

A Dictionary Architecture for Optimized Intra-Domain Routing

Joseph Chabarek
University of Wisconsin-Madison
Email: jpchaba@cs.wisc.edu

Paul Barford
University of Wisconsin-Madison
Nemean Networks
Email: pb@cs.wisc.edu

Abstract—The unpredictability of outages and traffic dynamics require network engineers to keep careful watch on their infrastructures. When significant changes are identified, engineers often tweak link weights (*i.e.*, traffic engineering), which can have unwanted or ineffective outcomes. In this paper, we describe a new intra-domain routing architecture that seeks simultaneously to enable complex route optimization and eliminate route convergence delays to best meet operational objectives in dynamic network environments. Our approach uses centralized, multi-objective optimization to generate a set of forwarding tables that we refer to as a *dictionary* for each node in a network. When deployed, nodes select from among tables in their dictionary in an autonomous fashion based on the measured state of the network. The set of tables only needs to be recomputed when there are significant changes in network structure or operational objectives. We show that this approach can be implemented with relatively modest resources. We demonstrate the feasibility and capabilities of our method through a series of simulations run on synthetic and real-world network topologies. Our results show that dictionary-based routing is able to effectively maintain performance objectives by automatically adapting to a variety of structural and dynamic changes. We also show that in the common case, dictionary-based routing outperforms link-state algorithms in terms of speed of convergence after a failure and in terms of scalability of failure recovery.

I. INTRODUCTION

Network operators are faced with significant challenges as the scale and complexity of their infrastructures grow. While extremely high availability (*e.g.*, no more than a few seconds down time per month) remains the primary objective, network operators must support a wide variety of application traffic, an increasing number of services, while at the same time ensuring performance meets the targets specified in Service Level Agreements (SLAs). All of this must be done in a highly dynamic environment that is constantly subject to emerging technologies and demands.

Intra-domain routing protocols play an intrinsic role in ISP network stability, performance and behavior. Along with the basic capability of establishing destination-based forwarding tables for all routers in a network, there are many additional features that enable policies to be imposed and enable the protocols to function effectively in large enterprise and service provider environments. The Open Shortest Path First (OSPF) protocol [1] is often cited as being the most widely used intra-domain routing protocol.

Intra-domain routing protocols and careful assignment of link weight costs establish the baseline as traffic flows through

a network [2]. Specifying link weights to achieve operational objectives on a large static topology can be challenging enough, but in infrastructures where faults and outages are routine [3], [4], it is extremely difficult to maintain the network in a satisfactory state over time. Further complicating the issue is the fact that distributed multi-objective optimization algorithms do not scale to large networks. Thus, there is at best limited capability to implement complex routing policies in networks.

Intra-domain routing protocols are also one of the defacto tools for traffic engineering in large networks. Link weights are often tweaked in order to adjust traffic flows on critical links in response to unexpected events (*e.g.*, faults, flash crowds or attack outbreaks). Adjusting link weights in response to changes in network state can sometimes produce highly undesirable results *e.g.*, violation of SLAs.

In this paper we consider the problem of how to efficiently and effectively generate and maintain routes optimized for a range of common scenarios in large, dynamic network environments. Specific requirements include, (*i*) the ability to consider multiple objectives in route generation (*e.g.*, including multi-path routes [5]), (*ii*) the ability to react quickly and correctly to changes in a variety of network conditions (*iii*) scalability and (*iv*) low overheads for on-router memory, and processing, and any required communication traffic.

A key contribution of our approach, which we call *dictionary routing*, is our ability to dynamically support multiple objectives. Instead of fixing the route computation with a single objective (*e.g.*, shortest path based on link weight) *dictionary routing* provides scalable flexibility. While it is common to have a routing protocol compute primary and backup paths in case of failure, *dictionary routing* can support a large number of potentially overlapping paths computed with multiple objectives for a wide range of scenarios. The paths are precomputed allowing for a quick local lookup at the router when real time conditions such as a device failure or a change in network usage that degrades performance on a path.

We deviate significantly from traditional link state algorithms with this approach. *Dictionary routing* does not maintain a complete topological database of the network at every router, it does not require routers to have synchronized views of the network, and failure messaging in the common case is directed to routers actively using the link to avoid costly flooding.

In *dictionary routing* we take cues from clean-slate design initiatives [6], the notion of designing optimizable protocols [7], and from prior work on route precomputation (e.g., [8]). *Dictionary routing*, begins by precomputing routes for all nodes in a network in a centralized fashion. In the route precomputation phase, multiple objectives and characteristics of the network topology are considered using an efficient optimization algorithm. Given the flexibility of our approach a range of objectives and local policies can be considered; these include optimizing for recovery from device failure, changing traffic demands, or minimizing power consumption.

One *dictionary* per router is produced, this data structure is a set of partially redundant ordered paths from a common ingress router to every egress router. The dictionaries are deployed on the routers in the network, which all begin by selecting the “best” forwarding table (e.g., the first entry in the dictionary). At this point, a measurement process is initiated on the links between routers. The details of the measurements are not specific to our protocol and depend on the operational objectives of the network. For example an active probe-based measurement tool could be used to monitor for SLA violations on paths [9]. When a measurement indicates that a performance threshold has been exceeded or a fault has occurred, a node selects a new path from its dictionary and uses other dictionary information to inform the nodes using the offending link that they must also adjust to a new entry in their dictionary. In this way, the routing problem becomes a problem of coordinating the usage of dictionary entries in nodes in an autonomous fashion.

Managing dictionary entries is challenging for several reasons. First, an event (e.g., a failure or performance degrading below a specified threshold) may be sensed by multiple nodes at the same time and the resulting route selections must be made quickly to maintain optimality given the new network state. Second, the impact of an event may not be local in the sense that making a change on one or a few nodes may be insufficient to fully adjust to the effects of the event. Events requiring a fast response including link failures prompt routers to locally look up intelligently precomputed replacement paths. Our protocol remains loop free as we route along entire paths via the *dictionary* and packet tagging, rather than next hops. Over a longer time period, the centralized controller recomputes the optimal state of the routing protocol and prompts the ingress routers to comply.

To assess the feasibility of dictionary routing, we begin with an analysis of the overhead required to support dictionaries in routers. The size of the dictionary depends on a number of issues including the number of routers in a network and the amount of desired path redundancy. We explore the trade offs of multipath routing in our protocol and we quantify the use of a small amount of storage at the source of each link that is used to direct the messaging that occurs when a performance threshold is exceeded. The results of our analysis on several example topologies show an expected proportionality between the memory overhead and network size indicating only a modest overhead requirement.

In order to compare the dynamic performance of dictionary routing with standard routing protocols, we conducted a series of simulation studies using a variety of synthetic and real-world network topologies. We developed an extensive simulation environment that enabled flexible topology specification, fault injection, link-state routing, and dictionary routing. The results of our analysis show that (i) dictionary routing is highly scalable (ii) failure messaging is reduced up to 66% compared to link-state flooding protocols and (iii) that in the common case the routers using the local dictionaries can adapt to failures in the network with local information. We believe that these results demonstrate the feasibility of dictionary routing and make a case for its potential and utility in real-world deployment.

II. RELATED WORK

The most well known intra-domain routing protocols are based on link-state and distance vector algorithms. There has been a good deal of algorithmic work done with respect to calculating the shortest paths in a network quickly. For example, Shaikh [8] stored the intermediate state gathered during the process of computing shortest paths in a network to enable the quick recomputation necessary for link-state broadcasts that include a dynamic quality of service metric. It should be noted that while we also seek to minimize route recall time, our protocol architecture does not run an OSPF process and is significantly different than Shaikh’s.

Centralized computation of routes for a network is often used to mitigate the costs of complex optimization over large amounts of traffic data. There have been many studies concerning the stability of the centralized solutions based on traffic matrices (e.g. [10], [11].) The techniques employed by COPE [12] address maintaining performance from one centralized solution across a range of traffic profiles via clever optimization techniques.

Recently, there have been a number of systems that use a centralized controller to coordinate some aspect of the control plane or network management. Ethane [13] implements a single centralized set of policy rules on switches. Similarly, Sane [14] uses a domain controller to manage a set of services and associated authentication requirements. The 4D architecture [15] presents a clean slate framework cleanly defining functionality across four partitions. In *dictionary routing* we utilize a central controller as well as distributed dictionaries to quickly adapt to changes in the network.

There has been work on multi-topology routing including MRC [16] and extensions for OSPF [17] where a limited number of multiple complete network topologies are computed, stored, and used locally with configuration identifiers. In comparison, our solution uses a collection of paths rather than hop by hop forwarding over configurations, is implemented separately from a link state protocol, and flexibly supports many different objectives.

MPLS [18] is commonly implemented on top of a topology revealing protocol such as OSPF and provides a convenient way to route traffic along specified paths. The MPLS layer

is dependent on another layer for topology information. Additionally, a traffic engineering process can be active inside of MPLS with reservations such as RSVP-TE [19] or out of band with periodic optimization and resulting redistribution of traffic over paths. MPLS also has a number of recovery mechanisms that facilitate connectivity when an underlying routing protocol has not converged including label swapping [20].

Recently, additions to OSPF by Xu and Rexford [21] allow for arbitrary traffic splitting in OSPF given a set of centrally computed routes within a single OSPF area. This allows for a streamlined process of traffic engineering including traffic measurement, data centralization, optimization, and link weight tweaks [22]. Our technique takes advantage of the path-based traffic splitting while supporting large networks without partitioning a domain into multiple areas as is common within OSPF. With multiple potential routes available at a router for one source destination pair, MATE [23] provides a framework for choosing between paths based on latency in a stable fashion. While we do not implement the framework in our simulation, it is important to note the opportunities inherent in path based routing. In FCP [24] the authors take advantage of the fact that the set of potential links is updated over a relatively long time scale while the set of functioning links can fluctuate quickly. With this intuition, the authors use failure carrying packets to update the set of functioning links on a router while relaxing consistency constraints. While we also relax the consistency requirement, our approach uses a logically centralized controller and is path based.

We draw on technologies used for inter-domain routing as inspiration for our system. RCP [25] provides a logically centralized routing system for iBGP. With this system, a centralized control platform makes intra-domain routing decisions based on external observation in real time. Finally, Yang et al. [26] show the advantages of a relatively small number of redundant paths in routing around disruptions in the global network using source routing based route deflections. Considering our vantage point, we have more control over packet formatting and do not have to depend on a source routing or user end-system testing.

III. DICTIONARY ROUTING ARCHITECTURE

1) *Framework*: We present a general, flexible framework for dictionary routing by describing the necessary components. Next, we provide a specific configuration including an objective for choosing paths. *Dictionary Routing* includes (i) a logically centralized controller, (ii) a dictionary data structure specific to each router, (iii) a lightweight technique to prevent stale dictionary entries, (iv) a control channel between each router and the controller, (v) and directed notification between routers to enable fast switching between paths for performance recovery. While retaining the flexibility inherent in the protocol, we specify measurement that is required for functionality and point out measurement capabilities within the architecture which are useful for traffic engineering but do not natively exist in link-state routing. We provide two specific

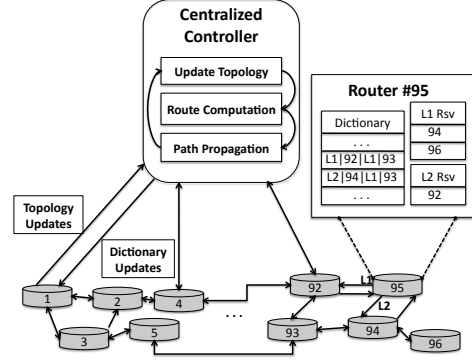


Fig. 1: The dictionary architecture includes these components: (i) a route controller that exchanges information with the routers, (ii) dictionaries stored in the routers, and (iii) the path reservations stored at the origin of each directed link.

heuristics for precomputing paths providing the controller with an objective function while noting that this component of the framework is configurable to support any local policy or performance objective.

To act on a path granularity and arbitrarily split traffic between paths the ingress routers tag a packet with the entirety of its path through the AS. While we simulate a simplified tagging architecture by prepending a series of global link identifiers the requirements in our framework regarding tags can be met by using MPLS. The tag is mutable, any of the intermediate nodes can choose to re-tag and reroute the packet, if there is a link outage downstream.

As shown in Figure 1, the logically centralized controller is used to generate and update the dictionaries on each router. To keep the controller’s view of the network up to date, the routers send the controller changes in the network as soon as they occur. The dictionaries contain potentially redundant paths as an approximation to the network topology, which we use to decouple failure recovery from expensive network-wide coordination or in-network topology querying at the time of failure. In the common-case, with appropriate heuristics, we show that *dictionary routing* is able to locally store the paths needed to recover from single link, single node, and multiple link failure. However, in extraordinary cases, the router must query the controller for new dictionary entries if the local dictionary does not have an alternative path, we call this a “cache miss.”

We design our framework with scalability as a requirement. As a result, our framework uses directed messages for failure notification and removes any assumption about global consistency between the dictionaries stored at different routers. We limit the communication required when a network event occurs avoiding, in the common case, the broadcast requirement that is characteristic of link state algorithms. This relaxation of dictionary consistency has a large impact on the

structure of the protocol.

Dictionary routing seeks to give nodes enough local information to adjust quickly to failures and rapidly fluctuating traffic patterns while providing some global perspective for optimization with the controller. The controller is designed to use the standard process of gathering path usage information to a central location, computing a traffic matrix, solving a traffic optimization problem, then shifting the routes to match a new configuration.

2) *Logically Centralized Controller*: Enabling global policy specification and coordination is most naturally addressed through the use of a logically centralized control plane. As a result, we need a controller to act as a central repository for topology updates. The controller must maintain two-way connectivity with all of the routers so that it can transmit updates when required. The controller has the choice to be proactive with route re-computations, when it receives topology information by periodically recomputing the dictionaries in the network and sending out changes in configuration to the routers. Also, if a router's dictionary finds that the number of available paths for a particular source-destination pair falls under a threshold the router can query the controller and ask for more paths.

Connectivity between the individual routers and the controller is vital. Each router has a set of paths to the controller which are active, exactly the same as the dictionary entries to a particular egress point. The controller must have the up to date topology information to compute paths. After detecting a change in topology a router's first response is to send an update to the controller. If connectivity between router and controller is lost we propose the following mechanism. If all routes become unavailable from the router to the controller, the router must initiate a network-wide broadcast requesting new paths to the controller from the controller. In the unlikely event that the controller loses connectivity with a router, the controller can initiate a similar broadcast. This broadcast can be implemented as a path vector advertisement, similar to BGP, where the intermediate routers mark the update so that at the destination the full path to the resource is known. The router or controller now has a path which it can use to update, if the router and controller cannot establish contact with the advertisement they are physically disconnected and the router is not able to take part in the protocol.

Any additional links or routers in the network can easily be accommodated by sending a message to the controller with the updated topology, the controller will propagate new dictionaries out to the network. While we have stipulated that the controller is logically centralized implementing *dictionary routing* with multiple synchronized controllers would improve robustness and minimize latency. We leave this as future work.

3) *Dictionary*: The dictionary is a ranked list of objects containing paths that start at the router and end with all egress nodes. Each entry contains an ordered list of identifiers for routers and links corresponding to the route. In addition to simply containing paths, the dictionary chosen by the controller can contain meta data giving priority to certain

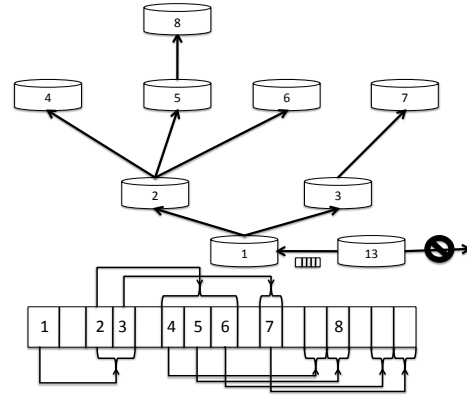


Fig. 2: Paths are compressed into a single use broadcast tree which is encoded in a special packet.

paths, in our prototype version we use a simple ordering of paths from best to worst, however there is opportunity for more sophisticated hints.

A dictionary entry can be set to active or inactive. An active entry will enable the router to receive an update if any link on the path goes down. There is no requirement that an active entry transit traffic, rather it is simply an indication that the router wants to receive status updates for this path. An inactive entry gives a potential path, however this path is not updated if a link failure occurs.

In a typical lookup into the dictionary, the router will have a particular egress node fixed, allowing for easy indexing and quick retrieval as the total number of entries for a particular source-destination pair is a small constant. The pertinent information in the entry can then be added into the forwarding table for speedy data-plane processing. Multiple dictionary entries for a source-destination pair can be used concurrently with any hash based traffic splitting algorithm.

4) *Link Usage Notification*: We implement link usage notifications to enable direct messaging between a device with an under-performing link and the routers with dictionary entries that use the link. Link usage notifications originate at the ingress router and inform routers along a path that an ingress needs to be notified when a link becomes unusable. When a dictionary activates a particular entry, the router sends a probe down the path to ensure every link in the path is in a usable state. The probe traverses the path listed in the dictionary and the egress node returns an acknowledgment, on any available path. The source node then sends a confirmation message to the egress down the activated path and when the intermediate routers process the confirmation message, they record the request to use the link.

With a link, the router that controls the source of the directed link now has a list of nodes which need to be notified in the event of a failure. When the router that controls a local link needs to notify the source list of a change in state, we implement a path compression algorithm to combine

the messages to the ingress routers. The path compression greedily takes the best paths to the affected ingress routers and combines overlapping paths into a special packet. The packet has a tag structure that allows it to traverse multiple links. Only active paths are used in the encoding to ensure delivery of the failure message. Specific link usage does not have to be encoded in the packet, the routers will know which routers serve as the endpoints of its outgoing links as shown in Figure 2. If a path from the notifying device does not exist in the forwarding table, the message is flooded to the entire network similar to link state algorithms, and the controller is queried for additional paths.

A. Intelligent Route Precomputation

While we would like to store every potential path from every ingress to every egress in a routers dictionary, for scalability reasons it is important to intelligently precompute routes that in addition to meeting the current performance objectives of the network will meet operational objectives over a broad range of topology changes. Our precomputed routes can have either a performance advantage (top shortest paths) or can improve reliability according to forward looking error prediction. There are many different ways to compute routes attractive for performance and failure recovery including shortest path, robust minmax objectives, and sophisticated multicommodity flow formulations that have predictable performance over a wide range of traffic profiles, any of these tools can be implemented in the controller. Commonly implemented techniques for choosing backup paths include routes that do not have any overlapping links and routes without overlapping nodes. To select a subset of the total potential routes from a source to a destination we have implemented two heuristics as shown in Algorithm 1 for computing a diverse set of paths: the link heuristic and the node heuristic. We note that these heuristics are not specific to the protocol and they act as a starting place for analysis of precomputation.

Algorithm 1 Link heuristic for computing dictionary entries.

```

1: for all node  $n$  such that  $n \in \text{graph } G$  do
2:    $edge\_array = original\_edge\_array$ 
3:    $weights = original\_weights$ 
4:    $G = initialize\_graph(edge\_array, weights)$ 
5:   while  $alt \leq \text{NUM DUPLICATE PATHS}$  do
6:      $paths = dijkstra\_shortest\_paths(G, n)$ 
7:      $report\_paths(paths)$ 
8:      $G = reweight\_active\_edges(MAX\_LATENCY)$ 
9:   end while
10: end for

```

The objective of the link heuristic is to create a cache of entries that use a large number of diverse links. We do not have any prior information on the reliability of individual links so we assume that the links fail with equal probability. To establish a diverse set of paths, the heuristic network topology algorithm begins with the up-to-date topology map with link weights based on latency and computes the shortest path.

The path is recorded, and then the links that were part of the path are assigned a penalty equal to the maximum link delay in the graph. The process resets the weight for each source-destination pair and continues until a specified number of redundant paths have been added for each pair to the dictionary.

The node heuristic uses the same technique as the link heuristic except every link associated with a node used in a previous iteration is penalized, in addition to the links on the path. Prior knowledge can be used to add details about projected fault models. For example, if a router or link has some instability in the past the device can be penalized so that there are backup routes. With using multiple heuristics, we remove redundant paths from consideration in the dictionary.

If the dictionary has the appropriate redundancy, device failure is easily overcome with our protocol. If the faulty component is a single unidirectional link, the controlling router will send a compressed error message to the affected ingress routers as long as there is an active dictionary that does not contain the failed link. With a failed router, the routers controlling the incoming links signal a failure to the ingress nodes and new dictionary entries are chosen. As we show in Section 5, the protocol is also highly resilient against multiple failures.

B. Measurement

Measurement is a vital component in our routing framework. We have three viewpoints built into the framework that can benefit from measurement the physical links, the paths in the dictionaries, and the global network view of the controller. When a directional link is down we require that the router that acts as the source for that link be able to notify the nodes with path reservations that the link is no longer in a good state. In the absence of a hardware mechanism this link level monitoring can be accomplished with a heartbeat probe sent at intervals approximate to the desired reaction time.

On a fine time scale, ingress nodes can periodically send active probe-based measurement over selected paths to reorder dictionary entries as dictated by local policy to conform to operator requirements or a SLA. With the path based architecture we envision that a framework similar to MATE can use changes in network usage such as path latency as queues fill for selection if network administrators are interested in adaptive route assignment.

On a coarse time scale with simple counters, ingress routers can report the number of packets marked with a tag from each path in the dictionary to the centralized controller which can generate an accurate traffic matrix. The traffic matrix can then be used to compute prioritized entries for the local dictionaries.

C. Local Management

Once a router has a dictionary of potential paths, there are many different possibilities for choosing which paths to forward packets along at any given time. There will potentially be conflicts between the coarse grained global preferences set by the controller and any local real-time measurement

or failure recovery that has to be resolved with local policy. Given that the controller has knowledge of the entire topology and potentially a corresponding traffic matrix the controller ordering of dictionary entries on a coarse time scale is preferable. However, directly after a fault or after crossing a performance threshold there will be a period of time between the event and when the controller sends out a new network configuration, during this period local path decisions by the router are necessary, though there is no guarantee of optimal performance. In future work, we envision computing dictionary entries using forward error prediction to minimize the performance disturbance during this period.

IV. ANALYSIS

We implemented a simulation framework to test the functionality and scalability of our system. This framework allows for convenient flexible prototyping. We can easily run a preliminary version of our protocol on networks with topologies on the order of hundreds of nodes.

A. Simulation Framework

We implemented a network simulation utility to measure the overhead of dictionary precomputation and storage and to demonstrate the correctness of our routing protocol. To allow for large topologies, the simulation framework focuses on routing interactions and does not capture packet level details.

We use a standard topology format annotated with link weights as an input. Given a topology file, we create a set of dictionary entries for the routers in the graph. Computing the dictionary is done centrally and with multiple objectives in mind as previously described. By default we set all the dictionary entries to be active to provide an upper bound of the storage overhead of the protocol but we also show the overhead of having one active entry versus the maximum active dictionary entries in our scenarios. A central action process runs the simulation, each node reports local information such as the local dictionary size, the number of path reservations, etc. when queried. Each node has provenance over choosing dictionary entries for routing, multiple potentially overlapping paths can be used.

In a typical scenario the network topology is initialized and nodes are seeded with complete dictionaries computed using our link and node heuristics, after which a number of dynamic events are modeled including failure scenarios and multipath dictionary route selection. We simulate the controller as globally available to all nodes. We do not explicitly simulate the coordination between the controller in the network and the routers for topology updates. Instead, we share information directly between the controller and the nodes via function calls.

Unidirectional link failures act as the atomic failure event. Bi-directional link failure and complete node failure are simulated with multiple unidirectional link failures. In the simulation, each link is set as active or inactive, when a failure occurs the link is toggled to inactive and the router acting as the source for that link sends a path update message to the

ingress nodes stored by the path-ingress cache for the affected link. Communication between nodes is simulated via the action process. Our failure scenarios are randomly generated events that can be persistent or transient. Considering that we are testing a routing protocol, if a link failure event causes the physical network to become disconnected from multiple persistent link outages the failure event is rolled back. All routing protocols will need operator assistance to reestablish connectivity after such an event. At this time we do not have multiple dependency based failures but this is an avenue for future work.

We model a basic version of OSPF consistent with standard link state flooding for route recomputation after link failure.

Since we do not model packet level interactions we cannot simulate path selection for traffic engineering. However, it is important to note that under our scheme any traffic aware route adjustments are easily done on a path granularity.

B. Topologies

We consider several different types of topologies for our analysis. We selected a random set of topologies from the Rocket Fuel project [27] shown in Table I. We also used synthetically generated topologies from Orbis [28] to approximate larger graphs from the Rocket Fuel topologies. Synth 500 is a representative large graph that is a 2k rescale of Rocket Fuel AS 3257.

TABLE I: Test networks used in our routing study. The randomly generated graphs have labels starting with Synth.

Description	Nodes	Directed Edges
AS 1221	108	306
AS 1239	315	1944
AS 1755	87	322
AS 3257	162	656
AS 3967	79	294
AS 6461	141	748
Synth 500	546	2200

With the selected topologies we do not have any data concerning where they attach to other networks. As a result, assume that all routers are capable of egress and all dictionaries must have a complete set of paths to every node. This a slightly pathological case, but a good indication that the dictionary routing framework is scalable.

C. Experiments

With our simulation framework and topologies we conduct a set of experiments to gauge the overhead and responsiveness of our system and compare it to the simulated behavior of a link state routing algorithm. First, we load realistic dictionaries and report the memory overhead. Next, we choose active paths and measure the overhead of the path reservation system. We proceed by injecting transient and permanent faults into the network and report the subsequent number of messages, cache misses, and an indication of the resulting maximum failure message latency. Finally, we investigate the cost of multipath routing in our system.

V. RESULTS

We generated dictionaries with up to six entries for a source destination pair for the simulation results. The first four entries in the dictionary are generated with the link heuristic to avoid common redundant links when possible. The remaining two entries are generated using the node heuristic to avoid service interruptions when entire nodes fault.

A. Dictionary Size

TABLE II: Per node dictionary sizes for test topologies (B) label indicates the column reports in Bytes

Topology	# Entries	Max (B)	Mean (B)	Median(B)	StdDev(B)
AS 1221	624	30064	23858	23448	3476.3
AS 1239	1890	143976	73928	70496	14128
AS 1755	522	32016	21664	21336	3281.3
AS 3257	644	42824	27978	27192	5129.1
AS 3967	474	25672	18272	18040	2300.2
AS 6461	828	41632	30618	30832	4759.5
Synth 500	2180	113600	78100	78024	9819.3

The Table II gives the sizes of the dictionaries loaded with our test parameters including path redundancy and requiring that each router be able to act as an egress. The dictionaries can vary in size because of the distance between the nodes. In AS 3257 you can see that with a graph with 162 nodes and 656 directed edges the maximum dictionary size is roughly 43 Kbytes which is a small memory footprint. With the larger Synth 500 topology with 546 nodes and 2200 directed edges the maximum dictionary size is roughly 114 Kbytes, also easily fitting in memory. This result shows that dictionary sizes are scalable to large networks.

B. Link Usage Notification Overhead

With dictionaries that are not guaranteed to be consistent, our protocol requires that ingress nodes go through a link usage notification exchange when first using a route. In Figure 3 and Figure 4 show that the number of ingress nodes using a path is bounded at the maximum number of nodes. There are significant numbers of nodes with both relatively low numbers of path reservations and links with near the complete set of nodes. Either way, the storage footprint for link usage notifications is scalable.

C. Messaging After Unidirectional Link Failure

Next, we tested dictionary routing by injecting a number of failures into the network. We begin by randomly knocking out directed links in both AS 3257 and Synth 500. Figure 5 and Figure 6 shows when the injected errors are permanent meaning that each failure causes a link to be down for the duration of the simulation. Note, the jump in the plot which shows the number of failure scenarios in which the protocol was unable to use a local update and had to flood the network to update the dictionary entries. In the dictionary misses, the network is still connected with these failures, the router in control of the dictionary does not have any backup dictionary entries leading from the controlling node to at least one of the ingress nodes.

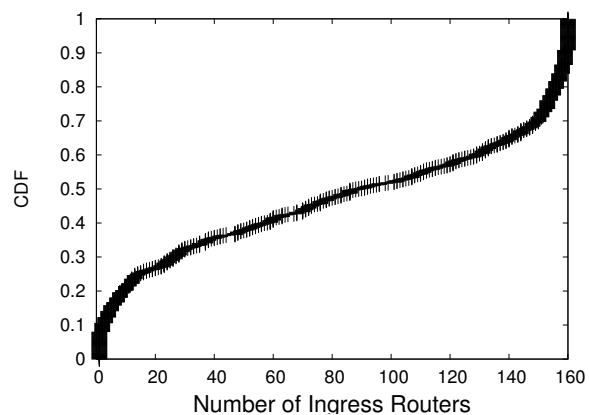


Fig. 3: Number of ingress routers with a link usage notification per link for AS 3257

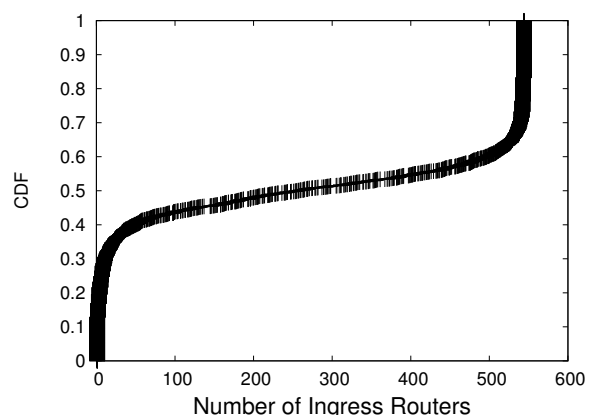


Fig. 4: Number of ingress routers with a link usage notification per link for Synth 500

Figure 7 and Figure 8 show the versatility of our protocol with transient outages. The transient failures show the performance gain in the number of messages in comparison to a link state protocol, which we have modeled to flood every link in the network with an update message in order to keep the topology database consistent. Path compression can be seen to help even when the link updates many routers with the majority of the error messages after a failure drastically below that of the link state algorithm. As seen in the failure usage scenarios, the directed messaging combined with the link usage notification enforces fault isolation.

These scenarios show that dictionary routing can respond well to dynamic changes in the underlying network. Our protocol can cut the number of messages required to reroute traffic after a failure by 66% when compared to OSPF in the common case.

D. Cache Misses

We measured the number of cache misses that we observed during the failure scenarios in Table III. A cache miss occurs when a router prompts the dictionary for a new route from source to destination and the dictionary does not have an

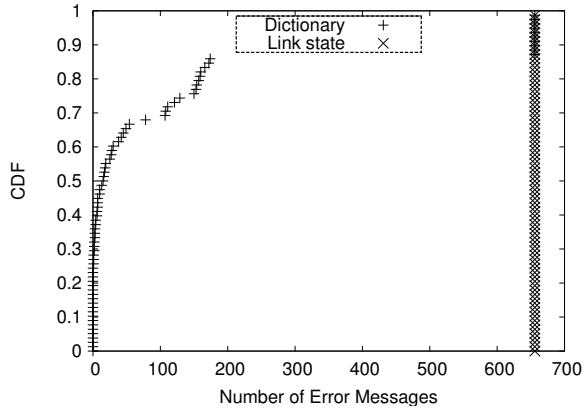


Fig. 5: Failure messages sent over 78 permanent directed link outages for AS 3257

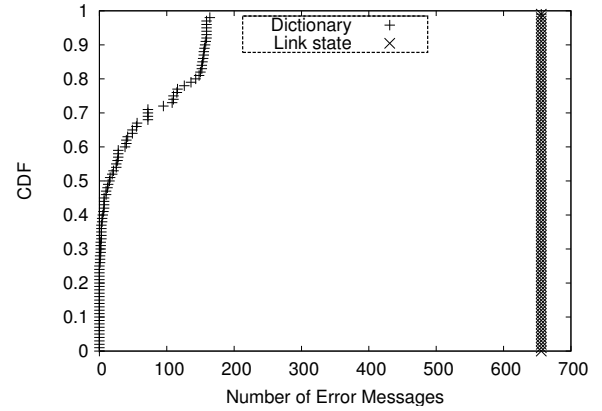


Fig. 7: Failure messages sent over 100 transient directed link outages for AS 3257

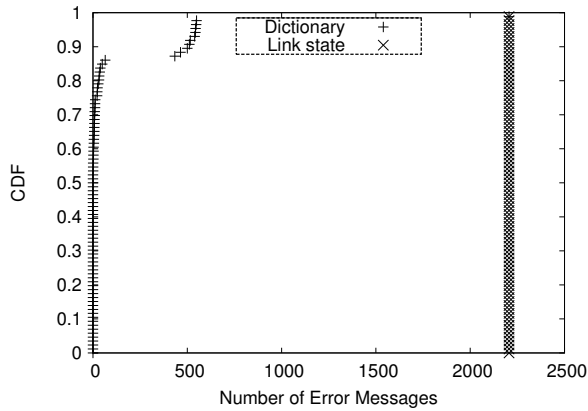


Fig. 6: Failure messages sent over 86 permanent directed link outages for Synth 500

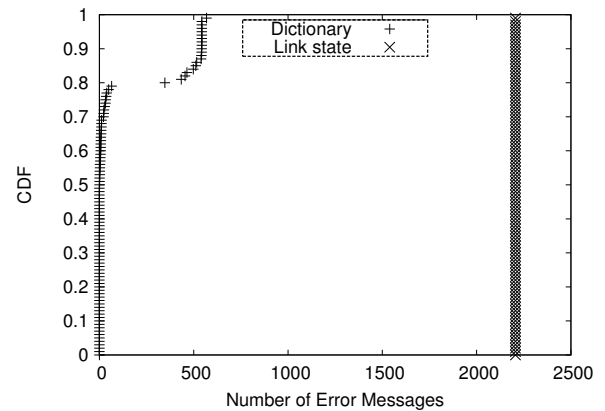


Fig. 8: Failure messages sent over 100 transient directed link outages for Synth 500

active entry available. In our scenario the cache miss requires the node to query the controller. With the full topology information for the network, the controller can recompute the required paths and refresh the dictionary entries. Cache misses are shown to be rare, indicating that the dictionary approach is robust even with multiple link failures.

TABLE III: Dictionary cache misses with 50 failures

Scenario	Number of Trials	Scenarios with cache miss
AS 3257 Permanent	50	17
Synth 500 Permanent	50	2
AS 3257 Transient	50	5
Synth 500 Transient	50	0

E. Maximum Failure Latency

We compare the maximum latency of a flooding based algorithm to the maximum latency of our compressed paths. To do this we randomly knock out an edge and then compute the latency required for the messages to get from the sender of the failure message to all of the nodes that are required by the protocol to be informed of the loss of the link. We do not consider the latency incurred by the routers along the path, we

report only an estimate of the latency due to the transmission times on the links. We see an expected improvement in latency due to the fact that only sources in the downed links source reservation list need to be identified. Before the failure notice, a link state algorithm will have to detect failure while the *dictionary routing* will have to detect failure and create the failure packet. After receiving a failure notice, a router using our framework will most likely only have to perform a look up to find a new path avoiding the downed link from the dictionary. The number reported for latency is the sum of the link weights an error packet would traverse. A link state algorithm will flood a failure message and perform the shortest path computation before it can update its forwarding tables.

F. Multipath Costs

Next, we demonstrate the overhead of activating the top 4 dictionary entries for all source destination pairs in each of the dictionaries when compared to 1 active dictionary entry in regards to the link usage notification cost Figure 9. The figure shows that the number of unique sources using the path grows with the the number of active entries. With the number of unique sources approaching the total number of sources in

TABLE IV: Average latency from 50 trials of failure messages after random outages where dictionary routing uses local look ups to recover, rounded sum of link weights reported.

Description	Dictionary Latency	Flooding Latency
AS 1221	9	16
AS 1239	24	37
AS 1755	27	35
AS 3257	23	35
AS 3967	38	50
AS 6461	21	33
Synth 500	35	60

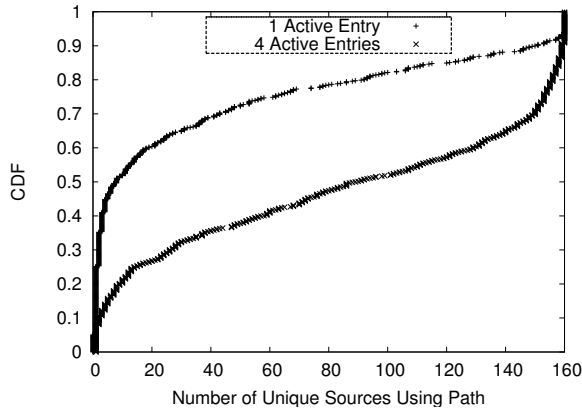


Fig. 9: Comparison of the overhead of path reservation for 1 active entry in a dictionary compared to 4 active entries.

the network, the number of messages sent as a result of a failure approaches that of a flooding broadcast.

VI. CONCLUSION

In this paper we describe *dictionary routing* a clean-slate approach to intra-domain routing based on precomputation. Our analysis shows that a dictionary-based approach requires a modest memory overhead on the routers. We conducted a simulation study using synthetic and real world topologies in which we assess the overhead and performance of dictionary routing. Our results show that dictionary routing is highly scalable in terms of drastically reducing the messaging overhead of flooding link-state protocols after a failure. These results demonstrate the feasibility of dictionary routing and make the case that it would provide important new capabilities versus standard link-state protocols. In our on-going work we are investigating the possibility of distributed dictionary computation and heuristics that improve cache performance.

REFERENCES

- [1] J. Moy, "OSPF Version 2," RFC 2328, April 1998.
- [2] A. Nucci, S. Bhattacharyya, N. Taft, and C. Diot, "IGP Link Weight Assignment for Operational Tier-1 Backbones," *IEEE/ACM Transactions on Networking*, vol. 15, no. 4, August 2007.
- [3] C. Labovitz, A. Ahuja, and F. Jahanian, "Experimental Study of Internet Stability and Wide-Area Network Failures," in *Proceedings of the International Symposium on Fault-Tolerant Computing*, Madison, WI, October 1999.
- [4] V. Paxson, "End-to-End Routing Behavior in the Internet," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, October 1997.

- [5] J. He and J. Rexford, "Toward Internet-wide Multipath Routing," *IEEE Network*, vol. 22, no. 2, March-April 2008.
- [6] NSF, "NeTS FIND Initiative," <http://www.nets-find.net>, January 2009.
- [7] J. He, J. Rexford, and M. Chiang, "Don't Optimize Existing Protocols, Design Optimizable Protocols," *ACM SIGCOMM Computer Communications Review*, vol. 37, no. 3, July 2007.
- [8] A. Shaikh, J. Rexford, and K. Shin, "Efficient Precomputation of Quality-of-Service Routes," in *Proceedings of the Workshop on Network and Operating Systems Support for Digital Audio and Video*, July 1998.
- [9] J. Sommers, P. Barford, N. Duffield, and A. Ron, "Accurate and Efficient Network-wide SLA Compliance Monitoring," in *Proceedings of ACM SIGCOMM*, Kyoto, Japan, September 2007.
- [10] D. Applegate and E. Cohen, "Making Intra-Domain Routing Robust to Changing and Uncertain Traffic Demands: Understanding Fundamental Tradeoffs," in *Proceedings of ACM SIGCOMM*, Karlsruhe, Germany, August 2003.
- [11] C. Zhang, J. Kurose, D. Towsley, Z. Ge, and Y. Liu, "Optimal Routing with Multiple Traffic Matrices Tradeoff Between Average Case and Worst Case Performance," in *Proceedings of IEEE ICNP '05*, Boston, MA, November 2005.
- [12] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg, "COPE: Traffic Engineering in Dynamic Networks," in *Proceedings of ACM SIGCOMM*, Pisa, Italy, September 2006.
- [13] M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," in *Proceedings of ACM SIGCOMM*, Kyoto, Japan, August 2007.
- [14] M. Casado, T. Garfinkel, A. Akella, M. Freedman, D. Boneh, N. McKeown, and S. Shenker, "SANE: A Protection Architecture for Enterprise Networks," in *Unix Security*, Vancouver BC, Canada, August 2006.
- [15] A. Greenberg, G. Hjalmtysson, D. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "Clean Slate 4D Approach to Network Control and Management," in *ACM SIGCOMM Computer Communication Review*, October 2005.
- [16] A. Kvalbein, A. Hansen, T. Cicic, S. Gjessing, and O. Lysne, "Fast IP Network Recovery using Multiple Routing Configurations," in *Infocom*, Barcelona, Spain, April 2006.
- [17] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault, "Multi-Topology (MT) Routing in OSPF," RFC 4915, June 2007.
- [18] E. Rosen, "MPLS Architecture," RFC 3031, January 2001.
- [19] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels," RFC 3209, December 2001.
- [20] V. Sharma and F. Hellstrand, "Framework for Multi-Protocol Label Switching (MPLS)-based Recovery," RFC 3469, February 2003.
- [21] D. Xu, M. Chiang, and J. Rexford, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," *Proc. IEEE INFOCOM*, April 2008.
- [22] J. Rexford, "Optimization in IP networks," *Handbook of Optimization in Telecommunications Springer Science Business Media*, February 2006.
- [23] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "MATE: MPLS adaptive traffic engineering," in *Proc. IEEE INFOCOM*, Anchorage, AK, April 2001.
- [24] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica, "Achieving Convergence-Free Routing using Failure-Carrying Packets," in *Proceedings of ACM SIGCOMM*, Kyoto, Japan, August 2007.
- [25] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a Routing Control Platform," in *Proc. Networked Systems Design and Implementation*, May 2005.
- [26] X. Yang and D. Wetherall, "Source Selectable Path Diversity via Routing Deflections," in *Proceedings of ACM SIGCOMM*, Pisa, Italy, September 2006.
- [27] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP Topologies with Rocketfuel," in *Proceedings of ACM SIGCOMM*, 2002.
- [28] P. Mahadevan, C. Hubble, D. Krioukov, B. Huffaker, and A. Vahdat, "Orbis: Rescaling Degree Correlations to Generate Annotated Internet Topologies," in *Proceedings of ACM SIGCOMM*, Kyoto, Japan, August 2007.