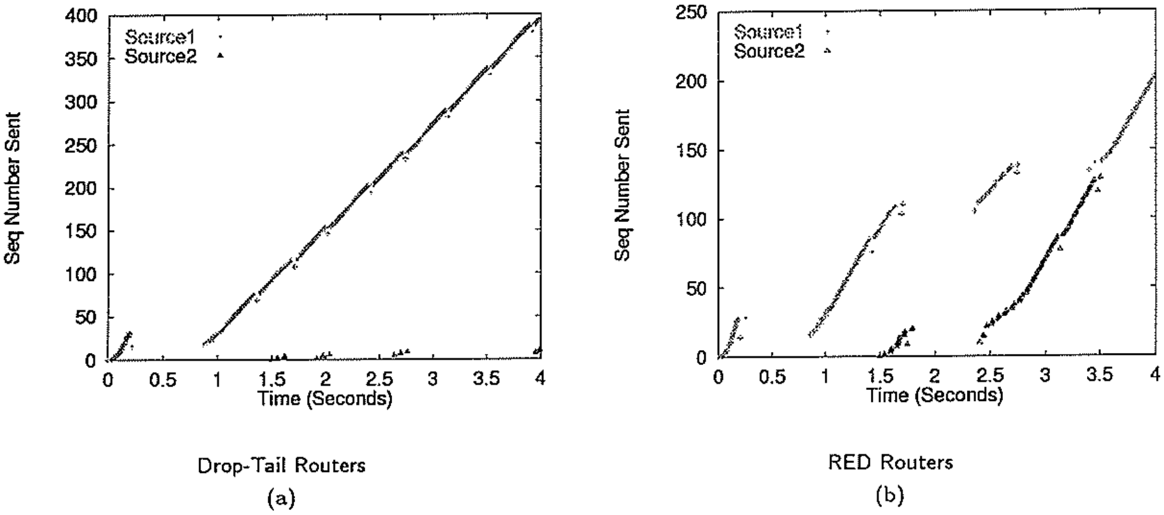


Figure 9: Behavior of RPCC TCP as the network characteristics are changed; RPCC performs better if the point of congestion is closer to the source.

To see how the performance of RPCC TCP depends on the point in the network where congestion



Drop-Tail Routers

(a)

RED Routers

(b)

Figure 10: Behavior of TCP-Reno with two FTP sources; the RED scheme is provides better fairness but leads to lower throughput.

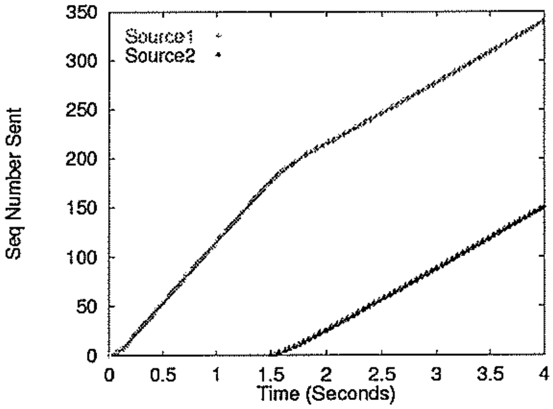


Figure 11: Behavior of RPCC TCP with two FTP sources; both the sources are transmitting at the same rate after 1.5 seconds; there were no packet drops.

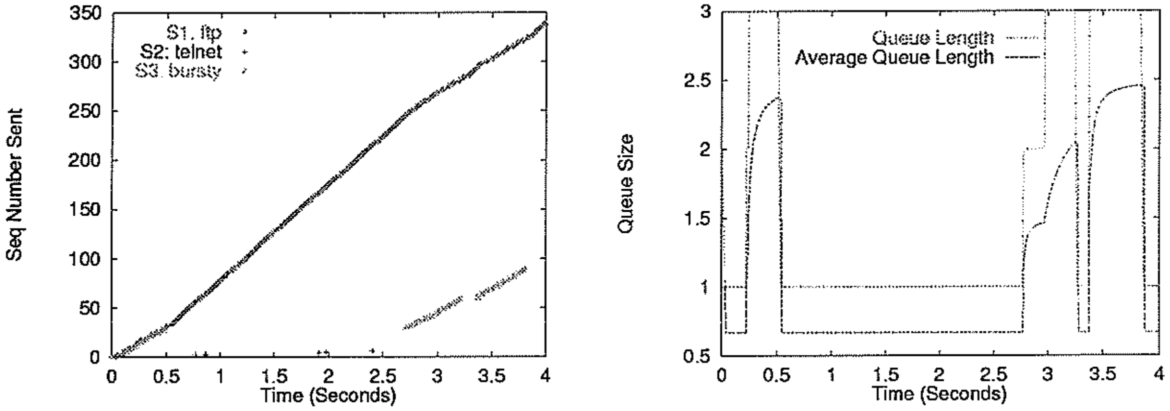
occurs, we varied the link delays in Figure 7.

When the delay between *R1* and *R2* is changed to 1 ms, the throughput of RPCC TCP increases marginally (the topmost plot in Figure 9). However when the delay between *S1* and *R1* is increased to 10 ms, the throughput decreases much sharply (the bottom plot in Figure 9). A larger delay between *S1* and *R1* means the congestion in the network occurs farther (at *R1*) from the source (*S1*). From this we conclude that the closer the point of congestion is to the source, the better RPCC performs. In other words, RPCC TCP will give better performance for networks for which the congestion occurs as close to the sender as possible (for example at the boundary between the domain and the backbone).

Fairness to Multiple Connections:. Next, we tested the behavior of TCP when a second source starts transmitting while a first source has been transmitting at full capacity (Figure 10). We started another FTP source from *S2* (see Figure 7 above) to *D2* at time 1.5 seconds.

Figure 10(a) shows the behavior of TCP-Reno with *Drop-Tail* router queues. Note how the scheme is unfair towards the second FTP source. On analyzing the behavior we observed that the source 2 TCP never comes out of the slow-start phase while the source 1 TCP never loses enough packets to get to the slow-start state. This is because in a *Drop-Tail* router the packets at the *tail* of the queue are dropped. Thus both the sources lose packets with equal probability (which is somewhat unfair because the first source has been transmitting more than the second source).

The *RED* scheme (Figure 10(b)) provides fairness but at reduced throughput. Unlike the *Drop-Tail* routers (discussed above), the *RED* based packet dropping ensures that a source loses packets proportional to its transmitting rate. Thus the first source looses more packets than the second source. This causes the source 1 TCP to go into slow-start, and hence slow down appreciably. However packet losses and timeouts considerably reduce the total throughput.



Sequence Number Plot

(a)

Queue Size at Router

(b)

Figure 12: Behavior of RPCC TCP with different types of traffic sources; RPCC can accommodate bursty sources well, it also provides low delay to Telnet sources because of small queue size at the router.

RPCC TCP (Figure 11) provides fairness at improved congestion control and avoids packet losses. Note how after 1.5 sec both the FTP sources are transmitting at the same rate (the slopes of the plots are equal). Also note the increased throughput (because packets are not lost) as compared to Figure 10. The behavior seen here is similar for *RED* as well as *Drop-Tail* routers (once again because packets are not lost). In RPCC each packet of a source carries the congestion bit. Hence a particular source gets congestion feedback proportional to the rate it is transmitting at. This guarantees fairness without packet loss.

Telnet and Bursty Sources:. Figure 12 shows the behavior of RPCC TCP with Ftp, Telnet, and Bursty traffic sources. All the traffic sources follow the same network path via the same congested link (*R1* - *R2* in Figure 7). The Ftp source generates traffic continuously (controlled by the TCP window). The Telnet source and the Bursty sources generate packets according to an exponential distribution with a mean inter-arrival time of 1 second. The burst size for the bursty source is 30 packets. Figure 12(a) shows the sequence number plot while Figure 12(b) shows the queue size at the router *R1*. Adaptive averaging and RPCC keep the queue length quite small, and hence the queue at the router is able to accommodate bursty sources. The ftp source obtains high throughput as desirable. Thus, bursty sources like HTTP [FGM$^+$97] can also benefit from RPCC. Telnet sources observe low queuing delays too.

4.4. Summary

From the above experiments and simulations, we see that RPCC improves the start up behavior of TCP. The sender's TCP congestion window oscillates less, and hence avoids congestion build up and packet loss. We also observed that RPCC provides fairness to multiple connections because a source gets congestion feedback proportional to the rate it is transmitting at. RPCC keeps a router's queue size small, and hence is able to accommodate sudden bursts of packets. Telnet sources see low delay because of the small queue size.

5. Conclusions

In this paper, we studied the problem of asymmetric routes. There seems little one can do to prevent asymmetries due to policy routes unless the policy makers can be made to cooperate! We showed, however, a simple new technique of avoiding non-unique shortest paths by adding random cost tiebreakers. The technique applies to all shortest cost routing algorithms; we described in detail how RIP could be modified and why 10 bit random numbers seem sufficient. This technique could be used to ensure symmetrical routing within a domain.

Then, we discussed a new form of congestion control that we called RPCC. RPCC is limited in applicability to networks that ensure symmetrical routes and that can pass a congestion bit from the congestion point back to the source in the routing header. For networks in which this is not universally true (e.g., the Internet), RPCC can only be used as an augmenting mechanism. It does, however, provide faster response than the DECbit scheme and can be applied to networks in which a complete end-to-end bit path does not exist. This can happen, for example, if some of the routers on the path do not pass such a congestion bit. Even as an augmenting mechanism, RPCC is still useful because most congestion today occurs at the boundary between the domain and the backbone.

Both our ideas (adding random costs as tiebreakers and RPCC) are orthogonal and can be used independently. One can avoid the use of any of these ideas by living with the suboptimality caused by asymmetric link costs and routes. Is it worth adding new mechanisms to avoid this suboptimality? Our paper has concentrated on exploring the issues involved and describing possibly useful new mechanisms. A more precise quantification of the benefits of symmetry is left to future work. We end, somewhat tongue in cheek, with the following quote from the physicist Abdus Salaam [Fer91]:

A broken symmetry breaks your heart . . .

References

[Bel86] E.T. Bell. *Men of Mathematics.* Simon and Schuster, Touchstone Edition, 1986.

[Fer91] Timothy Ferris. *The World Treasury of Physics, Astronomy and Mathematics.* Little Brown, 1991.

[FGM+97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol - HTTP/1.1. *RFC 2068*, January 1997.

[FJ96] S. Floyd and V. Jacobson. Random Early Detection gateways for Congestion Avoidance. In *IEEE/ACM Transactions on Networking*, volume 1, number 4, pages 397–413, August 1996.

[Hed88] C. Hedrick. Routing Information Protocol. *STD 34, RFC 1058*, June 1988.

[Hoe96] Janey C. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In *Proceedings of the ACM SIGCOMM '96 Symposium*, volume 26, number 4, pages 270–280, August 1996.

[Jac88] Van Jacobson. Congestion Avoidance and Control. In *Proceedings of the ACM SIGCOMM '88 Symposium on Communications Architectures and Protocols*, volume 18, number 4, pages 314–329, August 1988. part of ACM Sigcomm Computer Communication Review.

[Kes91] S. Keshav. A Control-Theoritic Approach to Flow Control. In *Proceedings of the SIGCOMM '91 Symposium*, pages 3–15, September 1991.

[Mal94] G. Malkin. RIP Version 2 Carrying Additional Information. *Xylogics Inc., RFC 1723*, November 1994.

[Mil85] D.L. Mills. Network Time Protocol (NTP). *RFC 958*, September 1985.

[Moy97] J. Moy. OSPF version 2. *RFC 2178*, July 1997.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[nLNS96] ns LBNL Network Simulator. *LBNL Network Research Group.* http://www-nrg.ee.lbl.gov/ns, 1996.

[Pax96a] Vern Paxson. *An Analysis of End-to-End Internet Dynamics.* (Partial Draft) ftp://ftp.ee.lbl.gov/.vp-routing.long.ps.Z, 1996.

[Pax96b] Vern Paxson. End-to-End Routing Behavior in the Internet. In *Proceedings of the ACM SIGCOMM '96 Symposium*, volume 26, number 4, pages 25–38, August 1996.

[Per92] R. Perlman. *Interconnections: Bridges and Routers.* Addison-Wesley, 1992.

[RJ88] K. K. Ramakrishnan and Raj Jain. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer. In *Proceedings of the ACM SIGCOMM '88 Symposium*, pages 303–313, August 1988. Also available as DEC-TR-508, Digital Equipment Corporation.

[RL95] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). *RFC 1771*, March 1995.

[Ste94] W. Richard Stevens. *TCP/IP Illustrated, Volume 1, The Protocols.* Addison-Wesley Professional Computing Series, 1994.

[WS94] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated, Volume 2, The Implementation.* Addison-Wesley Professional Computing Series, 1994.