UNIVERSITY OF CALIFORNIA, SAN DIEGO

XL: A Communication-Efficient Routing Algorithm

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Kirill Levchenko

Committee in charge:

Professor Ramamohan Paturi, Co-Chair
Professor Stefan Savage, Co-Chair
Professor Fan Chung Graham
Professor Russell Impagliazzo
Professor Alon Orlitsky

2008

The dissertation of Kirill Levchenko is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____
Co-Chair

_____
Co-Chair

University of California, San Diego

2008

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Ramamohan Paturi and Professor Stefan Savage both of whom advised and supported me over the past seven years. I have received invaluable guidance from Professor George Varghese and Professor Geoffrey M. Voelker. I am also grateful to my fellow graduate students without whom graduate school would not have been totally awesome. Finally, I owe much to my family, Heather and Dasha.

This dissertation describes my work on the XL routing algorithm (joint work with Ramamohan Paturi, Geoffrey M. Voelker, and Stefan Savage) which has been presented at the 2008 ACM SIGCOMM Conference in Seattle, Washington and appears in the proceedings of the conference, published in *Computer Communication Review*, 38(4). The first chapter contains additional historical background on routing in communication networks; it has not been previously published.

VITA

2001 B. S. in Mathematics and Computer Science, University of Illinois, Urbana-Champaign.

2008 Ph. D. in Computer Science, University of California, San Diego.

PUBLICATIONS

C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage, "Spamalytics: An Empirical Analysis of Spam Marketing Conversion," *CCS 2008*.

K. Levchenko, G. M. Voelker, R. Paturi, and S. Savage, "XL: An Efficient Network Routing Algorithm," *SIGCOMM 2008*.

C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage, "On The Spam Campaign Trail," *LEET 2008 Workshop*.

C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, and S. Savage, "The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff," *LEET 2008 Workshop*.

J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker, "Unexpected Means of Protocol Inference," *IMC 2006*.

A. R. Calderbank, A. Gilbert, K. Levchenko, S. Muthukrishnan, and M. Strauss, "Improved Range-Summable Random Variable Construction Algorithms," *SODA 2005*.

K. Levchenko, R. Paturi, and G. Varghese, "On the Difficulty of Scalably Detecting Network Attacks," *CCS 2004*.

A. Gilbert and K. Levchenko, "Compressing Network Graphs," *LinkKDD 2004 Workshop*.

ABSTRACT OF THE DISSERTATION

XL: A Communication-Efficient Routing Algorithm

by

Kirill Levchenko
Doctor of Philosophy in Computer Science

University of California San Diego, 2008

Professor Ramamohan Paturi, Co-Chair
Professor Stefan Savage, Co-Chair

We describe and analyze a new communication-efficient routing algorithm for packet forwarding networks such as the Internet. The explicit design objective of our routing algorithm, called XL, is to reduce the communication overhead of routing, allowing the routing algorithm to support larger networks without resorting to artificial network partitioning techniques such as OSPF areas. We achieve this by allowing suboptimal forwarding paths up to a user-specified *stretch* factor. (By setting stretch to 1 it is also possible to force optimal routing.)

The XL routing algorithm is a *link state* algorithm, meaning that network nodes disseminate information about the state of links in the network. The essential difference between XL and the classical link state algorithm used by OSPF and IS-IS is in the semantics of link state updates: in the classical algorithm, a link state update specifies the *exact* link cost, while in XL it specifies an *upper bound* on the actual cost. This allows XL to suppress updates when the route cost do not decrease significantly.

In addition to the formal specification and correctness proof, we also compare XL to four existing routing algorithms in simulation. Two of these algorithms are the classical distance vector algorithm and the link state algorithm which are the basis of the RIP and OSPF routing protocols, respectively. The other two are state-of-the art experimental protocols: distance vector with parent pointer, and the link vector algorithm, both based on the idea of restricting updates to those about links in a

node's shortest-path tree. Experimental results show that our algorithm consistently generates fewer updates in response to network changes, in some cases by nearly an order of magnitude.

Chapter 1

Introduction to Network Routing

The end of the twentieth century has come to be defined by technologies like the telephone and the Internet which have given us an unprecedented ability to communicate and access information. Making this possible are *communication networks*: the mass of wires, antennas, switches, and routers alive with design and purpose. The animating principle of any communication network is routing: the mechanism that defines how information gets from point A to point B in the network.

The subject of this work is routing in a packet-switched network such as the Internet, characterized by semi-reliable links and a simple forwarding mechanism based on destination addresses. Before delving into our problem, however, we survey the history of communication networks and routing.

1.1 Before the Internet

In this section we describe a number of communication networks that predated the Internet. We begin with postal networks.

1.1.1 The Postal Network

Perhaps the oldest example of a modern communication network is the postal service. Its roots go back as far the as the messenger services of ancient empires as described by Herodotus [22, Book 8, Chapter 98]. Early postal networks were tied to the road networks and were, in some sense, an extension of them. In some cases,

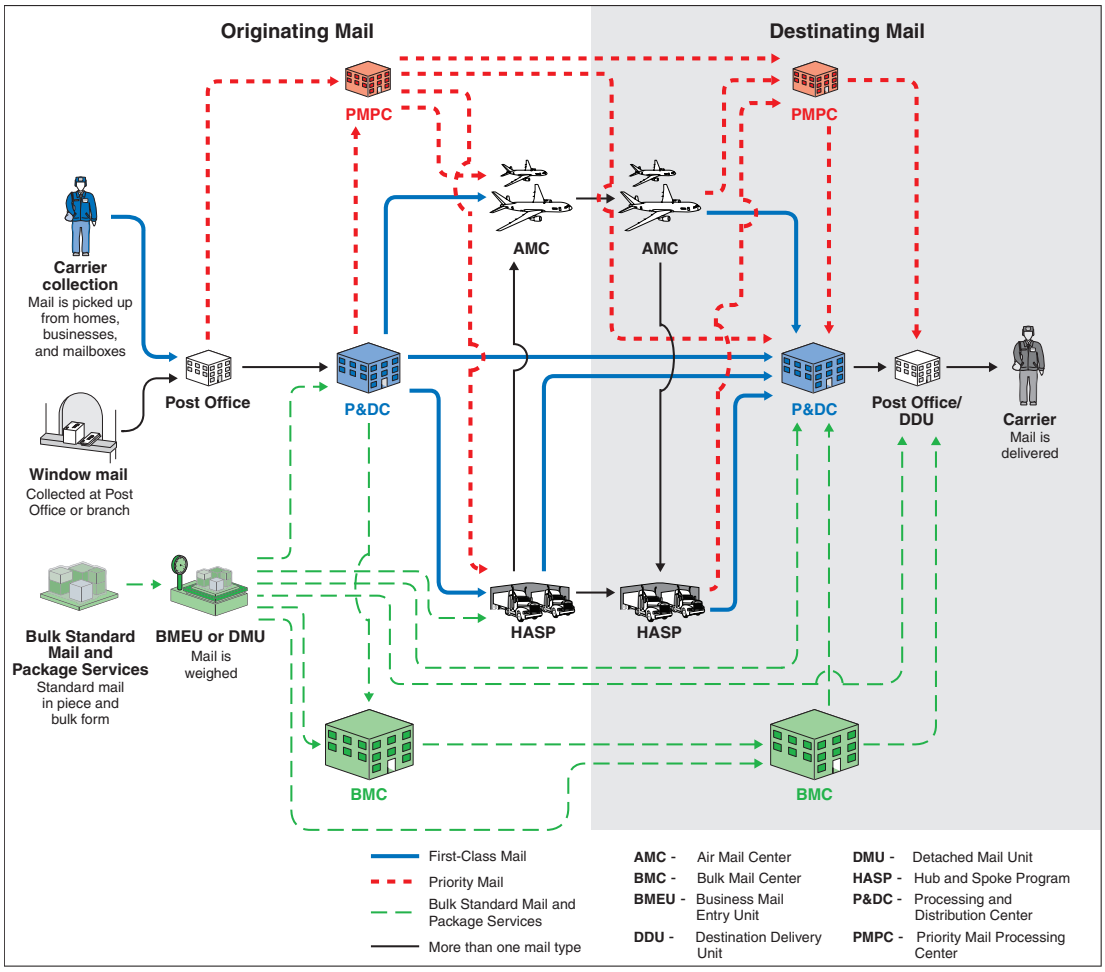


Figure 1.1: Mail routing in the United States Postal Service. The postal network is a three-level hierarchy: post offices, where mail enters and leaves the network, at the bottom, Regional Processing and Distribution Centers above them, and the Hub and Spoke Program nodes and Air Mail Centers at the top. Image source: United States Government Accountability Office [60].

roads were built and maintained *because* of the mail service: the US Constitution, in fact, expressly gives Congress the power "to establish post-offices and post-roads." The cost of mail was dominated by transportation—the price of a letter depended directly the distance to the destination [61]—and there was often only single cost-effective way of routing mail. Routing was straightforward: mail would be sorted at the originating post office and packaged for shipment to the destination post office.

However in the modern postal system it is *processing* (sorting, cancelling, etc.) and not transportation that dominates the cost of mail: in 2004, for example,

processing costs of the United States Postal Service (USPS) were $20 billion, compared to only $5 billion for transportation [60]. This fact shapes in the design of the USPS postal network and the way it routes mail (Figure 1.1). The USPS network is highly hierarchical, a theme we will see again in other networks. At the lowest level of the hierarchy are local post offices which both collect incoming mail and do final local delivery. Local post offices send their first-class mail for processing to a regional Processing and Distribution Center (P&DC). Local post offices do no processing of incoming mail; all processing takes place at the P&DC. At the highest level are the Air Mail Centers and Hub and Spoke Program centers, although, as shown, mail may also be taken to another P&DC directly [60]. The switch to centralized processing—in the past, mail was cancelled and sorted locally—was motivated by advances in automatic mail processing which significantly lowered costs [44]. Taking advantage of automation required large central facilities rather than local processing, which necessitated the transition to the highly centralized postal network we find today.

1.1.2 The Telegraph Network

The ancient precursors to the "modern" electrical telegraph were signal fires whose origins fade into myth. These were improved upon (much later) by the semaphore telegraph, which used movable displays that could be seen from several miles away with the aid of a telescope. One of the best known such systems, the Chappé telegraph, dates back to 1793. It consisted of two mechanical arms mounted on a movable beam which would be arranged in various positions. Messages were sent using a code which would translate words or phrases into positions [55]. Chappé system telegraph lines were made up of two kinds of stations. The first kind were simple repeaters in which the operator, who usually did not even know the code, simply mimicked the signals he saw. Such stations are the direct analogs repeaters found in digital networks today. The second kind of stations were called divisional points; they correspond to routers in modern networks: messages were decoded, copied to a record book, and re-encoded for transmission along the correct line.

At its peak circa 1846, the Chappé visual telegraph connected Paris to major cities in France. Moreover, the network had branches and multiple routes to a

destination, so that a message from Paris to Marseille could be sent via the Dijon or Bordeaux branch, which allowed problems on a line to be bypassed [63]—a precursor of the network routing mechanism we have today.

The electrical telegraph began to emerge at around the same time, with the best known example of which was Morse's telegraph. Whereas most of the mechanical telegraphs were run by governments and were not accessible to the public, access to electrical telegraphs was much broader, ushering in a new era of commercial telegraphy. By 1853, about a hundred telegraph lines were in operation in the the United States; Figure 1.2 shows a fragment of a telegraph map of North America.

A telegraph line in the Morse system was literally an electrical line carrying current. In its normal state the line was energized; a mark was made by interrupting the current. Several stations could be placed on a line, each being able to receive signals by observing break in current using an electromagnet and send signals by interrupting the current [55]. Thus, a telegraph line constituted a kind of broadcast medium akin to the original shared-medium Ethernet.

Message Switching

Messages could also be sent between lines not connected together electrically. This required that messages be transfered between lines manually. The message would be received by the station on the originating line and delivered on paper for transmission on the next line. If the next line was operated by another company, the message would be delivered together with payment for transmission along the remainder of the route. In an early treatise on the telegraph, the author described this arrangement:

> Lines occupying the same building have facilities in matters of accounts and the transfer of messages from line to line. In former years, when rivalry was at its highest, the companies would deliver the message and the money to the next in course, in the same manner as the public. No accommodation, no favor of any kind, nor any association between the agents of the companies, was entertained. [55, p. 761]

What's more, the sender of a message had explicit control over the routing of a message. By law, "no company can refuse to transmit a message offered, and in

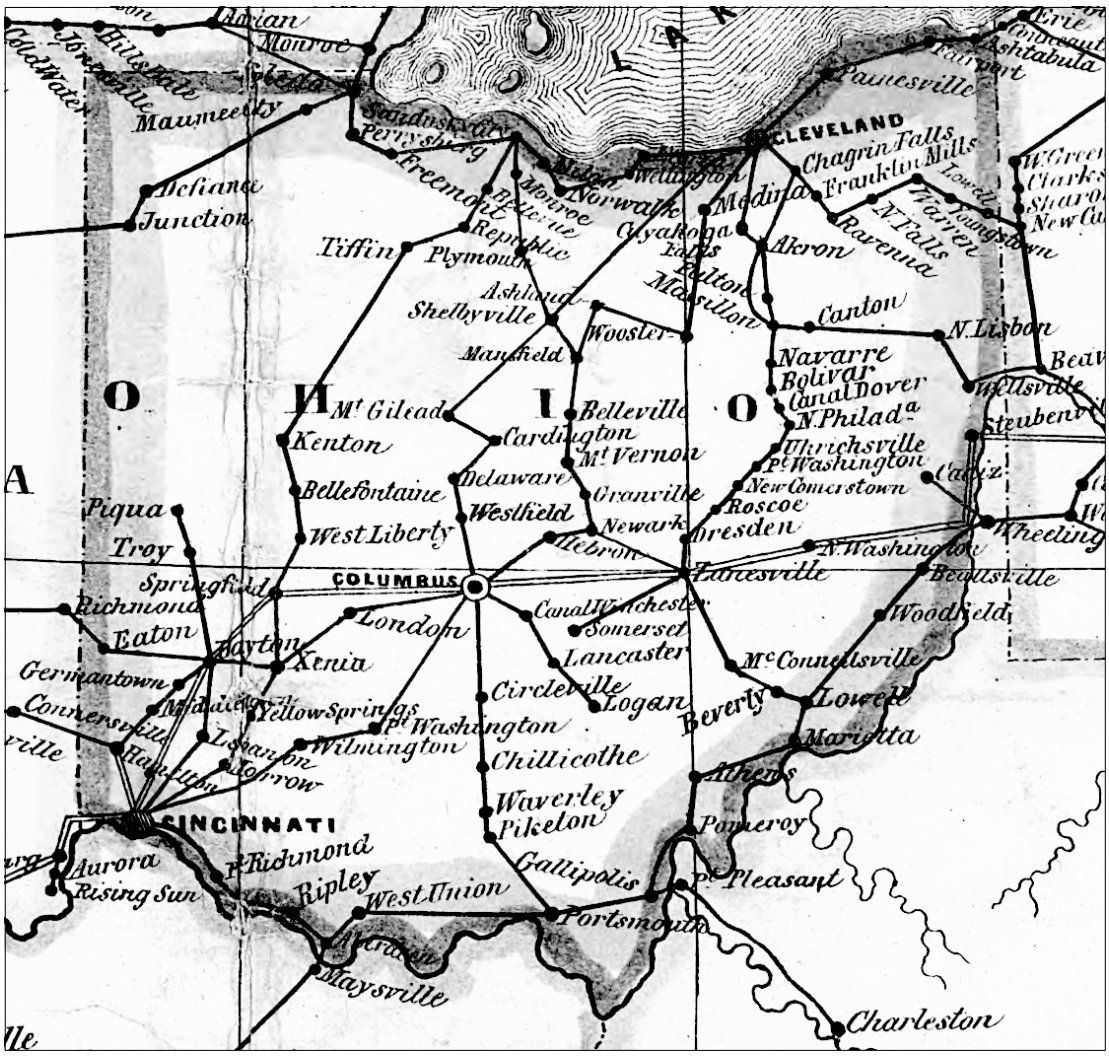


Figure 1.2: Fragment of the North America telegraph network showing the state of Ohio circa 1853 [5].

such manner as directed by the sender." [55, p. 765] The only exception was when a company had a patent monopoly between two cities, giving the exclusive right to transmit messages between the two cities, in which case such messages must be transmitted on the company's line. If the sender does not specify a route, a company may route the message in a manner of its choosing. Note the similarity of this mechanism to *source routing* in IP (Internet Protocol) networks.

Circuit Switching

It is also possible to route messages between different lines by physically connecting the lines, putting them on the same circuit. This is the idea behind the *telegraph exchange*, where it was use to connect private lines, such as those of banks, rather than long-distance telegraph lines of a company. The idea seems to have been invented as early as 1850, although exchanges did not come into use until the late 1860's [31]. It is experience with telegraph exchanges jump-started the development of telephone exchanges, which were perhaps the most important development in the history of the telephone.

1.1.3 The Telephone Network

The invention of the telephone is widely credited to Alexander Graham Bell, whose patent on the telephone was granted on March 7, 1876. It was not, however, until the introduction of the *telephone exchange* two years later that the telephone's potential was realized. In fact, it was Bell's competitor, the Western Union telegraph company, that opened the first major exchange:

> When the Western Union exchange opened in San Francisco, Bell's activities were confined solely to selling and installing private line service. The Bell company appears to have had its hands full providing just this type of service, and dealing with rapid growth. There were no active plans to offer exchange service in larger cities. [57]

The first telephone exchanges were based on a manually-operated switchboard. In a 1879 patent for an automatic switching system, the inventors M. D. Connolly, T. A. Connolly, and T. J. McTighe described the state of the art thus:

> Under the present system in use in the principal cities having telephonic facilities the lines from the several stations converge to a central office and terminate in a switch-board. When an individual member of the exchange desires to communicate with any other member he signals the central office, states his desires, and an attendant thereupon makes the desired connection. The operation of making these connections is now altogether a manual work, and requires not only constant attention but much dexterity in order that there shall be as little delay as possible; but in exchanges comprising many members the work of of the central

> office is very great, requiring many employés to meet the wants of the community. Even then, there are incessant delays, much confusion, and consequently many mistakes and annoyances which it is highly important should be obviated. [16]

The modern telephone network is a fully automated system. What it inherits from the early switchboard is its *circuit-switched* nature. When the caller dials number, he is connected to the called party through a number of switches which provide a *circuit* between the callers that lasts the duration of the phone call as if connected electrically by a switchboard. In today's telephone network, there is no actual electrical continuity between the end points: the electrical signal is multiplexed with other signals using a number of techniques (time or frequency division multiplexing and more recently digital packetization) for more efficient switching and transmission.

Network Design and Routing

Until recently, the telephone network was highly hierarchical in nature. Individual customer phone lines terminate at a *local office*, where they are aggregated into *trunks*—links which can carry many phone calls simultaneously. In the AT&T network [51], each local office is connected to a *toll center*. The next level of the hierarchy consists of primary centers, then sectional centers, and finally regional centers, which are fully interconnected (Figure 1.3 on the following page; note the resembalce to the postal network shown in Figure 1.1 on page 2). Trunks connecting levels in the hiearchy are called *final trunks*. In the absence of other links, a call would be routed up the hierarchy from the caller's central office to it's toll center and then to it's primary center, and so on, up to the lowest center from which it can be routed down to the called party's central office. Local offices and switching centers may also be connected by *high-usage* trunks which allow a call to be routed more directly. High-usage trunks are installed when call volume between two endpoints justifies the connection. Shown with dashed lines in the figure, they connect local offices or switching centers at the same level or one level above and below in the hierarchy.

The associated routing mechanism is called *fixed hierarchical routing* [51, Chapter 4]. Although we omit some details, fixed hierarchical routing works as follows. Calls are routed through the network one hop—local office or switching

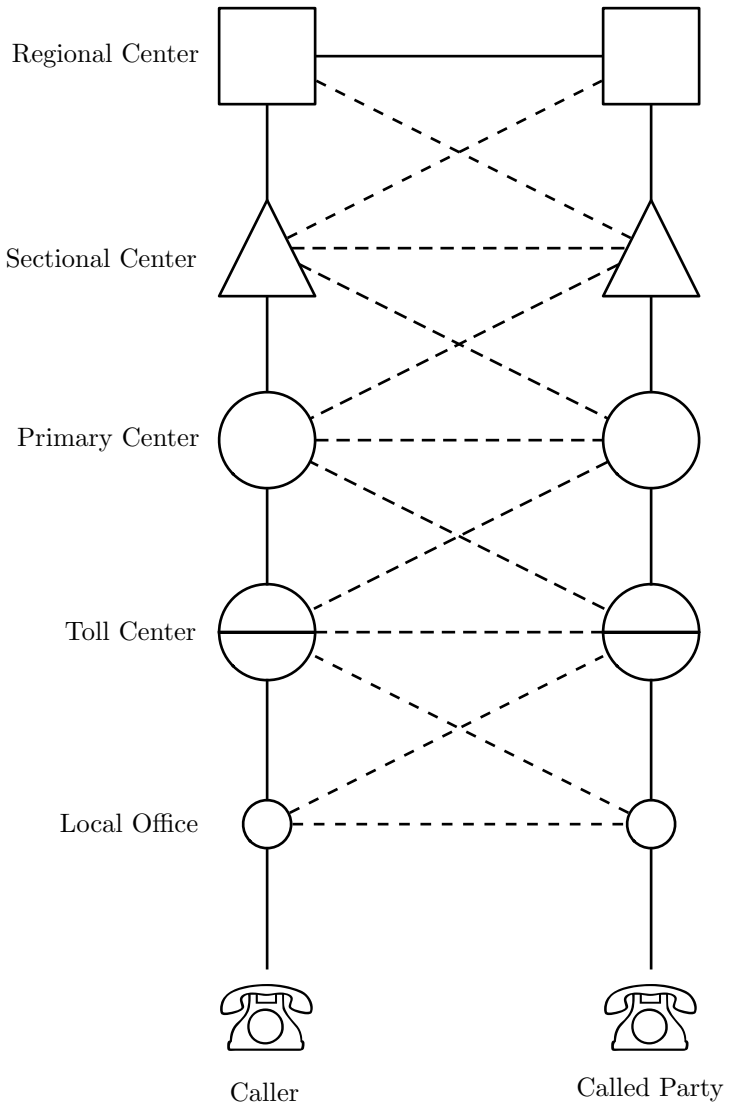


Figure 1.3: Modern telephone network hierarchy. Final trunks shown as solid lines, high-usage trunks as dashed lines. Only those high-usage trunks justified by demand would be present in an actual network.

center—at a time. At each hop, the call is first offered to a high-usage trunk that would short-cut the hierarchical route; if no high-usage trunks are available, the call is offered to the final trunk to the next higher center in the hierarchy. If the final trunk is busy, the call is rejected (or *blocked*, in telephone network terminology).

The main benefit of fixed hierarchical routing is its simplicity. Because it is based on local decisions, the switch at each hop only needs to know which of its trunks are busy in order to determine what to do with a call. Once an available trunk is found, the switch control does not need to attend to the call. The drawback of fixed hierarchical routing is that a call can be blocked by the network when, in fact, the necessary capacity exists in the network.

In the mid-eighties AT&T began transitioning to a new routing mechanism called Dynamic Nonhierarchical Routing (DNHR) [3]. The DNHR network replaces the top levels of the existing hierarchical network with a "flat" one. When a call enters the DNHR network, the originating DNHR switch (where it entered) identifies the terminating switch (where it will leave). The originating switch then tries to find a direct or indirect two-hop route to the terminating switch, trying these routes in a fixed sequence until it succeeds or blocks the call. DNHR significantly reduced blocking rates, although because of the two-hop limit it is still possible for DNHR to block a call for which there is capacity in the network. (Allowing longer paths improves efficiency slightly but has some technical disadvantages [21, Chapter 2])

Voice over IP

Voice over IP (VoIP) is an umbrella term describing mechanisms for carrying telephone calls over the global Internet or private packet-switched IP networks in contrast to the circuit-switched telephone network (see Goode [23] for an overview of VoIP). For technical and regulatory reasons carrying telephone traffic over the Internet is much cheaper than over the telephone network. Because the Internet does not provide any guarantees on bandwidth or delay, a number of mechanisms for providing improved service in the Internet have been proposed, ranging from resource reservation [65] to simply providing preferential service at routers [9]. None of these are widely deployed in the Internet, however; mass-market VoIP services are used on the Internet today without resource reservation of differentiated services. In fact,

recent studies show that VoIP services in well-provisioned ISPs provide adequate performance [8, 38], with routing instability being the main main cause of quality degradation [10].

1.2 The Internet

The Internet began as the ARPANET, an experimental packet-switched network sponsored by the United States Department of Defense Advanced Research Projects Agency (ARPA). In their 1978 project completion report, the authors write:

> This ARPA program has created no less than a revolution in computer technology and has been one of the most successful projects ever undertaken by ARPA. ... Just as the telephone, the telegraph, and the printing press had far-reaching effects on human intercommunication, the widespread utilization of computer networks which has been catalyzed by the ARPANET project represents a similarly far-reaching change in the use of computers by mankind. The full impact of the technical changes set in motion by this project may not be understood for many years. [24]

Although the ARPANET was not the first packet-switched network, it was arguably the most influential, acting as the test bed for both of the two dominant routing algorithms in use today. We start with an overview of packet switching and routing in such networks.

Packet Switching vs. Circuit Switching

The ARPANET is what is called a *packet-switched* network, meaning that the network provides the ability to send and receive discrete units of data—called *packets*—which are routed independently by the network. At the time it was developed, the dominant communication network paradigm was *circuit switching*, exemplified by the telephone and Telex [6] networks. Packet switching differed from circuit switching in the following important ways:

Interface to the user. In a circuit-switched network, a user must first establish a connection in order to communicate with another user. In a packet-switched network, a user can send discrete units of data, called *packets*, to any other

user without first establishing a connection. It is possible, of course, to simulate a packet interface on a circuit-switched network by establishing a connection for each packet, and, conversely, to provide a circuit-like interface to a packet-switched network using a connection-oriented protocol like TCP, but this inevitably incurs some overhead.

Switching mechanism. In a packet-switched network each packet is processed independently by each node along its path through the network, while in a circuit-switched network, each node maintains some state information so that once a circuit is established, no additional control processing is required by the node. The downside is that a failure along the path breaks the connection, where in a packet-switched network packets would be routed along a different path transparently.

Service guarantees. The most significant difference between packet and switching is in the implicit guarantees the network makes to the user. Establishing a circuit causes the network to reserve resources and implicitly guarantee a certain level of service. A modern telephone circuit, for example, guarantees a certain bandwidth suitable for voice signals (200Hz to 3.5kHz for commercially acceptable quality [51]). A packet-switched network such as the ARPANET provides only *best-effort* service, meaning that packets may be dropped or re-ordered in the event of network failure or congestion.

At the time, packet and circuit switching where viewed a opposing paradigms, with circuit-switching being the "proven and accepted technique" among communication engineers, so that "it remained for outsiders to the communicaton industry, computer professionals, to develop packet switching in response to a problem for which they needed a better answer: communicating data to and from computers" [52]. Today's packet switched networks incorporate elements of the circuit switching paradigm via efforts to support resource reservation [65]. Nevertheless, the Internet still essentially adheres to the "best effort" service paradigm as reflected by its routing mechanism.

1.2.1 Routing

An Internet network node–called a *router*–determines the next hop along a packet's route using a *forwarding table*, which specifies to which of the node's neighbors to forward the packet. This next hop is based only on the destination address of the packet. The simplicity of this mechanism has made it possible to do very fast switching; using special address prefix lookup algorithms (e.g. [62]) has allowed routers to forward packets at gigabit speeds.

For forwarding to work, forwarding tables must be configured so that taken together they form correct forwarding paths. In small, reliable networks forwarding tables can easily be configured and updated manually. However one of the central premises of packet-switched networks is that networks can be built using unreliable hardware [4]. Such networks must continue to provide service by adapting to failures automatically. The Internet today is in a constant state of change: links and routers occasionally fail, some are decommissioned, while others are added; the current rate of change is well beyond our ability to manage forwarding tables manually.

One possibility, used in the SITA network [14], a packet-switched network pre-dating the ARPANET, is to pre-compute alternate routes to each destination, so that when a link or node fails, forwarding tables can be switched automatically. Routing computation in the SITA was done centrally based on established link usage priorities. This was quite reasonable for the SITA network which at the time had only eight nodes. Unfortunately, as the network grows it becomes infeasible to pre-compute all routes.

An explicit ARPANET design decision was to make routing completely distributed *without* a central routing processor or pre-computed routes. Each router—called an Interface Message Processor (IMP) in the ARPANET—was responsible for computing its own forwarding table using information available to it directly about its links as well as information learned from its neighbors. In addition to maintaining forwarding-level connectivity, the routing mechanism also attempted to minimize connectivity. This was done by assigning each link a *weight* or *cost* based partly on the current packet queue length at the link; packets were then routed along minimum-cost path, taking the total cost of a path to be the sum of the costs of its edges [39]. This neatly reduced the routing problem to the well-studied problem of

finding a shortest path in a graph.

Unfortunately the above mechanism, specifically the use of queue lengths as the link metric, was vulnerable to developing a feedback loop under high load. While subsequent changes to the original protocol attempted to mitigate this behavior by changing how link costs are calculated [30, 40], load-based link metrics were ultimately abandoned. What survived of that approach was the formulation of routing as a shortest path problem. Today Internet routing is based on fixed link costs where a link either has a finite, statically-assigned cost if the link is up, or a cost of "infinity" if it's down [42, 50]. Moreover, it was even shown recently that traditional traffic engineering (centralized routing and management based on known traffic patterns) can be done by carefully setting link weights to drive the path-selection process [17].

Throughout the rest of this work, we assume that the problem of routing is that of finding shortest (equivalently, minimum-cost) paths. Because routing is distributed, each node is only aware of the costs of its own links and must communicate with its neighbors to learn about the rest of the network. In the context of this work, a *routing algorithm* is a distributed algorithm for computing the forwarding tables of nodes such that packets are sent along shortest paths; we define this more formally the next chapter.

Distance Vector Algorithms

In the first routing algorithm developed for the ARPANET [39], each node computed distances to each destination using a distributed Bellman-Ford algorithm. It became the basis of a number of routing protocols, including RIP [37], Cisco's proprietary IGRP [53], DUAL [19], and more recently AODV [48]. In the main iterative step of the algorithm, a node sends to its neighbors an estimate of its distance to each node, the so-called *distance vector*. Using the distance estimates received from neighbors, it then updates its own estimate by choosing for each destination the neighbor that minimizes the path cost. The path cost via a given neighbor is the sum of the edge cost to the neighbor (a quantity known to the node directly) and the distance to the destination reported by that neighbor. The neighbor giving the shortest path to destination is then set as the next hop to that destination.

The distance vector algorithm as described above has a major flaw, and that

is that when a node becomes disconnected, stale information remains in the network potentially causing a permanent routing loop while distance estimates computed by the algorithm continually increase, a phenomenon called "counting-to-infinity." (More generally, Jaffe and Moss [28] showed that long-term routing loops can form when a cost increase occurs, but never after a decrease.)

The simplest "fix," implemented in the ARPANET algorithm and RIP, is to bound the maximum distance, so that when a distance estimate exceeds this value, the destination is correctly declared unreachable. As a network becomes larger, however, this so-called "infinity metric" must increase, thereby increasing the time it takes for the loop to resolve. Partly for this reason routing protocols based on this variant of the distance vector algorithm are rarely use in practice.

Another approach is to have neighbors share their shortest-path trees rather than distance estimates only [13, 25, 49, 7]. This is usually done by including an additional piece of information with each distance estimate, a "parent pointer" giving the parent of the destination node in the shortest-path tree of the advertising node. Knowing the shortest-path tree of its neighbor allows a node to explicitly exclude paths that would result in a loop when computing its own distance to the destination. These SPT-based algorithms are generally more conservative in accepting a new path, thus being slower to accept a new path but minimizing the duration of transient loops, a phenomenon we will observe experimentally in Chapter 4.

Link State Algorithms

The second ARPANET algorithm [40] was designed to address the shortcomings of the first algorithm, namely its long convergence time. This family of algorithms came to be known as *link state* algorithms because nodes update each other on the state of links. OSPF [43] and IS-IS [45], the dominant routing protocols in use today, are based on the link state algorithm. In the link state algorithm, each node maintains a table giving the current cost of all links in the network, which allows it to compute its own shortest-path tree explicitly. The next hop is then chosen using the computed shortest-path tree as the next node on the path (in the tree) to the destination in question. The link state algorithm is simple and adapts quickly to network changes. Unfortunately, it requires that *every* change be broadcast to every

node, a quality that makes the algorithm fundamentally *unscalable*, as we argue next.

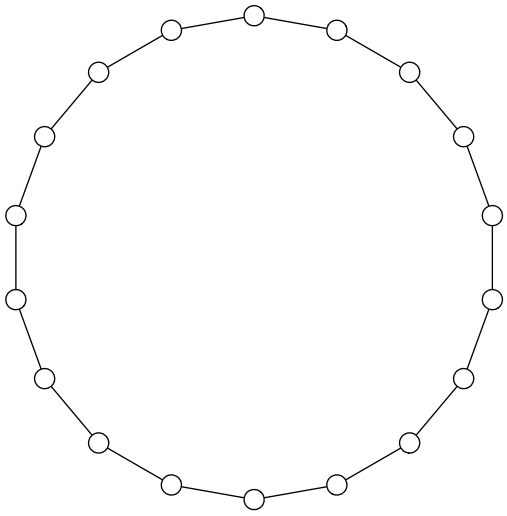
1.2.2 Scalability

The routing algorithms described above share a common characteristic, namely that network changes cause an update to be *flooded* throughout the rest of the network, either as a distributed Bellman-Ford computation or as an explicit link state update. In networking, scalability refers to the ability of a system to accommodate growth. However as a network grows, the requirement to universally communicate and act on each topology change can become problematic. This is because a larger network also generates routing updates more often (assuming the likelihood of an individual link failure is independent of overall network size), necessitating more frequent routing updates and route re-computation. Worse yet, these costs are incurred by every router in the network, meaning that the most resource-constrained router effectively determines the maximum network size that can be served by a routing algorithm. Thus, these routing protocols are frequently said to "not scale well."

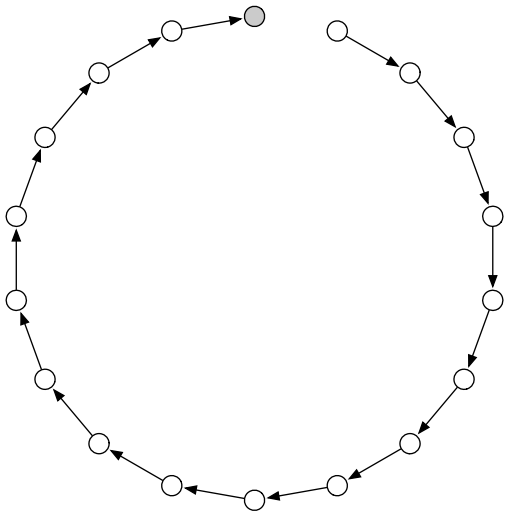
To address this problem, both OSPF and IS-IS allow the network to be divided into *areas*. Each area acts as a separate network which connects to other areas via a set of *border routers*. These border routers limit the propagation of link state updates outside the area. Instead, area routers report cost summaries (essentially distance vectors) to destinations inside the area.

The hierarchy imposed by areas is completely artificial: areas do not delineate policy regions but rather serve as a routing algorithm optimization; Cisco's OSPF Design Guide [15], for example, states, "Areas are introduced to put a boundary on the explosion of link-state updates."

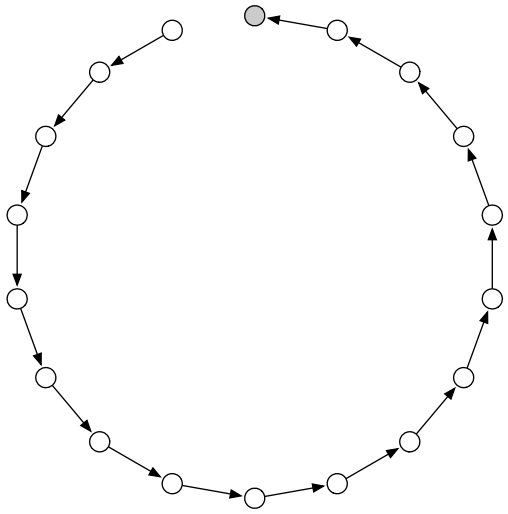
Furthermore the process of properly configuring and maintaining areas is a complex art form; one with ad-hoc rules of thumb ("no more than 50 routers per area") and complex design trade-offs. Indeed, the structure imposed by areas inherently limits the kinds of topologies that can be mapped onto routes and, if not carefully managed, can produce arbitrarily sub-optimal routes and unnecessary points of failure [59]. This dissertation is focused on minimizing or removing the *need* for such artificial hierarchy by improving the communication efficiency of the underlying



(a) Initial configuration



(b) Right link failure



(c) Left link failure

Figure 1.4: A worst-case link failure scenario that requires ever node to be updated. The initial state of the network is shown in (a). The failure of a single link forces the forwarding network to forward packets clockwise to the shaded node (b). When the link is restored and the node's other link fails, the forwarding network must forward counter-clockwise (c). Each alternating failure of the node's links requires a complete network update.

routing protocols.

Asymptotic Behavior

A natural question is whether it is inherently necessary to flood updates to the whole network or whether it is possible to notify only *some* nodes in the network. Unfortunately, in pathological cases, the answer is "No." Figure 1.4 shows one such network, a ring (cycle), which requires a complete update when a node's link fail in alternation. Referring to figure, when the shaded node's right link fails, forwarding to the shaded node must be clockwise. When the right link is restored and the left link fails, the entire network must be re-configured to forward counter-clockwise. Each alternating failure thus requires all nodes to update their forwarding tables. Fortunately, in real-world networks it is often possible to avoid complete flooding. However establishing meaningful upper and lower bounds requires both a network and a link change model. Developing models that are of practical interest and yet analytically tractable is a challenging open problem. We have instead opted to evaluate our algorithm empirically by comparing our algorithm to existing algorithms in simulation (Chapter 4).

Our problem should not be confused with *compact routing*, the problem of designing addressing and forwarding schemes that minimize the *memory* requirements of each node [11, 12, 18, 20, 47]. Compact routing differs from our problem in that for us, the addressing and forwarding mechanism (via forwarding table lookup) are fixed. Instead, we seek to minimize the *communication* cost of adapting to changes in the network, rather than the spacial or computational complexity of forwarding.

Acknowledgements

This chapter contains work previously presented at the 2008 ACM SIGCOMM Conference and appearing in its proceedings, which is joint work with Ramamohan Paturi, Geoffrey M. Voelker, and Stefan Savage [32].

Chapter 2

XL: A Communication-efficient Routing Algorithm

In this chapter we describe our routing algorithm, called XL (Approximate Link state), which explicitly aims to reduce the number of updates generated as a result of a change in the network. We give a complete formal description and prove the correctness of the algorithm.

XL is fundamentally a link-state routing algorithm. It differs from the standard link-state algorithm in propagating only *some* link state updates. The core of the algorithm consists three rules describing when an update should be propagated; our main technical contribution is showing that these are *sufficient* for correctness. These conditions are:

S1 When the update is a cost increase (bad news),

S2 When the link is used in the node's shortest-path tree (propagated only to the next hop to the link), and

C1 When it improves the cost to any destination by more than a $1+\epsilon$ cost factor, where ϵ is a design parameter of the algorithm.

Any updates not covered by the three rules above may be suppressed. The intuition behind these rules is that S1 and S2 ensure that each node's estimate of the distance to a destination decreases along the forwarding path, which ensures that no loops are formed. Rule C1 ensures that all nodes know about some good (not but necessarily

optimal) paths. Before formally describing the XL algorithm, we need to define our network model.

2.1 Formal Definitions

The subject of this dissertation is the design of a communication-efficient routing protocol for a dynamic, destination-based forwarding network such as the Internet. A *routing protocol* is a mechanism by which network nodes can coordinate packet forwarding to ensure any two nodes in the network can communicate while optimizing some objective function such as cost. In a *destination-based forwarding* network, forwarding is based only on the packet destination address. A node's forwarding decision is made using a *forwarding table* which gives the next hop to each destination or indicates that the destination is unreachable. The objective of a routing protocol is a network configuration in which all nodes are mutually reachable (provided the network is connected) and forwarding paths near-optimal according to the objective function. In this work, the objective is to minimize the *cost* of each path, where the cost is defined as the sum of the costs of links in the path.

A routing protocol operates on *views*—representations of the network state—from which the forwarding tables are computed. In this section, we define these terms formally which lays the groundwork for the design and analysis of routing protocols.

2.1.1 Network

We model a communication network as a graph $G = (V, E, e)$ with vertex set V, edge set E, and edge weight function e. The vertices represent network nodes, edges represent links, and edge weight represent link costs. Throughout the paper, we will use pairs of terms node and vertex, link and edge, interchangeably.

To simplify exposition the set of nodes and edges is fixed and globally known, and only the edge weight function may vary with time. Extending a protocol in our model to allow for vertex or edge insertions and deletions is straightforward.

The range of the weight function is the set of non-negative real numbers together with the special value ∞ having the expected semantics. We use $N(u)$ to denote the set of neighbors of a node $u \in V$. The set of edges E is undirected,

meaning that $(u, v) \in E \Rightarrow (v, u) \in E$, although the weight function e is *not* directed, meaning that $e(u, v)$ is not necessarily equal to $e(v, u)$. Let $\delta(u, w)$ denote the weight of a minimum-weight path from u to w, or ∞ if no such path exists; $\delta(u, w)$ is the lowest possible cost of forwarding a packet from u to w.

We use a superscript to denote the time at which the value of a function or variable is considered. For example, $\delta^t(u, w)$ denotes the weight of a minimum-weight path in G at time t. The domain of t is the set of non-negative reals.

A path is a sequence of nodes of which any consecutive pair is adjacent in the graph. The *length* of a path α, denoted $|\alpha|$, is the number of edges in the path. The *weight* of a path α in G, denoted $\|\alpha\|$ is sum of the weights (given by the weight function e) of its edges. As with other time-dependent values, $\|\alpha\|^t$ denotes the weight of α at time t.

2.1.2 Forwarding

To each node u in the graph we associate a *forwarding table* f_u which maps a destination node w to a neighbor of u, with the semantics that a packet arriving at u destined for w will be sent to the neighbor of u given by the forwarding table. In addition to forwarding to one of its neighbors, a node may also forward to no one, if for example, it has reached its destination. In other words,

$$f_u(w) \in N(u) \cup \{\text{NONE}\}, \tag{2.1}$$

where $N(u)$ are the neighbors of u.

The *configuration* of the forwarding network at some instant in time is the set of forwarding tables of its nodes. A natural objective in a forwarding network is a configuration in which all nodes are reachable via packet forwarding. To capture the iterative nature of packet forwarding, we consider the path taken by a packet in the network. The *(instanteneous) forwarding path from u to w*, denoted $\phi(u, w)$, is the successive application of f to w, starting at u, up until NONE. Formally, $\phi(u, w)$

is maximum-length sequence satisfying

$$\phi_0(u, w) = u \tag{2.2}$$

$$\phi_{i+1}(u, w) = f_{\phi_i(u,w)}(w) \tag{2.3}$$

$$\phi_{i+1}(u, w) \neq \text{NONE}. \tag{2.4}$$

Note that $\phi(u, w)$ may be an infinite sequence, if, for example, $f_u(w) = v$ and $f_v(w) = u$, resulting in a *forwarding loop*. If $\phi(u, w)$ is a finite path from u to w, we say that w is *reachable by forwarding* from u.

2.1.3 Soundness and Completeness

To each node we associate a *routing process* responsible for computing the forwarding table of the node. The routing process knows (or measures directly) the costs of incident links and communicates with its neighbors via these links. A *routing algorithm* is the mechanism that defines what information is exchanged with neighbors and how the forwarding tables are computed. The central purpose of a routing algorithm is to maintain a forwarding configuration in which nodes are mutually reachable by forwarding. It is often also desirable for the paths taken by forwarded packets to be optimal or near-optimal. We formalize these objectives using the notions of *soundness*, *completeness* and *stretch*.

Definition. A configuration is *sound* if for all nodes u and w, $f_u(w) \neq \text{NONE}$ implies $\phi(u, w)$ is a path from u to w. A routing algorithm is *sound* if it produces a sound configuration after the network becomes quiet.

In a nutshell, soundness says that a node should only attempt to forward to destinations it can reach by forwarding. We will show that the XL routing algorithm we describe in this paper has this property. There is also a weaker property that is sufficient for many applications, and it is simply that there be no forwarding loops:

Definition. A configuration is *loop-free* if for all u and w, $\phi(u, w)$ is finite. A routing algorithm is *loop-free* if it produces a loop-free configuration after the network becomes quiet.

The difference between a sound and a loop-free configuration is that in the latter, a node only needs to know that forwarding to its next hop will not cause a loop (but the packet could be dropped somewhere down the path), while in a sound configuration, forwarding to the next hop must actually reach the destination.

The easiest way to achieve soundness is for every node to "pretend" everyone is unreachable by setting $f_u(w) = \text{NONE}$ for all destinations w. Clearly this is a degenerate configuration, so what we also want is for $f_u(w)$ to be NONE only if w really *is* unreachable from u in the network. We call this property *completeness*.

Definition. A configuration is *complete* if for all distinct u and w, $\delta(u, w) \neq \infty$ implies $f_u(w) \neq \text{NONE}$. A routing algorithm is *complete* if it produces a complete configuration after the network becomes quiet.

Together the soundness and completeness properties say that all nodes are reachable by forwarding, but they say nothing about the optimality of the forwarding paths. This is the subject of our next definition.

Definition. The *stretch* of a configuration is the maximum taken over all distinct nodes u and w of the ratio $\|\phi(u, w)\|/\delta(u, w)$, with the convention that $1/\infty$ is 0, and ∞/∞ is undefined and not included in the maximum. A routing algorithm has *stretch* $1+\epsilon$ if it produces a configuration with stretch at most $1+\epsilon$ after the network becomes quiet.

2.1.4 Routing and Computation Model

To each node we associate a *routing process* responsible for computing the forwarding table of the node. Because the process resides on the node, it can only directly determine the costs of links incident on the node; however by communicating with its neighbors it can learn about the rest of the network. A *routing algorithm* is a distributed algorithm for the routing processes to coordinate their forwarding table updates to achieve a desired global configuration (e.g., all nodes reachable by forwarding).

In a real network adjacent nodes communicate by sending messages defined by a routing protocol. The routing protocol typically defined not only the format and

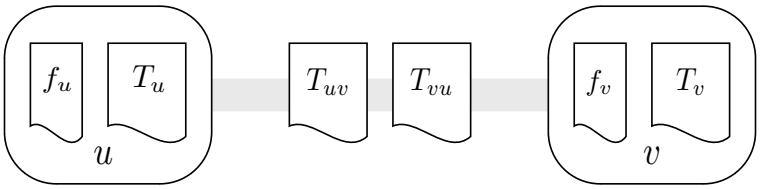


Figure 2.1: The routing process of each node maintains the forwarding table (f_u and f_v), internal view (T_u and T_v), and, for each neighbor, an external view (T_{uv} and T_{vu}). The forwarding table and internal view and private, while the external view T_{uv} can be atomically updated by u and atomically read by v, and similarly, the external view T_{vu} can be atomically updated by v and atomically read by u.

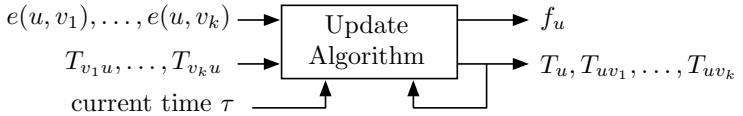


Figure 2.2: The update algorithm computes the new forwarding table, internal view, and external views. The input to the algorithm are current incident edge weights, neighbors' external views, its previous internal view and external views. The algorithm also has access to the current time.

semantics of these messages but also the routing algorithm itself. For this reason, the terms *routing protocol* and *routing algorithm* have come to be used interchangeably.

Rather than deal with message handling in out model, we instead define that a node communicates with its neighbor by updating a data structure called an *external view*. The external view is a kind of transactional one-way shared memory which can be atomically updated by a node and atomically read by the node's neighbor. We denote by T_{uv} the external view of node u that it maintains for a neighbor v. In addition to its external views, a node u also has a private *internal view* denoted T_u which is not visible to its neighbors. Figure 2.1 illustrates this arrangement. In the classical link state algorithm the external and internal views of a node are identical; in the XL routing protocol, however, these views are distinct. For a pair of nodes u and v, their external views T_{uv} and T_{vu} will normally be the same, as the algorithm attempts to maintain "consensus" of external views. Together the forwarding table, internal view, and external views thus constitute the main state of the routing process (Figure 2.1).

To simplify analysis and exposition, we describe routing processes in the form of an iterated function. The process first starts by initializing its internal view and external views to the *initial view*. It then continually updates its forwarding table, internal view, and external views using an *update algorithm* (see Fig. 2.2). Updating an external view incurs a communication cost, since the update must to be sent to corresponding neighbor. Our goal is to minimize the frequency of external view updates. To simplify analysis, we assume that external views can be updated even when the corresponding link has infinite cost. In practice, such updates would be queued until the link comes back up.

2.2 The Routing Algorithm

Formally, a view in XL is a function mapping each edge to an *edge datum*, which is simply a pair of values p and t, written $p \mathbin{@} t$, meaning that the edge had weight p at time t. Furthermore, views must only have correct information, meaning that the edge in question should have really had cost p at time t. We call this the *view invariant*. To avoid writing each definition twice, once for the internal views and once for external views, we will use the placeholder subscript $\lozenge$ to mean both u and uv. With this convention, the view invariant is:

$$T_\lozenge(x, y) = p \mathbin{@} t \;\Rightarrow\; e^t(x, y) = p. \tag{V1}$$

For convenience, let $e_\lozenge(x, y) = p$ denote the weight of (x, y) according to $T_\lozenge$, that is, if $T_\lozenge(x, y) = p \mathbin{@} t$. But note that $e_\lozenge$ is distinct from the true weight function e written with no subscript.

We say an edge datum $p \mathbin{@} t$ is *more recent* than datum $p' \mathbin{@} t'$ if $t > t'$. We will also use the terms *less recent* and *as recent* having the obvious meanings. Finally, we define a "most recent" operator "rec." Applied to a set of edge data S, $\operatorname{rec} S$ is the most recent datum in S. Formally, if there exists an edge datum $p \mathbin{@} t \in S$ that is more recent than all other $p' \mathbin{@} t' \in S$, then $\operatorname{rec} S = p \mathbin{@} t$; otherwise, $\operatorname{rec} S$ is undefined.

Let $\pi_\lozenge(z, w)$ be a minimum-cost path[1] from z to w in $T_\lozenge$. Since the underlying

[1]Ties may be broken arbitrarily, as long as the following consistency property is preserved: if $a\gamma b$ is a subsequence of $\pi_\lozenge(z, w)$, then $\pi_\lozenge(a, b) = a\gamma b$.

graph is connected, such a path always exists, although the cost may not always have finite cost. Define $d_\Diamond(w) = \|\pi_\Diamond(u, w)\|_\Diamond$; as before, $\Diamond$ stands for both u and uv.

As described earlier, a routing algorithm is structured as an iterated state update algorithm. The process starts in the initial state defined by the initial views and then repeatedly executes the update algorithm, which updates the views and forwarding table. We start by defining the initial view.

2.2.1 Initial View

The initial view defines the initial state of the routing process, before it has determined the incident link costs or communicated with its neighbors. In other words, it serves as the "base case" for the algorithm. The initial view, both internal and external, is defined as

$$T_\Diamond(x, y) = \infty \mathbin{@} 0. \tag{2.5}$$

To satisfy the view invariant (Equation V1), we also define $e^0(x, y)$ to be ∞ for all $(x, y) \in E$.

2.2.2 Update Algorithm

For the remainder of this section, fix a node u executing the update algorithm. The XL update algorithm has three phases. In the first phase, the algorithm computes a new internal view of u and the preliminary external views for its neighbors; in the second phase, it updates the forwarding table using the new internal view; and in the last phase, it computes new external views for each neighbor. We now describe these phases. Table 2.1 summarizes the notation used in the description and analysis of the routing algorithm.

2.2.3 Phase I: Internal and Preliminary External Views

The first phase is concerned with view bookkeeping. Conceptually, we would like to have a single shared view for each pair of neighbors. However since the neighbors operate asynchronously, this would require a synchronization to ensure that the common view is updated correctly. Instead, we allow each neighbor to have

Table 2.1: Notation used in the description and analysis of the update algorithm. The symbol $\Diamond$ represents the possible subscripts u or uv in the definitions.

τ	Time at the start of the iteration (INPUT).
$\epsilon_u(w)$	Maximum allowed relative error for destination w with respect to u (ALGORITHM PARAMETER).
T'_u, T'_{uv}	The internal view and external view for $v \in N(u)$, respectively, computed in the last iteration of the update algorithm, or, during the first iteration, the initial internal and external views (INPUT).
T_{vu}	The external view of $v \in N(u)$ (INPUT).
T_u, T_{uv}	The internal view and external view for $v \in N(u)$, respectively, currently being computed (OUTPUT).
T^*_{vu}	The preliminary external view of $v \in N(u)$ (Section 2.2.3).
f_u	The forwarding table of u, currently being computed (OUTPUT).
$e(x, y)$	Weight of edge (x, y) in G.
$e_\Diamond(x, y)$	Weight of edge (x, y) in $T_\Diamond$.
$\|\alpha\|$, $\|\alpha\|_\Diamond$	Cost of path α in G and $T_\Diamond$, respectively.
$\pi_\Diamond(z, w)$	Shortest path from z to w in $T_\Diamond$, with ties broken arbitrarily but consistently (Sections 2.2.4 and 2.2.5).
$d_\Diamond(w)$	Cost of the shortest path from u to w in $T_\Diamond$; by definition, $d_\Diamond(w) = \|\pi_\Diamond(u, w)\|_\Diamond$ (Section 2.2.5).
$D_u(w)$	Minimum distance proxy from u to w (Section 2.4).

its own version of this shared view. Neighbors keep their respective external views in agreement by only updating them with more recent information and by maintaining the invariant that a node's external view is no older than its neighbors. This ensures that the pair of views converge to the same single view. Thus first step in Phase I is to make sure the local external view is up to date with respect to the neighbor's external view for u. We call this updated view the *preliminary external* view. For each edge (x, y), the preliminary external view takes the more recent datum of the previous external view T'_{uv} and the neighbor's external view T_{vu}:

$$T^*_{uv}(x, y) = \text{rec}\left\{T'_{uv}(x, y),\, T_{vu}(x, y)\right\} \tag{2.6}$$

The preliminary external view is what the node and it's neighbor already agree on, or will agree on after the neighbor performs an update. It is the starting point for any updates the algorithm decided to communicate to its neighbor.

Next, we make the internal view the most recent information about each edge

available to u. For edges incident on u, the most recent information is available locally and is only updated if the edge weight changes. Formally, for $v \in N(u)$,

$$T_u(u,v) = \begin{cases} e^\tau(u,v) \mathbin{@} \tau & \text{if } e^\tau(u,v) \neq e'_u(u,v), \\ T'_u(u,v) & \text{otherwise,} \end{cases} \tag{2.7}$$

where "rec" is the "most recent" operator.

For all other edges, the source of the most recent information are the external views. We collect the most recent datum for each edge. For all x and y where $x \neq u$,

$$T_u(x,y) = \operatorname*{rec}_v T^*_{uv}(x,y). \tag{2.8}$$

The following lemma follows by construction.

Lemma 1. *The internal view and preliminary external view are well-defined and satisfy the view invariant.*

2.2.4 Phase II: Shortest-Path Tree and Forwarding Table

Having computed the internal view, which is the most recent information available to u about the state of the network, the update algorithm now computes a shortest-path tree using the internal view T_u and sets the forwarding table accordingly. This step is identical to the standard link-state algorithm.

Recall that $\pi_u(u,w)$ is a minimum-cost path from u to w in T_u, such that the set of all such paths from u forms a shortest-path tree. The distance from u to w in this tree is $d_u(w)$, which may be infinite if no finite-cost path exists. The forwarding table is now set according to the computed shortest-path tree: If $d_u(w) < \infty$ then set $f_u(w) = v$ where v is the next node in the path to w in the shortest-path tree; that is, where $\pi_u(u,w) = uv \cdots w$. Otherwise, if $d_u(w) = \infty$, set $f_u(w) = \text{NONE}$.

2.2.5 Phase III: External Views

In last phase, the algorithm decides whether to propagate the latest datum to each of the neighbors. That is, for each neighbor v and each edge $(x,y) \in E$, the algorithm chooses whether to set $T_{uv}(x,y) = T_u(x,y)$, thereby propagating the

new datum to v, or to set $T_{uv}(x, y)$ to $T^*_{uv}(x, y)$ suppressing the update. Recall that our goal is to bring the forwarding network into a sound and complete configuration with low stretch, as described in Section 2.1.3. We achieve these *global* objectives by enforcing the following three *local* constraints on external views.

The first two constraints, as we will soon show, guarantee soundness:

$$\forall (x, y) \in E \;\; e_{uv}(x, y) \geq e_u(x, y) \tag{S1}$$

$$\forall w \;\; \big(f_u(w) = v\big) \;\Rightarrow\; \forall (x, y) \in \pi_u(u, w) \;\; e_{uv}(x, y) = e_u(x, y) \tag{S2}$$

Constraint S1 states that we must never under-report an edge weight. This constraint ensures that in steady state all views reflect edge costs that are greater than or equal to the actual costs. Constraint S2 states that a node must advertise the latest edge cost to the neighbor v used to reach that edge. Intuitively, this constraint ensures that if v is our next hop to some destination w, then its own estimate of the distance to w will be no worse than ours, and, therefore, v will not attempt to reach w through us.

The third constraint guarantees completeness as well as bounded stretch. Before stating it, we need one more definition. Let $D_u(w)$ be a lower bound on the minimum distance from u to w in G. We show how $D_u(w)$ may be computed in Section 2.4. With these definitions in mind, the third constraint is:

$$\forall w \;\; d_{uv}(w) \leq \big(1 + \epsilon_u(w)\big) D_u(w) \;\text{ or }\; d_{uv}(u, w) = d_u(w). \tag{C1}$$

It states that distances in the external view should not be much worse than actual. The lower bound $D_u(w)$ is used as a proxy for the actual distance $\delta(u, w)$.

It is possible to satisfy all three constraints by setting $T_{uv} = T_u$, that is, by propagating all edge datum updates. The resulting algorithm would behave exactly like the standard link-state algorithm. However by updating only the edges in the external view T_{uv} necessary to satisfy the constraints above, we can can reduce routing communication. The following algorithm does this.

Satisfying Constraints S1 and S2 is straightforward: an edge must be updated if it causes S1 or S2 to fail. Constraint C1 is more complicated.[2] Call an edge *hot*,

[2]In fact, minimizing the number of edges that need to be updated to satisfy Constraint C1 is a hard problem (reduction from Set Cover).

denoted $\textsc{Hot}(x, y)$, if it lies on a path to a destination that causes Constraint C1 to fail.

$$\textsc{Hot}(x, y) = \exists w \ \big((x, y) \in \pi_u(u, w)\big) \wedge \big(d_{uv}(w) > (1 + \epsilon_u(w))D_u(w)\big).$$

Our approach is to greedily update hot edges until Constraint C1 is satisfied. The complete update procedure in given in Algorithm 1.

Algorithm 1 PHASE III.

```
1.  for all (x, y) ∈ E do
2.      T_uv(x, y) ← T*_uv(x, y)
3.      if e_uv(x, y) < e_u(x, y) then
4.          T_uv(x, y) ← T_u(x, y)
5.      end if
6.      if ((x, y) ∈ π_u(u, y)) ∧ (f_u(y) = v) then
7.          T_uv(x, y) ← T_u(x, y)
8.      end if
9.  end for
10. for all (x, y) ∈ E do
11.     if Hot(x, y) then
12.         T_uv(x, y) ← T_u(x, y)
13.     end if
14. end for
```

It remains to show that Algorithm 1 produces an external view satisfying the Soundness and Completeness constraints above.

Lemma 2. *After executing Algorithm 1 (above) the external view T_{uv} satisfies the View Invariant V1 and Constraints S1, S2, and C1.*

Proof. By inspection, for every edge (x, y), $T_{uv}(x, y)$ is assigned either $T_{uv}(x, y)$ or $T^*_{uv}(x, y)$. Therefore, the view invariant holds by Lemma 1.

Now consider the loop in lines 1 through 9; we claim that after it is executed, T_{uv} satisfies Constraints S1 and S2. It is easy to verify that lines 3–5 ensure S1 holds. Also, if $f_u(w) = v$ for some w and (x, y) is an edge in $\pi_u(u, w)$, then $f_u(y) = v$

also. This implies the assignment on line 7 was executed and $e_{uv}(x, y) = e_u(x, y)$ as required.

In lines 10 through 14 the algorithm updates edges to satisfy Constraint C1. We claim that the resulting external view indeed satisfies Constraint C1. First, note that after lines 1 through 9, the distance $d_{uv}(w)$ cannot increase, because $e_{uv}(x, y) \geq e_u(x, y)$ per Constraint S1. Now consider, toward a contradiction, a node w such that $d_{uv}(w) > (1 + \epsilon_u(w))D_u(w)$ and $d_{uv}(w) \neq d_u(w)$. The latter implies that there must be an edge (x, y) in $\pi_u(u, w)$ where $e_u(x, y) < e_{uv}(x, y)$. But then line 12 would have been executed for edge (x, y), and $e_{uv}(x, y) = e_u(x, y)$, a contradiction. □

2.3 Analysis

We now show that Constraints S1 and S2 produce a sound forwarding network configuration and Constraint C1 produces a complete configuration with bounded stretch. For the analysis, we assume that each execution of the update algorithm takes a bounded amount of time; let Δ be this duration. We will also need the following definition.

An edge (or set of edges) is *coherent* at a point in time if its associated external views are the same at that point in time. That is, an edge (u, v) is coherent at time t if $T^t_{uv} = T^t_{vu}$. Also, recall that a set of edges is *quiet* during a time interval if their weights do not change during the time interval.

Together the following two lemmas bound the cost of the forwarding path from u to w by $1 + \epsilon$ times the cost of the optimal path.

Lemma 3. *Fix a time $t > \Delta$. If $\phi^t(u, w)$ is a non-empty path that is both quiet during time interval $[t - \Delta, t]$ and coherent at time t, then $\phi^t(u, w)$ is a finite path from u to w and $\|\phi^t(u, w)\|^t \leq d^t_u(w)$.*

Proof. Consider the state of the network at the fixed time t. For notational simplicity, we will omit the temporal superscript t. To prove the lemma, we first show that $\phi(u, w)$ is finite, and then show that its last element is w. We then use this fact to prove the bound. We start with two observations.

Observation 1. At time t the path $\phi(u, w)$ has been quiet for duration at least Δ, so the update algorithm has been executed at least once by each node along the path $\phi(u, w)$ during the quiet interval $[t - \Delta, t]$. By Equation 2.7, $e_x(x, y) = e(x, y)$ for each edge (x, y) in $\phi(u, w)$.

Observation 2. The distance estimate $d_u(w)$ must be finite; otherwise $f_u(w) =$ NONE, implying $\phi(u, w)$ is the empty path.

To show that $\phi(u, w)$ is finite, it is sufficient to show that the estimated distance $d_z(w)$ decreases by an edge cost at each node along the path $\phi(u, w)$. Without loss of generality, consider the first edge $(u, f_u(w))$. Let $v = f_u(w)$ and let $\pi_u(u, w) = uv\alpha$, where α is some sub-path. Then:

$$\begin{aligned}
d_u(w) &= e_u(u, v) + \|v\alpha\|_u \\
&= e(u, v) + \|v\alpha\|_u && \text{by Obs. 1} \\
&= e(u, v) + \|v\alpha\|_{uv} && \text{by Constr. S2} \\
&= e(u, v) + \|v\alpha\|_{vu} && \text{by Coherence} \\
&\geq e(u, v) + \|v\alpha\|_v && \text{by Constr. S1} \\
&\geq e(u, v) + \|\pi_v(v, w)\|_v && \text{by opt. of } \pi_v(v, w) \\
&= e(u, v) + d_v(w). && (\star)
\end{aligned}$$

Thus $\phi(u, w)$ is finite. Now let w' be the last node in $\phi(u, w)$. We claim that $d_{w'}(w) = 0$ and therefore $w' = w$. By Observation 2, $d_{w'}(w) \leq d_u(w) < \infty$. But if $d_{w'}(w) \neq 0$ then by definition $f_{w'}(w) \neq$ NONE, contradicting w' being the last node.

It remains to show that $\|\phi(u, w)\| \leq d_u(w)$. The proof is by induction on the length of $\phi(u, w)$. The base case is length 1 which implies

$$\|\phi(u, w)\| = e(u, w) = e_u(u, w) = d_u(w),$$

as desired. Now consider $\phi(u, w)$ and assume $\|\phi(v, w)\| \leq d_v(w)$ where $v = f_u(w)$. Continuing from $(\star)$,

$$\begin{aligned}
d_u(w) &\geq e(u, v) + d_v(w) \\
&\geq e(u, v) + \|\phi(v, w)\| \\
&= \|\phi(u, w)\|.
\end{aligned}$$

□

Lemma 4. *Fix a time $t > \Delta$. Let β be a path from u to w. If β is (i) quiet during $[t - \Delta, t]$, and (ii) coherent at time t, then*

$$d_u^t(w) \leq (1 + \epsilon)\|\beta\|^t,$$

where $\epsilon = \max_{x \in \beta} \epsilon_x(w)$.

Proof. As in the proof of Lemma 3, consider the state of the network at the fixed time t. For notational simplicity, we will omit the temporal superscript t. Also as in that proof, we claim $e_x(x, y) = e(x, y)$ for each edge (x, y) in β.

The proof of this lemma is by induction on the length of β. If β is the empty path, then $u = w$ and we're done. Now let $\beta = uv\alpha$ for some path α, and assume $d_v(w) \leq (1 + \epsilon)\|v\alpha\|$. Then, using Coherence in step $(\star)$:

$$\begin{aligned}
d_u(w) &\leq e_u(u, v) + \|\pi_u(v, w)\|_u \\
&= e(u, v) + \|\pi_u(v, w)\|_u \\
&\leq e(u, v) + \|\pi_u(v, w)\|_{uv} \\
&\leq e(u, v) + \|\pi_{uv}(v, w)\|_{uv} \\
&= e(u, v) + \|\pi_{vu}(v, w)\|_{vu} \qquad (\star) \\
&= e(u, v) + d_{vu}(w) \\
&\leq e(u, v) + \max\left\{(1 + \epsilon_v(w))D_v(w),\, d_v(w)\right\} \\
&\leq e(u, v) + \max\left\{(1 + \epsilon)D_v(w),\, d_v(w)\right\} \\
&\leq e(u, v) + \max\left\{(1 + \epsilon)\|v\alpha\|,\, d_v(w)\right\} \\
&\leq e(u, v) + \max\left\{(1 + \epsilon)\|v\alpha\|,\, (1 + \epsilon)\|v\alpha\|\right\} \\
&\leq e(u, v) + (1 + \epsilon)\|v\alpha\| \\
&\leq (1 + \epsilon)\|\beta\|.
\end{aligned}$$

□

Both lemmas above are still conditioned on coherence. Here we show that a quiet network eventually becomes coherent, which will imply that our routing algorithm converges in finite time.

Lemma 5. *If a network is becomes quiet at some time t, then after a finite period of time it also becomes coherent.*

Proof. Divide the time line after t into epochs of duration Δ. We claim that if none of the views change during an epoch, then they will not change in subsequent epochs and the network is coherent. This is because the Update algorithm is a deterministic function of the views and edge weights, with the property that if the internal view and edge weights do not change, then the current time input is ignored (by Equation 2.7). Furthermore, from by Equations 2.6, 2.7, and 2.8 it follows that if the external views don't change, then they must be coherent.

Since an edge datum is only injected into the network in Phase I when an edge cost changes, no new edge data are injected after time t. Each view update consists of some number of edge datum values being updated to more recent values from another view. Since there is a fixed number of internal and external views in the network, each view can only be updated finitely many times. It follows that the network can only change a finite number of times after time t. But since the network must change each epoch as shown above, it will stop changing and become coherent in a finite period of time. □

We can now state our main theorem.

Theorem 1. *If a network is quiet at and after some time t, then after a finite period of time the forwarding configuration becomes sound, complete, and has bounded distortion ϵ, where*

$$\epsilon = \max_{u,w} e_u(w).$$

Proof. By combining Lemmas 3, 4, and 5. □

2.4 Minimum Distance Proxy Function

Recall that the minimum distance proxy function D_u was used instead of the actual minimum distance function δ to define the Completeness constraint (C1) in Section 2.2.5 and was also used in Algorithm 1 to compute an external view. The correctness of the XL routing algorithm requires only that $0 \leq D_u(w) \leq \delta(u, w)$ for all u and w. However to give the algorithm leeway in suppressing updates, $D_u(w)$ should be as close to $\delta(u, w)$ as possible. Computing the exact distance $\delta(u, w)$ is exactly what we're trying to avoid by using approximation, so we choose $D_u(w)$ to

be the distance computed by taking the weight of each edge to be the lowest cost of the edge ever observed. Because this value only changes when an edge cost drops below its all-time minimum cost, or an edge is added to the network, updates are infrequent and therefore introduce very little overhead to the algorithm. Furthermore, because all-time minimum link costs can only decrease, it can be computed using a distance vector-style algorithm without fear of loop formation, as shown by Jaffe and Moss [28].

A simpler alternative which does *not* guarantee globally bounded stretch is to set $D_u = d_u$. In other words, instead of computing and maintaining the cost lower bound as described above, we simply use out best estimate of the current cost from the internal view. In some cases, this will cause the stretch to exceed $1 + \epsilon$, although in practice the excess is likely to be quite small.

2.5 Cut Vertex Partitioning

Recall that in a *sound* configuration a node must only forward to a destination if the destination is reachable. This is hardly the case in the Internet today where ASes advertises prefixes, not individual destinations, even if part of the prefix is unreachable. For this reason, we introduced a weaker notion, that of a *loop-free* configuration, in which every forwarding path $\phi(u, w)$ must only be finite (loop-free) and not necessarily a path to the destination w. It means, essentially, that a node does not need to "know" that a destination is reachable before forwarding, only that forwarding to the next hop will not cause a loop. Practically, this means that sending a packet to an unreachable destination will generate an ICMP Unreachable message from a router further in the network rather than the local router.

As we have shown above, the basic XL algorithm is sound. If we relax the requirement of soundness, however, and settle for a loop-free algorithm, we can realize significant savings in routing communication using an extension to the XL routing algorithm we call Cut Vertex Partitioning (CVP).

The idea behind CVP is based on the observation that a *cut vertex* (also called an *articulation point*), which is a vertex whose removal disconnects the graph, partitions the network graph into two or more separate subnetworks that can only

communicate with each other through the cut vertex. This means that to communicate with a destination "across" a cut vertex, a node can simply forward to the cut vertex and it does not need to know about the network beyond the cut vertex. Thus with respect to routing, each subnetwork can be considered separately.

In general, real networks do not have cut vertices that partition the network into large subnetworks where CVP could be used as a "divide and conquer" technique. However, what many real networks *do* have is a large number of leaves. Since the neighbor of a leaf is necessarily a cut vertex, CVP eliminates leaves from the routing computation, effectively reducing the size of the network. In fact, our implementation of CVP only considers such leaf cuts. Our experiments (Chapter 4) show that this "reduction by a thousand cuts" significantly decreases the communication load or routing.

CVP partitions the network into subnetworks akin to OSPF *stub areas* [42, Chapter 6]. A stub area is an OSPF area which does not have external connections. However stub areas may have several routers connecting them to area 0, whereas with CVP, there can only be a single such router—the "cut vertex." On the other hand, CVP can applied at any cut vertex without regard for the special conditions imposed on OSPF stub areas. The most important distinction is that CVP is *automatic*, requiring no manual configuration or special network design considerations.

The CVP extension to the XL routing algorithm consists of the cut vertex forwarding policy described above, a mechanism for nodes to discover that they are cut vertices, and a cut vertex advertisement for nodes to learn which cut vertex to use to reach each destination. In our fixed, globally known network model where only the edge weight function changes with time, all the necessary computation can be carried out by each node separately. In practice, however, where the topology is unknown and can change, cut vertex discovery and advertisement is slightly more involved; we describe it next.

2.5.1 Cut Vertex Discovery

In the simplest case, a cut vertex is a neighbor of a degree-1 node, separating the degree-1 node from the rest of the network. This is by far the most common case

in the real-world networks we use in simulation (Chapter 4). This case is particularly easy to handle as it requires only direct negotiation between neighbors: a degree-1 node notifies its parent that it is a leaf and the parent takes appropriate action, namely suppressing network updates across their link.

Non-trivial cut vertex discovery is more involved. The approach we propose is to simply disseminate information on the presence of links so that each node may determine whether it is a cut vertex. Since the underlying topology changes much less frequently than link weights, flooding is an acceptable mechanism. Of course, this can also be done in a centralized manner by having a designated node compute the cut vertices periodically notify them of their status. Furthermore, cut vertex discovery can be combined with minimum distance proxy computation described in the previous section.

Acknowledgements

This chapter contains work previously presented at the 2008 ACM SIGCOMM Conference and appearing in its proceedings, which is joint work with Ramamohan Paturi, Geoffrey M. Voelker, and Stefan Savage [32].

Chapter 3

Forwarding Network Simulator

In this chapter we describe FNS, a discrete-event forwarding network simulation system we used to empirically evaluate the performance of our routing protocol. The simulation includes the forwarding tables and all routing protocol communication; it does *not*, however, simulate other network traffic or network characteristics such as packet loss, latency, and bandwidth.

The FNS network model closely resembles the network model defined in Chapter 2, the main difference is that nodes communicate by explicitly sending messages instead updating views in the two-party publish/subscribe model. While the two are trivially equivalent, message-passing is simpler to implement in a simulator.

FNS consists of several standalone programs of which the actual simulator is only one. A simulation experiment consists of several steps performed by a different FNS program. A typical workflow in shown in Figure 3.1.

The underlying network is described a pair of files: a topology file, defining the nodes and links of the network, and a weight file, giving the nominal link costs. The Generator program then generates the event script for the simulation, which is a sequence of edge weight changes. The Simulator program then simulates a routing protocol on the network using the event script. The output of the Simulator program is a sequence of forwarding table updates. These updates are processed by the Surveyor program, which generates a sequence of actual distance matrix updates as defined by the forwarding tables, making it possible to reconstruct the distance matrix at any point in time. The event script used by the Simulator program is also

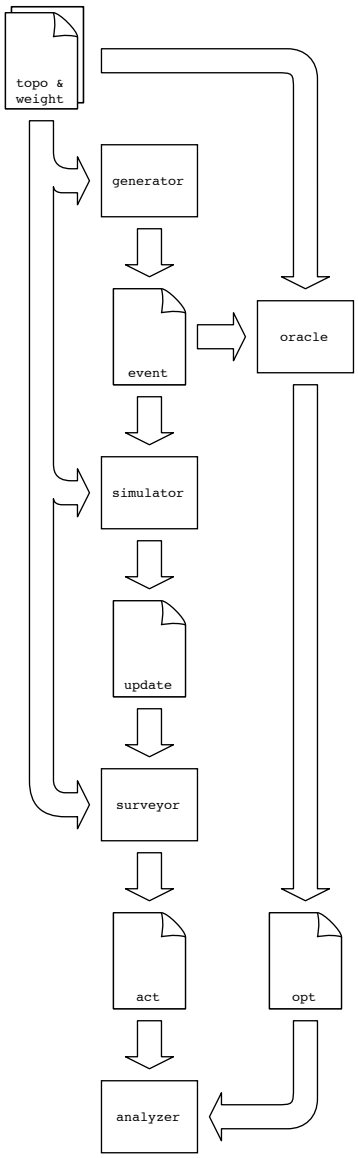


Figure 3.1: Workflow of a typical FNS experiment as described in the accompanying text.

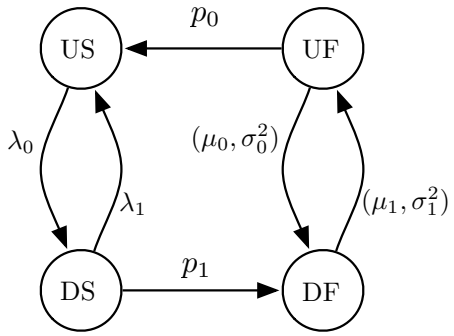


Figure 3.2: Link failure model used to generate link failure events by the Generator program. The up/stable, down/stable, up/flapping, and down/flapping states are denoted US, DS, UF, and DF, respectively.

processed by the Oracle program that generates the sequence of optimal distance table updates, giving the minimum distance between every pair of nodes at any point in time. Lastly, both actual and optimal distance table updates are processed by the Analyzer program, which computes convergence times, stretch, and other statistics.

3.1 The Generator Program

The Generator program generates a sequence of link cost changes according to a stochastic model of link failures. In the generated event sequence, a link is either *up*, in which case its cost is the nominal cost given defined by the weights file, or *down*, in which case its cost is ∞. The two directions are coordinated, that is, links (u, v) and (v, u) are either both up or both down.

Link failure and recovery is controlled by a stochastic process (Fig. 3.2) in which each link is independent. At a given instant in time, a link is in exactly one of four states: up/stable, down/stable, up/flapping, or down/flapping. In the up/stable and up/flapping states, the link is considered up and has a finite weight as defined in the `weight` file. In the down/stable and down/flapping states the link has weight ∞.

In addition to being up or down, a link is also either *stable* or *flapping*. In the stable state, the link time-to-failure is distributed exponentially with mean λ_0. Once

down, a link may remain in the down/stable state, in which case the time-to-recovery is distributed exponentially with mean λ_1, or, with probability p_1 a link may become unstable and transition to the flapping/down state. Thus, parameter p_1 controls the propensity of links to flap. In the flapping state, the time-to-recovery has a normal distribution truncated to $[0, \infty)$ with parameters μ_1 and σ_1^2, and time-to-failure has a similarly truncated normal distribution with parameters μ_0 and σ_0^2. After recovering from failure in the flapping state a link leaves the flapping state with probability p_0. Parameter p_0 thus controls how long a link remains flapping.

Our link event model is a generalization the two-state model of Park and Corson [46]; we added the flapping failure mode, which we expected the XL algorithm to handle particularly well. When $p_1 = 0$, link failures are independent with exponentially-distributed failure and recovery times. On the other hand, when $p_1 = 1$, all links have an exponentially distributed time-to-first-failure followed by repeated up-down cycles controlled by the p_0 parameter.

3.2 The Simulator Program

The Simulator program is a discrete event simulator that executes a single routing protocol under a given topology and link event sequence. In other words, it simulates n instances of the routing protocol running in parallel, one on each node. The update algorithm is repeatedly executed in the context of each node. The input to the algorithm is the simulated time and message queue of messages posted by its neighbors between the last time the algorithm is executed and current simulated time. The simulator repeatedly executes the update algorithm of each node, providing as input the (simulation) time at the start and end of the current iteration of the algorithm, the costs of incident links, and its message queue, consisting of messages sent by its neighbors since the last invocation of the update algorithm on this node. The (simulated) duration of each invocation of the update algorithm is chosen randomly according a normal distribution truncated to $[0, \infty)$ with parameters μ_Δ and σ_Δ^2.

The update algorithm also has access to the node's forwarding table, incident link weights, and any additional per-node state used by the algorithm. The update

algorithm can modify the forwarding table of the node and post messages from the node to its neighbors. The messages are considered sent at the end of iteration (a random variable as defined above) and will be available to any neighbor executing at or after this time.

Finally, the routing algorithm also has access to the complete network topology; it is only the link weights which are unknown during the simulation. An alternative would have been to have the routing algorithms discover the network topology and adapt to change in the topology itself, rather than only the link weights. However this would have significantly complicated the implementation of the algorithm. Current implementations of routing algorithms use simple static memory structures for link or node lists without the trouble of addressing and dynamic structure management.

The Simulator program contains implementations of the following five routing algorithms.

`ls` Link state algorithm (Section 1.2.1).

`dv` Distance vector algorithm (Section 1.2.1). The maximum distance bound ("infinity metric") is a global parameter of the protocol.

`dv+p` Distance vector with parent pointer algorithm [13, 25, 49].

`lv` Link vector algorithm [7].

`xl` The XL protocol introduced in this work (Chapter 2), parameterized by maximum allowed error ϵ.

The Simulator program produces the `update` file, a chronological sequence of global forwarding table updates, for subsequent processing. In addition, it also generates a history of the communication at one-second granularity as well as a summary of the total number of messages sent. These statistics are our primary measure of "communication overhead" of each protocol used in the evaluation (Chapter 4).

3.3 The Surveyor Program

The Surveyor program turns the sequence of forwarding table updates produced by the Simulator program into a sequence of actual (i.e., forwarding) distance

matrix updates. It other words, it computes the forwarding distance at every instant in time during the simulation. At time instant t, the (u, w) entry of the actual distance matrix is the value $\|\phi^t(u, w)\|^t$. If $\|\phi^t(u, w)\|^t = \infty$, then the Surveyor program also records whether $\phi^t(u, w)$, the path itself, is finite, or whether the path includes an infinite-weight edge.

3.4 The Oracle Program

The Oracle program produces a sequence of optimal distance matrix updates from the link event sequence used by the simulator. It computes the optimal distance between every pair of nodes at every instant in time: at time instant t, the (u, w) entry of the optimal distance matrix is the value $\delta^t(u, w)$.

3.5 The Analyzer Program

The Analyzer program reconstructs the actual and optimal distance matrices from the respective updates sequences produces by the Surveyor and Oracle programs. At each point in time t, it is possible to determine whether w was reachable from u, and if so, the resulting stretch, that is, $\|\phi^t(u, w)\|^t / \delta^t(u, w)$. For each such pair, the Analyzer program computes the duration of time u could not reach w even though there was a path from u to w as well as the median and highest centile (over time) stretch between u and w. It does this by stepping through each update to the actual and optimal distance matrices and computing the above information. Th maximum, average, and median values of the above statistics taken over all pairs are reported.

Acknowledgements

This chapter contains work previously presented at the 2008 ACM SIGCOMM Conference and appearing in its proceedings, which is joint work with Ramamohan Paturi, Geoffrey M. Voelker, and Stefan Savage [32].

Chapter 4

Routing Algorithm Simulation

In this chapter we describe the results of simulating our algorithm, as well as four existing routing algorithms, on a variety of synthetic and real-world networks. Our main result is a significant reduction in the number of update messages compared to existing algorithms. We also analyze the convergence times and other statistics obtained from simulation.

Our objective is to evaluate the claims that the XL routing algorithm:

- Sends fewer routing updates,
- Does not significantly sacrifice correctness, convergence time, or stretch, and
- Continues to perform well as the network grows.

Our evaluation is based on simulations of the four algorithms implemented by the `simulator` program (`ls`, `dv`, `dv+p`, `lv`, and `xl`) on a number of networks and under two different link event models. The main result of simulation is that the XL routing algorithm does indeed reduce the number of updates: compared to the link-state algorithm, XL generates between 2 and 20 times fewer updates (Table 4.3). This experiment is discussed in Section 4.2; first, however, we describe our experimental setup.

Table 4.1: Network topologies used in the experiments. Column legend: n – number of nodes; m – number of links; D_1, D_2, and D_3 fraction of nodes of degree 1, 2, and 3, respectively. All but the FUEL networks have unit link costs.

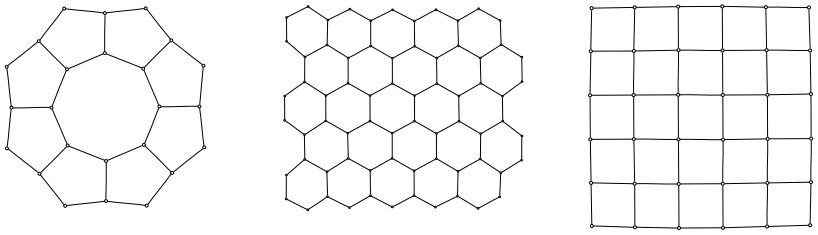
Name	n	m	D_1	D_2	D_3	Description
CROWN X	$3X$	$4X$	0	1/3	2/3	Two cycles of size X and $2X$ with nodes in the smaller connected to alternate nodes in the larger.
HONEY	—	—	0	~ 0	~ 1	A hexagonal grid.
QUAD	—	—	0	~ 0	~ 0	A rectangular grid.
ABILENE	11	14	0	45%	55%	Abilene with routing metrics [1].
ARPANET	59	72	7%	48%	41%	ARPANET (March 1977) [24].
FUEL1221	104	151	49%	19%	6%	AS 1221 from RocketFuel [36].
FUEL1239	315	972	10%	19%	16%	AS 1239 from RocketFuel [36].
FUEL1221C	50	97	0	50%	6%	The 2-core of FUEL1221.
FUEL1239C	284	941	0	22%	18%	The 2-core of FUEL1239.
ORB145	145	227	29%	28%	17%	FUEL1239 rescaled (`-n 200`).
ORB257	257	433	31%	20%	21%	FUEL1239 rescaled (`-n 300`).
ORB342	342	606	33%	24%	14%	FUEL1239 rescaled (`-n 400`).
ORB406	406	791	27%	28%	14%	FUEL1239 rescaled (`-n 500`).
ORB497	497	961	29%	26%	17%	FUEL1239 rescaled (`-n 600`).
ORB575	575	1081	31%	25%	16%	FUEL1239 rescaled (`-n 700`).
ORB664	664	1300	26%	27%	17%	FUEL1239 rescaled (`-n 800`).
ORB729	729	1427	32%	24%	16%	FUEL1239 rescaled (`-n 900`).
ORB813	813	1584	29%	25%	16%	FUEL1239 rescaled (`-n 1000`).
ORB892	892	1694	34%	26%	15%	FUEL1239 rescaled (`-n 1100`).

4.1 Experimental Setup

Each experiment consists of a number of simulation runs. Each run simulates a single routing algorithm for 86,400 seconds (one day) at a rate of 10 iterations of the update algorithm per second.

4.1.1 Networks

We used the following networks in our simulations: three synthetic networks, the Abilene backbone [1], the ARPANET topology from March 1977 [24], two Rocketfuel networks with inferred link costs [36], and a series of networks created by



CROWN 8 HONEY 5×5 QUAD 5×5

Figure 4.1: Small examples of the synthetic networks used in the experiments.

"re-scaling" the Sprint backbone (AS 1239) from the Rocketfuel dataset using Orbis [34]. Orbis is a topology generator that generates a random graph having a specified joint degree distribution. Orbis can also "re-scale" a joint degree distribution to a larger or smaller network. By extracting the degree distribution of a known network, Orbis can generate "re-scaled" version of the same network. The command-line arguments to the `dkRescale` program were "`-k 1 -n` n_{nom}", where the nominal size n_{nom} ranged from 200 to 1100. Table 4.1 describes the networks used in the experiments and Figure 4.1 shows small instances of synthetic networks. The synthetic networks allowed us to test the routing algorithms on topologies based design decisions different from the AS router-level topologies. In particular, the large-diameter HONEY and GRID networks shed some light on how the algorithms might perform in wireless ad-hoc networks.

We also created the 2-cores of the two Rocketfuel networks. The 2-core of a graph is the graph resulting from repeatedly removing all degree-1 nodes [54]. With no degree-1 nodes, CVP (which was implemented only for leaf nodes) would have no effect, allowing us to also evaluate the value of this optimization.

4.1.2 Link Events

All link events for the simulation were generated using the `generator` program (Section 3.1). Recall that in the `generator` link event model, a link is either up (nominal weight) or down (infinite weight); the time between failures and failure duration are controlled by the four-state stochastic model shown in Figure 3.2. In our simulation, we used two different sets of model parameters: a Standard set in

which a link fails about once a day, and comes back up in about an hour, and the Flapping set in which links are less likely to fail, but more likely to fail repeatedly (flap); Table 4.2 gives the precise model parameters.

Both the Standard model and Flapping model are more aggressive that what might be expected of a real network [26, 56]. We wanted to stress the routing algorithms under the kinds of conditions where routing algorithm efficiency matters greatly, namely where many links are unstable (Standard model) or only some are unstable but tend to oscillate (Flapping model).

4.1.3 Algorithm Parameters

The distance vector algorithm (`dv`) requires a maximum distance bound (the so-called "infinity metric") to detect routing loops. For the simulations, this value was computed by using a linear program to approximate the cost of the longest path. The XL routing algorithm (`xl`) has an error parameter ϵ that determines the stretch. In the experiments, we simulated `xl` with $\epsilon = 0.0$ and $\epsilon = 0.5$, corresponding to no stretch and a maximum stretch of 1.5. Increasing ϵ beyond 0.5 did not appear to significantly reduce the number of updates generated by the algorithm beyond the $\epsilon = 0.5$ level.

4.2 Performance

In this section we evaluate our first two claims: that compared to existing routing algorithms, the XL algorithm uses fewer updates to achieve comparable performance. We simulated each routing algorithm on the synthetic and measured

Table 4.2: Parameters used to generate link events according to the `generator` link event model described in Section 3.1. Mean time-to-failure is controlled by the λ_0^{-1} parameter and the probability of a repeat failure by the p_1 parameter. Units: d – days, h – hours, m – minutes, s – seconds.

	p_0	p_1	λ_0^{-1}	λ_1^{-1}	μ_0	σ_0	μ_1	σ_1
Standard	0.25	0.10	1 d	1 h	1 m	10 s	1 m	10 s
Flapping	0.25	1.00	2 d	10 s	10 s	1 s	10 s	1 s

Table 4.3: Average number of messages after initialization, relative to `ls` (average of 10 simulation runs). The `xl` columns shows values for algorithm parameters $\epsilon = 0.0$ (first value) and $\epsilon = 0.5$ (second value).

	Standard model					Flapping model				
	`dv`	`dv+p`	`lv`	`xl`		`dv`	`dv+p`	`lv`	`xl`	
Crown 64	3.13	1.11	1.10	0.64	0.41	0.85	0.82	0.82	0.45	0.11
Honey 16×16	0.95	0.69	0.65	0.31	0.18	0.28	0.65	0.60	0.20	0.06
Quad 16×16	0.12	0.40	0.39	0.14	0.10	0.06	0.38	0.37	0.07	0.04
Abilene	0.82	0.71	0.71	0.50	0.43	0.88	0.79	0.79	0.47	0.33
Arpanet	2.33	1.02	1.02	0.47	0.40	1.80	1.00	0.99	0.36	0.24
Fuel1221	7.90	0.63	0.62	0.14	0.10	7.05	0.61	0.60	0.12	0.05
Fuel1239	5.01	0.25	0.26	0.17	0.09	1.21	0.25	0.25	0.14	0.04
Fuel1221c	0.79	0.45	0.46	0.34	0.22	0.39	0.42	0.42	0.27	0.11
Fuel1239c	0.99	0.25	0.25	0.19	0.09	0.21	0.24	0.24	0.14	0.04

topologies. Each combination of algorithm, network and link event model (Standard or Flapping) was simulated 10 times and averaged in reporting results. For each combination, the 10 simulations differed only in the link events.

4.2.1 Total Communication

Table 4.3 shows the average number of messages sent during the simulation relative to `ls`, the link state algorithm—a convenient baseline for comparison.

Referring to the table, the most erratic performer was `dv`, which was highly sensitive to topology: It did extremely well on networks such as Quad 16×16 with many equal-cost paths and poorly on less well-connected networks with long cycles that trigger its "counting-to-infinity" behavior. Equal-cost paths benefitted `dv` because the failure of any single link would often not change a node's distance estimate, so that no distance update would be generated.

Both `dv+p` and `lv` performed similarly (as expected): they routinely did better than `ls` because only updates to links in a node's shortest path tree would be propagated. On the other hand, they could not take advantage of the multiple equal-cost paths as `dv` could. This is because these algorithms communicate their shortest-path trees; when a path cost changes, an update must be sent to that effect,

even if another path having equal cost is also available.

The XL algorithm performed consistently well on all networks. Like `dv`, it was able to take some advantage of path redundancy in the QUAD synthetic network because updates when a link returned to service could be suppressed (see Section 4.2.5). Like `dv+p` and `lv`, it also only propagated updates that affected its shortest-path tree. Finally, it on "leafy" networks like FUEL1221, Cut Vertex Partitioning (Section 2.5) allowed many updates to and from leaf nodes to be suppressed.

We note that XL algorithm performed particularly well in the flapping model. Why is this? The reason is that the XL algorithm tends to move away from flapping links: The first time a link fails, an update is propagated to all nodes in whose shortest-path tree the link appears, in other words, to nodes that used the link to reach some destination. When the same link comes back up, many of the nodes which used it keep their current path because it is only slightly worse than the previous path which used the link. As a result, fewer nodes now have the link in the shortest-path tree, so that when it fails again, they are not affected. Thus, after the first failure, the effects of the link are generally limited to a small neighborhood around the link where the link is a significant fraction of path costs (Section 4.2.5).

4.2.2 Per-Node Communication

In the introduction we motivated our problem of reducing the number of routing updates by the fact that the slowest router in the network limits the total size of the network by the update rate it can sustain. It is worth asking, therefore, whether the reduction in the number of messages shown above is spread uniformly across the network, or whether there were bottleneck routers whose performance would ultimately dictate the sustainable update rate. Table 4.4 shows the *maximum* number of messages generated by any single node during the simulation, relative to `ls`. In contrast to the total communication, this number shows the maximum load placed on an *individual* node rather than the network as a whole. Although it is does not show short-term load on a node, it does show whether a routing algorithm spreads the communication costs evenly across the network.

These results do not differ markedly from the total communication results

Table 4.4: Average (over 10 simulations) of the maximum number of messages generated by any one node, relative to `ls`. The `xl` columns shows values for algorithm parameters $\epsilon = 0.0$ (first value) and $\epsilon = 0.5$ (second value).

	Standard model					Flapping model				
	`dv`	`dv+p`	`lv`	`xl`		`dv`	`dv+p`	`lv`	`xl`	
Crown 64	3.41	1.07	1.06	0.68	0.46	1.09	0.79	0.78	0.49	0.17
Honey 16×16	1.09	0.73	0.68	0.35	0.23	0.42	0.71	0.64	0.24	0.09
Quad 16×16	0.16	0.45	0.43	0.18	0.14	0.12	0.44	0.42	0.10	0.07
Abilene	0.97	0.77	0.77	0.64	0.55	0.98	0.83	0.83	0.55	0.46
Arpanet	2.28	0.91	0.89	0.51	0.45	1.86	0.89	0.87	0.39	0.28
Fuel1221	7.32	0.46	0.46	0.12	0.09	6.56	0.44	0.43	0.10	0.05
Fuel1239	4.85	0.23	0.23	0.20	0.11	1.16	0.21	0.21	0.16	0.05
Fuel1221c	0.74	0.38	0.38	0.37	0.26	0.34	0.35	0.36	0.30	0.16
Fuel1239c	0.95	0.22	0.22	0.22	0.11	0.20	0.22	0.21	0.17	0.05

shown in Table 4.3, indicating that none of the algorithms impacted any one node more heavily than the link-state algorithm, in which the number of messages sent by a node is proportional to its degree. We conclude, therefore, that the benefits of the XL algorithm are spread evenly across the network.

4.2.3 Stretch

In addition to counting the number of messages, we performed additional analysis as described in Section 3. The first quantity we consider is stretch; recall that stretch is the ratio of the forwarding cost to optimal cost between a pair of nodes. Because stretch is an instantaneous measure for each pair, it is not an easy value to summarize for an entire simulation. We use the top stretch centile for each pair. By the top centile, we mean the lowest upper bound for 99% of the simulation duration. In other words, a pair's stretch is at most the top centile value 99% of the time. In Table 4.5 we report the median, average and maximum top centile stretch over all pairs for `xl` with parameter $\epsilon = 0.5$, corresponding to maximum allowed stretch of 1.5. For all other algorithms, including `xl` with $\epsilon = 0.0$, the maximum top centile stretch was zero as expected, and is not shown.

Clearly, while the observed stretch approaches the 1.5 (the maximum allowed

Table 4.5: Top centile stretch for `xl` with parameter $\epsilon = 0.5$. The median, average, and maximum of the top centile were taken over all source-destination pairs; a pair's instantaneous stretch is at most its top centile value 99% of the time.

	Standard model			Flapping model		
	Med	Avg	Max	Med	Avg	Max
Crown 64	1.00	1.02	1.43	1.00	1.01	1.39
Honey 16×16	1.00	1.05	1.45	1.00	1.02	1.44
Quad 16×16	1.00	1.02	1.43	1.00	1.01	1.40
Abilene	1.00	1.01	1.22	1.00	1.01	1.18
Arpanet	1.00	1.02	1.45	1.00	1.01	1.41
Fuel1221	1.00	1.01	1.34	1.00	1.01	1.33
Fuel1239	1.00	1.04	1.41	1.00	1.02	1.41
Fuel1221c	1.00	1.02	1.35	1.00	1.01	1.33
Fuel1239c	1.00	1.04	1.42	1.00	1.02	1.41

per the stretch parameter) for some source-destination pairs, the average observed stretch is quite good, in all cases at most 5% of 1.0. In fact, the median was 1.00, indicating that for the majority of nodes the forwarding path is optimal. There are three reasons why this occurs. The first is that in a network with sufficient redundancy, a single link will affects only a few of the total shortest paths; in this sense, the above stretch measurement is sensitive to the number of link changes and their downtime duration in the simulation. The second reason pairwise stretch is generally low is that the effect of a sub-optimal detour is diminished with distance, so that while a suboptimal path may have stretch near the maximum, the stretch of longer paths including the sub-optimal path as a sub-path will decrease in proportion to the overall path length. Finally, another phenomenon is taking place: stretch allows the XL algorithm to suppress updates to distant nodes for which the next hop is *not affected* by a slight increase in the cost. In other words, the optimal path is chosen even though the node in question "thinks" it is using a longer path. Thus, by just *allowing* the XL algorithm to choose sub-optimal paths we were able to get the reduction in communication complexity while paying only a fraction of the allowed 50% penalty.

Table 4.6: Forwarding loop duration maximum over all source-destination pairs, relative to `ls`. The forwarding loop duration for a pair of nodes u and w is the duration of time $\phi(u, w)$ was infinite.

	Standard model					Flapping model				
	`dv`	`dv+p`	`lv`	`xl`		`dv`	`dv+p`	`lv`	`xl`	
Crown 64	4.08	0.00	0.00	1.04	0.88	9.28	0.00	0.00	1.17	0.66
Honey 16×16	17.2	0.00	0.00	0.99	0.88	1.49	0.00	0.00	0.90	0.80
Quad 16×16	5.96	0.00	0.00	1.00	0.98	1.24	0.00	0.00	1.16	1.03
Abilene	2.27	0.00	0.00	0.79	0.87	1.83	0.00	0.00	0.93	0.98
Arpanet	3.12	0.00	0.00	0.91	0.82	2.86	0.00	0.00	0.94	0.82
Fuel1221	74.2	0.00	0.00	0.79	0.79	46.0	0.00	0.00	0.79	0.81
Fuel1239	85.6	0.00	0.00	0.92	0.87	24.9	0.00	0.00	0.95	0.85
Fuel1221c	10.8	0.00	0.00	0.87	0.85	2.60	0.00	0.00	0.96	0.95
Fuel1239c	25.1	0.00	0.00	0.95	0.86	2.24	0.00	0.00	0.99	0.85

4.2.4 Convergence

Finally, we consider the convergence time of the XL routing algorithm. In Chapter 2 we proved that a finite time after the network becomes quiet the views stop changing, meaning that that the algorithm eventually converges. In this section, we would like to determine experimentally how long is "eventually." Experimentally, "convergence time" means the time it takes a routing algorithm to establish a desirable (i.e., sound and complete) forwarding configuration. In essence, it combines the time it takes a routing algorithm to re-establish a sound (or loop-free) configuration after a link failure and the time it takes the algorithm to start using a lower-cost path when it becomes available.

The `analyzer` program does not measure convergence time directly; instead, it measures the duration of forwarding loops and the time to establish a new forwarding path when a node becomes reachable. The former is reported in Table 4.6 as the maximum, over all source-destination pairs, of the combined duration of forwarding loops. The time to establish a new forwarding path is reported in Table 4.7 as the maximum, over all source-destination pairs, of the total time the forwarding distance was infinite while the optimal distance was not. In both tables, results are shown relative to `ls`.

Table 4.7: Maximum duration of infinite forwarding-to-optimal distance ratio relative to `ls`. The maximum is taken over all source-destination pairs. The infinite forwarding to optimal distance ratio duration for a pair of nodes u and w is the duration of time when $\|\phi(u, w)\|$ was infinite but $\delta(u, w)$ was not.

	Standard model					Flapping model				
	`dv`	`dv+p`	`lv`	`xl`		`dv`	`dv+p`	`lv`	`xl`	
Crown 64	2.58	2.74	2.73	1.54	1.74	5.29	5.44	5.37	1.45	1.41
Honey 16×16	1.19	3.08	2.46	1.10	1.09	1.30	4.85	3.12	1.02	0.93
Quad 16×16	1.10	2.54	2.00	1.03	1.03	1.02	2.92	2.12	0.99	0.99
Abilene	1.25	1.41	1.41	1.05	1.14	1.36	1.55	1.56	1.01	1.02
Arpanet	1.29	1.41	1.34	0.95	0.94	1.20	1.48	1.46	0.96	0.89
Fuel1221	1.04	1.15	1.09	0.60	0.63	1.06	1.16	1.14	0.52	0.52
Fuel1239	1.15	1.44	1.36	0.75	0.76	1.04	1.24	1.22	0.74	0.70
Fuel1221c	1.16	1.38	1.36	1.03	1.09	1.33	1.62	1.41	1.00	0.98
Fuel1239c	1.54	1.76	1.57	1.05	1.03	1.50	1.70	1.63	1.01	0.93

It comes as no surprise that the generic distance vector algorithm has a problem with long-lasting loops. In contrast, loops in `dv+p` and `lv` are extremely rare and short-lived because, although they are not loop-free at all times, their policy for accepting a next hop are fairly conservative. Conversely, this "reluctance" to accept a new path is also responsible for the longer time to establish a new forwarding path.

With the exception of the Crown network, `xl` had slightly better convergence times than `ls`. At first glance, it may seem paradoxical that *any* algorithm should do better than `ls`, since `ls` floods all updates without delay. The reason `xl` sometimes outperforms `ls` is because under `xl`, when a new path becomes available, a node's forwarding path changes only if the new path is much better, thus avoiding transitory loops while the path stabilizes. On the other hand, the time to accept a new forwarding path is generally longer than `ls` because `xl` has less information about the network, so that when a link fails, it may be necessary for the link failure update to propagate before a bypass route is advertised. Cut Vertex Partitioning partially remedies this, because when a cut edge comes up, only the corresponding cut vertices need to be updated to restore the path.

It is also worth noting that the absolute durations behind Tables 4.6 and 4.7 for `ls` are actually quite small: relative values within 50% of 1.00 are unlikely to be

operationally significant.

4.2.5 Scalability

To evaluate the scalability of the XL routing algorithm relative to existing algorithms, we simulated each algorithm on families of networks of increasing size: the HONEY synthetic network family and the ORB re-scaled network family described earlier. Each combination of algorithm, network, and link event model (Standard and Flapping) was simulated 5 times and averaged in reporting results. Figures 4.2 and 4.3 shows the number of messages as a function of network size for the ORB and HONEY families of networks, respectively.

As the network size increases, the XL algorithm maintains its good relative performance. As with other algorithms, however, the routing communication load still grows linearly with the size of the network. This is because a link failure still triggers partial flooding to nodes whose shortest-path tree included the failed link, and roughly half of all simulation events are link failures. In a connected network, a node's shortest-path tree contains $n-1$ nodes, so that when link events are independent, the probability of a node being affected by a network change is $(n-1)/m$. Thus the expected number of nodes affected by a random, independent link failure is about n^2/m; in a network such as the Internet where m/n is a small constant, a random link failure will be propagated to a constant fraction of the nodes.

When link failures are not independent, the XL algorithm can extract an advantage. This is because when a link comes up after failure, only nodes for which the link significantly improves the distance to a destination learn of the change, per Condition C1. If the link fails again, it is only these same nodes that need to be updated. Figure 4.4 shows an example of this phenomenon in the QUAD 13×6 network.

In general, the nodes for which an edge (u, v) changing weight from ∞ to a finite value a improves some distance in the network by a factor of $(1+\epsilon)$ are precisely the nodes of distance at most

$$r = \frac{b - (1+\epsilon)a}{\epsilon}, \tag{4.1}$$

from u, where b is the distance from u to v when $e(u, v) = \infty$, that is, b is the *detour*

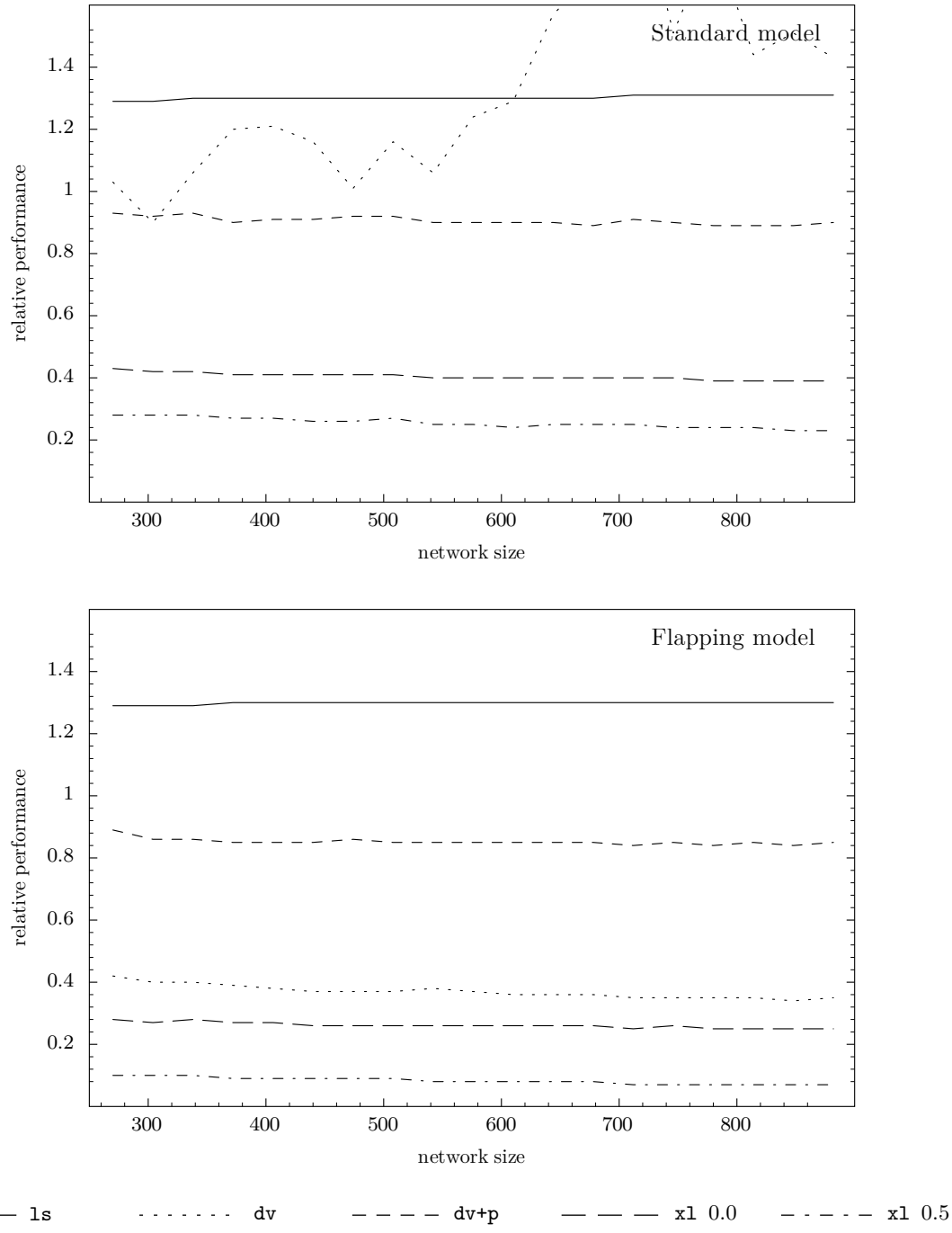


Figure 4.2: Number of messages as a function of network size for the HONEY $16 \times Y$ family of networks; values are normalized by the number of edges in the graph. The family consists of 19 networks for $Y = 7 \ldots 25$.

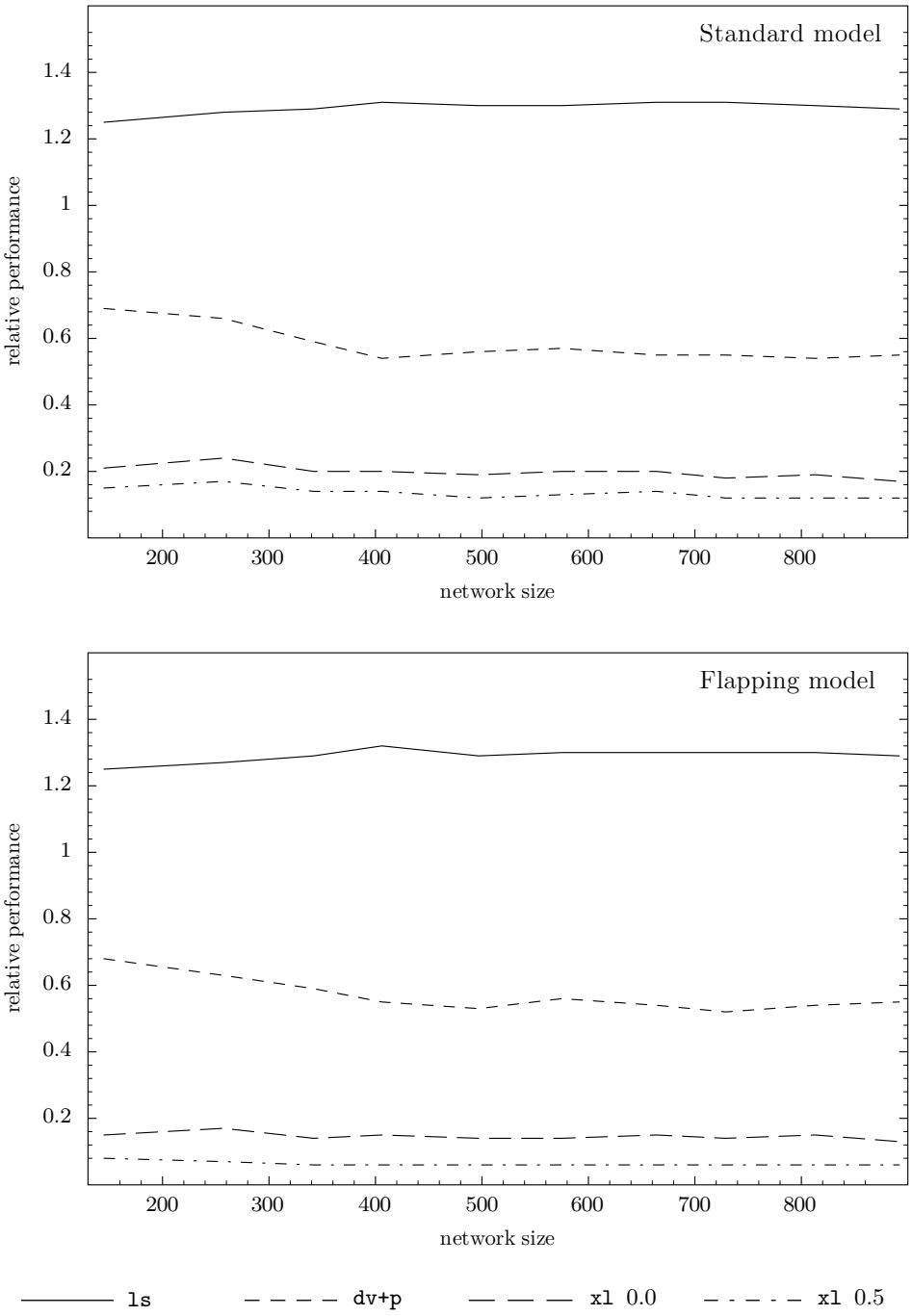
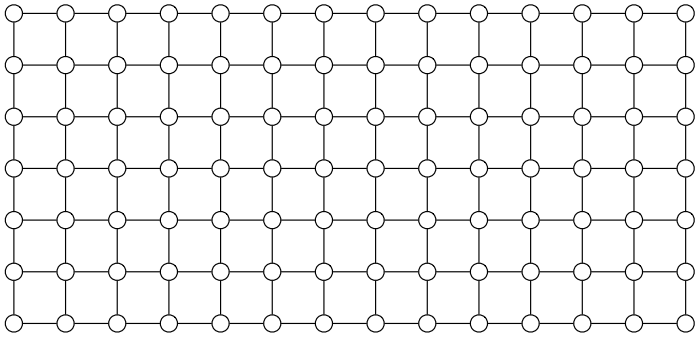
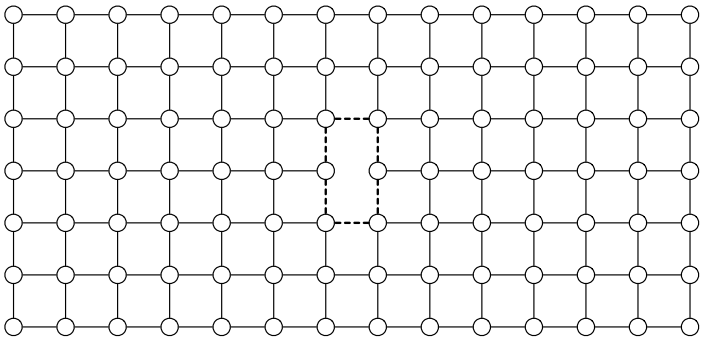


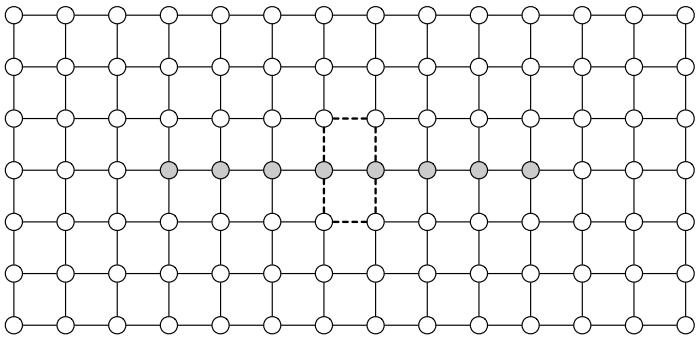
Figure 4.3: Number of messages as a function of network size for the ORB family of networks; values normalized by number of edges in the graph. Both `dv+p` and `lv` performed similarly (within 5%); only `dv+p` is shown. The distance vector algorithm was omitted because its communication exceeded the other algorithms by a factor of 5 in the Standard model and nearly an order of magnitude on the Flapping model.



(a) Initial configuration



(b) Detour when link fails



(c) Updated nodes when link restored

Figure 4.4: A flapping link in the QUAD 13×6 network under the XL algorithm with $\epsilon = 0.5$. Starting with the initial network in subfigure (a), a link failure is flooded to every node on the network, as well as any previously-suppressed updates about the possible detour shown dotted in subfigure (b). When the link comes back up, Constraint C1 only requires a subset of nodes (shaded) to be notified, shown in subfigure (c). If the link continues to fail, only the shaded subset is notified.

cost of (u, v). In the absence of other link events, the effect of a single flapping link will be limited to a neighborhood around the link, the size of which is proportional to the detour cost of the link.

Acknowledgements

This chapter contains work previously presented at the 2008 ACM SIGCOMM Conference and appearing in its proceedings, which is joint work with Ramamohan Paturi, Geoffrey M. Voelker, and Stefan Savage [32].

Chapter 5

Conclusion

The primary objective of this dissertation was to determine whether routing can be made more scalable without imposing artificial hierarchies on a communication network. Our main contribution is a new link state routing algorithm, called XL, which *automatically* limits the scope of routing updates, thus significantly reducing the communication overhead of routing. This chapter summarizes this and other contributions, and then describes a number of directions for future work.

5.1 Contributions

At the outset, we were motivated by whether complete flooding of network state information, as done by OSPF and other link state protocols, is inherently necessary. Although in certain cases this is indeed so (see the example in Section 1.2.2 on page 15), it seemed to us that in many real networks complete flooding was *not* necessary because information from one part of the network often did not affect the routing decisions made in another. *Often*, but not *always*, and therein lay the difficulty. Contrary to proposals that simply limited the distance an update was propagated [27], some far-away updates are *necessary* for correct routing and cannot be suppressed or delayed (again, see the example in Section 1.2.2).

Our main technical contribution is a set of sufficient conditions for correct routing specifying which link state updates must be propagated and which can be suppressed. In Chapter 2 we formalized "correct routing" in terms of *soundness* and

completeness—formal properties of a forwarding network configuration. We then proved that the XL routing algorithm achieves these properties.

A major advantage of XL over other proposed routing algorithms is that XL is fundamentally compatible with other link state algorithms, as use in OSPF, for example. This is because naive flooding automatically satisfies the XL conditions, and can thus be regarded as an instance of XL satisfying Conditions S1, S2, and C1. In particular, *existing* routing protocols like OSPF can be augmented with the XL update suppression mechanism and remain compatible with existing implementations, providing an attractive path for incremental deployment.

We evaluated XL in simulation. Of course, a purely analytical evaluation would have been desirable as well; unfortunately meaningfully establishing the advantage of one routing algorithm over another requires developing believable realistic network and a link event models. The former is still an active area of research [2, 29, 33, 35, 41, 58, 64], while work on the latter is in its infancy. Without a network model, the analysis simple runs into the trivial flooding lower bound exhibited in Section 1.2.2.

On Chapter 4 we demonstrated experimentally that XL significantly reduces the number of updates generated during a network change. At the same time, XL does not sacrifice other desirable qualities such as convergence time. Moreover, the actual stretch induced by XL is much less than specified by the maximum stretch parameter; in practice, over half of all forwarding paths experienced no stretch at all! Thus by allowing *some* stretch and using Cut Vertex Partitioning, we have allowed the algorithm to suppress 70-90% of the updates without sacrificing any performance in most of the network.

Our experiments used a forwarding network simulation system (Chapter 3) written expressly for the purpose of routing algorithm comparison. Unlike traditional network simulators, our simulation system only simulates link cost changes, forwarding tables, and the routing algorithms. By restricting the simulation to the relevant aspects of routing algorithms, we were able simulate larger networks and measure precisely the qualities of interest to us. Our simulation system proved a useful tool for studying routing algorithm behavior.

5.2 Directions for Future Work

The obvious next step in this work is a working OSPF-compatible implementation of the XL algorithm. Such an undertaking poses the significant challenge of not only implementing the nitty-gritty details of a routing protocol (e.g., keep-alives, message re-tramission, queuing) but also ensuring compatibility with existing OSPF implementations. Of course, complete or even partial deployment of the XL algorithm in the real world would enable further study of the algorithm. Perhaps one of the most interesting questions we could answer is to what extent partial deployment of XL on new hardware reduced the overhead of routing updates on older hardware running the basic link state algorithm and which, for one reason or another, could not be upgraded to use XL.

Of more theoretical interest is the improvement of XL to reduce updates further. One obvious direction for improving XL made apparent by the example in Section 4.2.5 is to reduce the scope of S1-triggered updates which are normally flooded to the entire network (more precisely, to the nodes that include the failed link in their shortest-path tree). This would allow updates to be wholly limited to a local neighborhood, provided certain requirements are met locally.

As we suggested in the introduction, routing and network design go hand-in-hand. By combining sophisticated routing algorithms with reasonable network design criteria, such as the existence of short detours, we believe it is possible to achieve *fully scalable* routing while still retaining the flexibility afforded by a completely distributed network architecture.

Index

Bibliography

[1] Abilene interior-routing metrics. `http://noc.net.internet2.edu`, March 2006.

[2] D. Alderson, J. C. Doyle, R. Govindan, and W. Willinger. Toward an optimization-driven framework for designing and generating realistic Internet topologies. *ACM SIGCOMM Computer Communication Review*, 33(1):41–46, January 2003.

[3] G. R. Ash. Design and control of networks with dynamic nonhierarchial routing. *IEEE Communications Magazine*, pages 34–40, October 1990.

[4] P. Baran. On distributed communications networks. *IEEE Transactions on Communications Systems*, 12(1):1–9, March 1964.

[5] C. B. Barr. Telegraph stations in the United States, the Canadas & Nova Scotia (map), 1853.

[6] R. W. Barton. *Telex: A Detailed Exposition of the Telex System of the British Post Office*. Sir Isaac Pitman & Sons, 1968.

[7] J. Behrens and J. J. Garcia-Lunes-Aceves. Distributed, scalable routing based on link-state vectors. In *Proceedings of the ACM SIGCOMM Conference*, pages 136–147, 1994.

[8] R. Birke, M. Mellia, M. Petracca, and D. Rossi. Understanding VoIP from backbone measurements. In *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM)*, pages 2027–2035, 2007.

[9] S. Blake, D. L. Black, M. A. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC 2475: An architecture for differentiated services, December 1998.

[10] C. Boutremans, G. Iannaccone, and C. Diot. Impact of link failures on VoIP performance. In *Proceedings of the 12th international Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 63–71, 2002.

[11] H. Buhrman, J.-H. Hoepman, and P. Vitányi. Optimal routing tables. In *Proceedings of the 15th Symposium on Principles of Distributed Computing*, pages 134–142, 1996.

[12] H. T.-H. Chan, A. Gupta, B. M. Maggs, and S. Zhou. On hierarchical routing in doubling metrics. In *Proceedings of the 16th Symposium on Discrete Algorithms*, pages 762–771, 2005.

[13] C. Cheng, R. Riley, S. P. R. Kumar, and J. J. Garcia-Lunes-Aceves. A loop-free extended Bellman-Ford routing protocol without bouncing effect. *ACM SIGCOMM Computer Communication Review*, 19(4):224–236, September 1989.

[14] G. J. Chretien, W. M. Konig, and J. H. Rech. The SITA network. In *Proceedings of the NATO Advanced Study Institute on Computer Communication Networks*, pages 373–396, 1973.

[15] Cisco Systems. *OSPF Design Guide*. Document ID 7039.

[16] M. D. Connolly, T. A. Connolly, and T. J. McTighe. Automatic telephone exchange. US Patent 222,458, December 1879.

[17] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional IP routing protocols. *IEEE Communications Magazine*, 40(10):118–124, October 2002.

[18] P. Fraigniaud and C. Gavoille. Memory requirement for universal routing schemes. In *Proceedings of the 14th Symposium on the Principles of Distributed Computing (PODC)*, pages 223–230, 1995.

[19] J. J. Garcia-Lunes-Aceves. Loop-free routing using diffusing computations. *Transactions on Networking*, 1(1):130–141, Feb 1993.

[20] C. Gavoille and S. Pérennès. Memory requirement for routing in distributed networks. In *Proceedings of the 15th ACM Symposium on the Principles of Distributed Computing*, pages 125–133, 1996.

[21] A. Girard. *Routing and Dimensioning in Circuit-Switched Networks*. Addison-Wesley, 1990.

[22] A. D. Godley. *Herodotus: The Histories*. Harvard University Press, 1920.

[23] B. Goode. Voice over Internet Protocol (VoIP). *Proceedings of the IEEE*, 90(2):1495–1517, September 2002.

[24] F. E. Heart, A. McKenzie, J. M. McQuillan, and D. C. Walden. ARPANET completion report. Technical Report 4799, Bolt, Baranek and Newman, 1978.

[25] P. A. Humblet. Another adaptive distributed shortest path algorithm. *IEEE Transactions on Communications*, 39(6):995–1003, June 1991.

[26] G. Iannaccone, C. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot. Analysis of link failures in an IP backbone. In *Proceedings of the Second Internet Measurement Workshop*, pages 237–242, 2002.

[27] A. Iwata, C.-C. Chiang, G. Pei, M. Gerla, and T.-W. Chen. Scalable routing strategies for ad hoc wireless networks. *IEEE Journal on Selected Areas in Communication*, 17(8):1369–1379, August 1999.

[28] J. M. Jaffe and F. H. Moss. A responsive distributed routing algorithm for computer networks. *IEEE Transactions on Communications*, COM-30(7):1758–1762, July 1982.

[29] R. Kannan, P. Tetali, and S. Vempala. Simple Markov-chain algorithms for generating bipartite graphs and tournaments. *Random Structures and Algorithms*, 14(4):298–308, July 1999.

[30] A. Khanna and J. Zinky. The revised ARPANET routing metric. *ACM SIGCOMM Computer Communication Review*, 19(4):45–56, September 1989.

[31] J. E. Kingsbury. *The Telephone and Telephone Exchanges*. Arno Press, 1972.

[32] K. Levchenko, G. M. Voelker, R. Paturi, and S. Savage. XL: An efficient network routing algorithm. In *Proc. of the 2008 ACM SIGCOMM Conference*, pages 15–26, 2008.

[33] L. Li, D. Alderson, W. Willinger, and J. C. Doyle. A first-principles approach to understanding the internet's router-level topology. In *Proc. of the 2004 ACM SIGCOMM Conference*, pages 3–14, 2004.

[34] P. Mahadevan, C. Hubble, D. Krioukov, B. Huffaker, and A. Vahdat. Orbis: Rescaling degree correlations to generate annotated Internet topologies. In *Proc. of the 2007 ACM SIGCOMM Conference*, pages 325–336, 2007.

[35] P. Mahadevan, D. Krioukov, K. Fall, and A. Vahdat. Systematic topology analysis and generation using degree correlations. In *Proc. of the 2006 ACM SIGCOMM Conference*, pages 135–146, 2006.

[36] R. Mahajan, N. Spring, D. Wetherall, and T. Anderston. Inferring link weights using end-to-end measurements. In *Proceedings of 2nd Internet Measurement Workshop*, pages 231–236, 2002.

[37] G. Malkin. RFC 2453: RIP version 2, 1998.

[38] A. Markopoulou, F. A. Tobagi, and M. J. Karam. Assessment of VoIP quality over Internet backbones. In *Proceedings of the 21st IEEE International Conference on Computer Communications (INFOCOM)*, pages 150–159, 2002.

[39] J. M. 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):1802–1811, Dec 1978.

[40] J. M. McQuillan, I. Richer, and E. C. Rosen. The new routing algorithm for the ARPANET. *IEEE Transactions on Communications*, 28(5):711–719, May 1980.

[41] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An approach to universal topology generation. In *Proceedings of the Ninth Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 346–353, 2001.

[42] J. T. Moy. *OSPF: Anatomy of an Internet Routing Protocol*. Addison-Wesley, 1998.

[43] J. T. Moy. *OSPF Complete Implementation*. Addison-Wesley, 2000.

[44] H. Mustafa. *Postal Technology & Management*. Lomond Systems, Inc., 1971.

[45] D. R. Oran. RFC 1142: OSI IS-IS intra-domain routing protocol, February 1990.

[46] V. D. Park and M. S. Corson. A performance comparison of the temporally-ordered routing algorithm and ideal link-state routing. In *Proceedings of the 3rd IEEE Symposium on Computers and Communications*, pages 592–598, 1998.

[47] D. Peleg and E. Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM*, 36(3):510–530, July 1989.

[48] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *Second IEEE Workshop on Mobile Computer Systems and Applications*, page 90, 1999.

[49] B. Rajagopalan and M. Faiman. A new responsive distributed shortest-path routing algorithm. In *Proceedings of the ACM SIGCOMM Conference*, pages 237–246, 1989.

[50] J. Rexford. *Handbook of Optimization in Telecommunications*, chapter Route Optimization in IP Networks. Springer, 2006.

[51] R. F. Rey, editor. *Engineeering and Operations in the Bell System*. AT&T Bell Laboratories, 1983.

[52] L. G. Roberts. The evolution of packet switching. *Proceedings of the IEEE*, 66(11):1307–1313, 1978.

[53] C. L. H. Rutgers. *Introduction to IGRP*. Cisco Systems, August 1991. Document ID 26825.

[54] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, September 1983.

[55] T. P. Shaffner. *The Telegraph Manual: A Complete History and Description of the Semaphoric, Electric and Magnetic Telegraphs of Europe, Asia, Africa, and America, Ancient and Modern.* Pudney and Russell, 1859.

[56] A. Shaikh, C. Isett, A. Greenberg, M. Roughan, and J. Gottlieb. A case study of OSPF behavior in a large enterprise network. In *Proceedings of the 2nd Workshop on Internet Measurement*, pages 217–230, 2002.

[57] S. Swihart. The genesis and early development of telephone exchange service. *Telecom History*, 1:2–89, 1994.

[58] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topology generators: Degree-based vs. structural. In *Proc. of the 2002 ACM SIGCOMM Conference*, pages 147–159, 2002.

[59] M. Thorup. OSPF areas considered harmful. Unpublished manuscript, April 2003.

[60] United States Government Accountability Office. U.S. Postal Service: The Service's strategy for realigning its mail processing infrastructure lacks clarity, criteria, and accountability. GAO-05-261, April 2005.

[61] United States Postal Service. The United States Postal Service: An American history 1775–2006. Publication 100, May 2007.

[62] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. Scalable high speed IP routing lookups. In *Proceedings of the 1997 ACM SIGCOMM Conference*, page 182, 1997.

[63] G. Wilson. *The Old Telegraphs.* Phillimore & Co., 1976.

[64] S.-H. Yook, H. Jeong, and A. L. Barabási. Modeling the Internet's large-scale toplogy. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 99, pages 13382–13386, 2002.

[65] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new resource reservation protocol. *IEEE Network*, 7(5):8–18, September 1993.