Sequencing of Configuration Operations for IP Networks

P. Krishnan – ISPsoft, Inc.
Tejas Naik – Bell Laboratories[1]
Ganesan Ramu – CoSine Comm., Inc.
Roshan Sequeira – ISPsoft, Inc.

ABSTRACT

When configuring an IP network, changes may need to be made to several devices as part of one configuration operation. Given an in-band access to the devices where data and control packets flow on the same network, it becomes imperative to sequence the changes intelligently to retain connectivity to the devices that need to be reconfigured. In this paper, we present techniques to sequence configuration operations for IP networks. We describe our experiences with implementation of the techniques, and experimentally evaluate the usefulness of the proposed techniques. We demonstrate that automatic sequencing is valuable in IP network configuration.

Introduction

A basic operation performed during network operations and management is the configuration of the network. Many configuration and reconfiguration operations are typically performed from a remote management station. To maintain consistency, several devices may need to be updated as part of one configuration operation. To complete the configuration action, the devices to be updated must continue to be reachable from the management station. Reconfiguring some devices may jeopardize the connectivity to other devices that need to be configured. *Sequencing* is the process of determining the order of updates to ensure a successful configuration.

In this paper, we present one of the first studies of automated sequencing in the context of router[2] reconfigurations in IP networks. Many deployments today use an in-band access to the routers for configuration, where the control and data packets flow on the same network[3]. In this case, configuration of a router is done through a telnet to the router. Control (configuration information) and data go over the same physical and logical network, directed by the same routing tables. Parameters that an administrator can reconfigure include what we call *critical* and *non-critical* parameters. Changing critical parameters affects how the router computes and exchanges routing information with other routers, and consequently threatens continued connectivity to the router and other routers. Examples of critical parameters include routing protocol parameters, like the Open Shortest Path First (OSPF) [10, 12] *hello interval*, *router dead interval*, *authentication*, and *authentication key* parameters, fundamental parameters like IP addresses and subnet masks, some access list configurations, etc. Non-critical parameters, on the other hand, do not affect connectivity. Examples include parameters like the OSPF *maximum paths* and *cost* parameters.

One can see that routers can be updated in any arbitrary order while changing non-critical parameters, or when an out-of-band access is available to all the routers. In this case, by out-of-band we mean access through the console or modem ports of the router. However, the order in which the updates are committed (i.e., sequencing) becomes important. Administrators currently try to deduce a sequence manually from their knowledge of the network topology and their understanding of protocols. No automated techniques have been published for sequencing.

In this paper, we study the problem of sequencing when updating critical parameters, and present automated methods to compute an order in which to update the routers. We have implemented the techniques in Java, and conducted experiments that quantify the usefulness of our techniques and the convenience they can provide to the administrator. We demonstrate that our automated sequencing techniques speed up network configuration and make it more reliable.

The rest of the paper is organized as follows. The sequencing problem is explored in more detail in the next section. Our algorithms for sequencing router configurations are presented thereafter. The implementation issues we encountered are presented in the implementation section which is followed by a section on experiments and results. We then conclude.

Sequencing: The Problem and Assumptions

To illustrate the problem of sequencing more concretely, consider the network in Figure 1. Let us

[1]Portions of this work were performed while all authors were at Bell Labs.

[2]We use the term *router* to refer to any remotely configurable network element.

[3]A few routers in the network may have an out-of-band connectivity through the modem or console port of the router. Some establishments provide an out-of-band access to all routers in their network.

assume that all depicted routers are running the OSPF routing protocol, and are in the same OSPF Area. (See Appendix 2 for a brief description of OSPF.) Let us assume that we want to change the OSPF authentication parameter. The OSPF authentication parameter should be set consistently for all routers in an OSPF area to ensure that they exchange routing information. If during configuration, we update router *R* before all routers in the set *S*, there is a possibility that routers in set *S* may become unreachable from the network operations center (NOC)[4]. Sequencing ensures that the routers that remain to be updated will continue to be reachable. In our example, updating all routers in set *S* before router *R* is a viable strategy.

In our study of sequencing during IP network configuration, we make certain assumptions. We assume that the changes are being made to the routers from a remote centralized network operations center and that all routers are initially reachable. Further, the changes to be made are correct; i.e., if the desired changes are committed to the routers, the routing will stabilize soon enough, based on the routing protocol behavior, and to a state as desired by the administrator. We refer to this as the *correctness assumption*. The reachability of routers can be affected by several factors, some of which are outside the scope of any software tool (e.g., power failure at a router). Such issues are not considered as part of this study of sequencing. We assume that any unreachability during the configuration operation is only caused due to the configuration changes being made by the administrator.

[4]We use the terms *network operations center* and *management station* interchangeably in the paper.

Situations may arise where no sequencing is possible. Remote configuration of a set of routers could be theoretically impossible, under some conditions, mostly stemming from asymmetric routing. Consider, for example, the situation in Figure 2.

Assume that the routes are set up (e.g., via static routes) so that all packets can only move clockwise around the circle *C* of routers. If a critical update is made to any router in the circle of routers, one of the paths (either to the router from the management station, or from the router to the management station) is broken, disabling the ability to reach and configure the other routers in the circle. Such a routing topology is unlikely in practice, since most people use routing protocols that will adapt to network change. However, this gives some insight into the theoretical limits of remote in-band configuration and sequencing in general. For simplicity, we assume in the following discussion that the routing in the network is symmetric (although such a restriction is not really needed for all our techniques), and elaborate more on such situations later in a section on asymmetry in routing.

An alternative approach to the problem of sequencing is to have built-in support in all routers in the network for scheduling configuration changes. This means that the software running on the routers (e.g., the Cisco IOS) must support the scheduling feature. In this scenario, the required configuration changes are uploaded to the router and a timer is set which determines when the changes are executed. Ongoing work in the area of scheduling and scripting management information bases [8, 9] can aid in this problem. However, routers in the marketplace do not

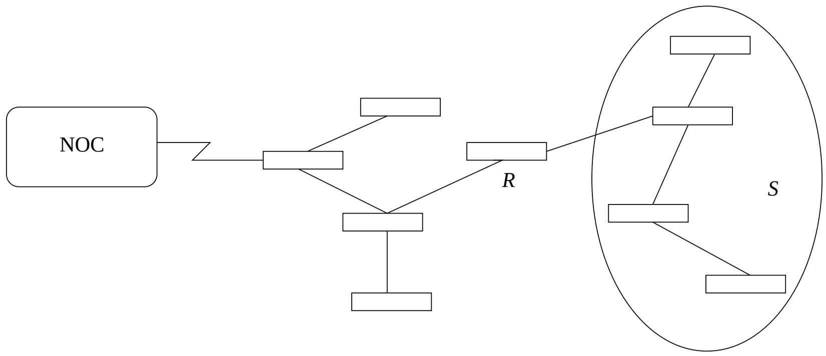


Figure 1: A network of routers and its network operations center (NOC).

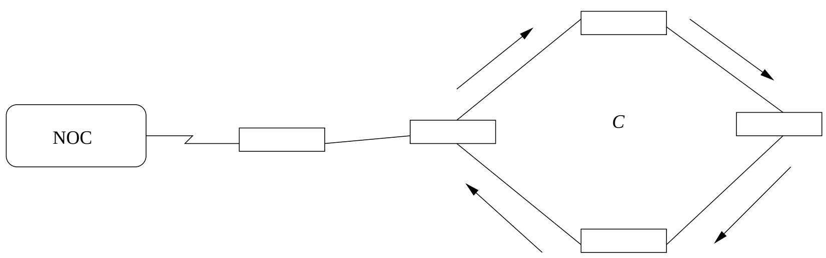


Figure 2: Circular dependencies and sequencing.

provide such a feature. Given that in-band remote configuration is becoming increasingly popular for IP networks, developing smart automated sequencing techniques is essential.

Context for our Study

This study of sequencing arose in the context of an IP network configuration tool [5, 13] that was built at Bell Labs. The tool helps in configuring IP routers, treating them as a part of the network, rather than considering them as stand-alone devices. The tool ensures consistency of configuration and updates several routers as part of a configuration operation, if necessary. The protocol used to contact the routers for configuration is orthogonal to the issue of sequencing. Although there are several proposed standards for management that can be used for configuration [3, 4], most deployed routers today provide a command-line interface for configuration. Configuration commands typically take effect as soon as they are executed.

Our network configuration tool provides a GUI to the administrator for configuration, and hides all its interactions with the network and the routers from the administrator. To interact with the routers for configuration, the tool stores router passwords in an encrypted form in its database, and uses these passwords to login to the routers.

Before we present our sequencing techniques, we introduce the notion of access to a router via *indirect telnet*.

Access via Indirect Telnet

An indirect telnet is essentially a proxy telnet. The basic idea in indirect telnet is to access a (possibly unreachable) router *R* through an intermediate router *N*, by first telnet'ing to the intermediate router *N* and then initiating a telnet to router *R* from router *N*.

The usefulness of indirect telnet is best illustrated by the case where routers *R* and *N* have a directly connected interface. Since directly connected IP-configured interfaces are visible to each other, if router *N* is reachable, then we can access router *R* through its neighbor *N*. Such a technique for reaching routers is sometimes used by administrators while configuring routers manually. The concept can be extended to hop through several intermediate nodes en-route to a final destination router. Our tool implements an automated 1-hop indirect telnet scheme.

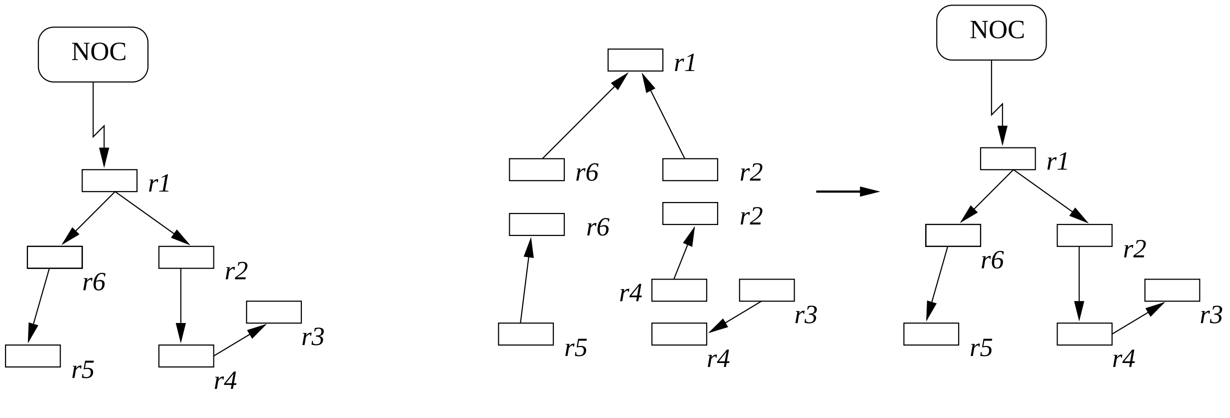
Sequencing Techniques

We present two main techniques for sequencing. The treegen method is a tree generation family of techniques. The treegen method deduces the routing topology to determine the order for updates. The reachable method is a sequential technique that uses indirect telnet (described in in the previous section) in an interesting way to perform sequencing. For the following discussion, we refer to the technique that chooses to update routers in no particular order (i.e., effectively, randomly) as the heuristic random; unlike our techniques, the random technique does not aim to update all the routers.

The treegen Family of Techniques

The basic purpose of the treegen family of techniques is to deduce the routing path tree to the nodes that need to be updated, and perform the configurations bottom-up in this tree. Such a tree can be built by performing traceroutes to the routers that need to be updated. The output of the traceroutes can be put together (using commonly known techniques, e.g., as done in [2, 7] for a different problem) to obtain a directed tree of the routers which depicts how packets are routed to these routers.

A simple such traceroute tree is shown in Figure 3a; it may have been constructed through a traceroute from the network operations center (NOC) to router r_5 followed by traceroutes to routers r_2 and r_3. The technique assumes that the part of the routing tree not yet updated will be stable for the time required to perform the configurations. Multiple routes to the same host may exist, but choosing one is generally sufficient.



(a) Forward traceroute tree (b) Getting the reverse traceroute tree

Figure 3: Forward and reverse traceroute trees.

Once the tree is obtained, the sequence is specified from the leaves to the root of the tree; i.e., all nodes reachable from a node must appear before the node in the sequencing order. Such an order can be obtained from a tree, e.g., by using depth first search techniques [1]. In Figure 3a, r_3, r_4, r_2, r_5, r_6, r_1, and r_5, r_3, r_6, r_4, r_2, r_1 are valid sequences. We call our technique described above as the traceroute method.

While doing a traceroute, if there is no response to a probe within the specified timeout, a "$*$" output is observed. It is not uncommon to see "$*$" outputs for all probes, especially at higher time to live values (i.e., for probes reaching farther from the host). This slows down the traceroute method and also makes it less reliable. Routers on the network with out-of-band connectivity could be exploited in the sequencing process. We now present some enhancements to our traceroute method that deal with some of these issues.

The **reverse-traceroute** *Method*

If we have access to the routers (i.e., the router passwords, which are needed for configuration), we can perform a traceroute *from* the routers to the management station rather than to the routers from the management station. These traceroutes can be put together to deduce the tree from the NOC to the routers. If the routing is symmetric, such a method that does traceroutes from the router would give the same tree as the generic traceroute method. In this case, however, we have the advantage that we can do a restricted traceroute, possibly looking only at the next hop on the routing path from the router to the management station. These {router, next hop} pairs can be put together to get the full tree. For the example in 3a, the method to obtain the reverse traceroute tree by conducting one-hop traceroutes from each of the routers is depicted in Figure3b.

In theory, we need to examine the traceroute from each router until we hit a router that also needs to be updated. This ensures that we have the complete dependency between the routers that need to be updated. In most cases, however, the set of routers to be updated is contiguous and looking at the next hop suffices. If access to all routers exists, the routing table entries can be examined in lieu of a traceroute. A traceroute is a simple way to get the route information.

The **forward-reverse** *Method*

The safer method for constructing the tree is to perform both the traceroute and reverse-traceroute methods. This will avoid depending on the symmetric routing assumption. If we get a tree (or a directed acyclic graph [1]) by putting the trees derived from the traceroute and the reverse-traceroute methods together, the sequencing order can be determined. A cycle indicates a possible generic problem in determining a sequencing order. If such a situation is due to multiple routing paths in the network, there is no problem, since the routing path will readjust, if needed. If the situation is due to forced asymmetric routing (as described in Figure 2), there is a problem. However, this is a pathological example and most installations do not design their networks this way.

Other Generalizations

There can be several possible start nodes for initiating the traceroutes. It is necessary to have access (i.e., login) to each of the routers and machines from which a traceroute is to be performed. Furthermore, there must be a path to the start nodes that does not go through other nodes that are to be updated. The following nodes are examples of candidates for start nodes: the machine in the network operations center, OSPF area border routers, and routers with out-of-band (e.g., modem or console) connections.

Let S be the set of nodes to which changes need to be made. Let SN be the non-empty set of possible start nodes. The output of the sequencing method is an ordered list of elements: (s_i, sn_j), where s_i is a node from set S and sn_j is a node from set SN. The implication is that configuration changes must be made in the order specified by the list, and a telnet to node s_i must be accomplished from node sn_j (i.e., via an indirect telnet strategy).

Conceptually, we build *traceroute trees* from each of the start nodes. Each tree can either encompass all nodes in the network, or only the nodes not covered by already created trees. The dependencies can be computed from these sets of trees using standard graph theoretic approaches as in [1]; we omit these details from this paper. This generalization lends to interesting extensions of the basic approach, and techniques that mix pure in-band and pure out-of-band configuration approaches.

Optimizations

The treegen family of techniques lend themselves to various optimizations. Nodes that do not depend on each other can be updated in parallel to improve performance. The traceroutes in the reverse-traceroute method can be done in parallel to improve performance. In some cases, instead of deducing the routing topology by examining the routing tables or performing traceroutes, an analysis of the physical topology can indicate the routing topology (e.g., a simple line network).

The reachable Technique

Unlike the treegen family of techniques, the reachable technique merges the processes of determining the update order and the router configuration updates. At the ith iteration, the reachable technique determines the next router to update, updates this router, and then determines the $i + 1$st router to update. The choice of the next router depends on the current state of the network. The reachable technique assumes that a one-hop indirect telnet access is available to reach a router.

The reachable algorithm updates reachable routers via telnet. If there are no remaining reachable routers, the process switches to indirect telnet via a router whose neighbor is reachable. (Indirect telnet was explained in an earlier section.) More formally, let S denote the set of all routers to be reconfigured, let U denote the set of already reconfigured routers, let TBU denote the set of routers that remain to be reconfigured, and let R denote the set of routers that are currently directly reachable from the management station. Initially, $U = \Phi$, and $TBU = R = S$. The set R can be determined at any point in time via pings from the management station, and TBU is always equal to $S - U$. Algorithm reachable uses the following logic to determine the next router to reconfigure. If there is a router to be reconfigured that is directly reachable (i.e., in set R), it updates that router. If there is no directly reachable router (i.e., $R \bigcap TBU = \Phi$), reachable reconfigures via indirect telnet a router from set TBU whose neighbor is reachable (i.e., the neighbor is in set R). Otherwise, it declares failure. After configuring a router, it updates sets U and TBU, recomputes set R, and continues, as long as $TBU \neq \Phi$.

An interesting property of the reachable algorithm is that under most conditions, it will not fail. In other words, it will always find a router to update. We formalize this notion in Appendix 1.

Implementation

We implemented the techniques described earlier to obtain measurements on their efficacy. The implementation was done in Java. For our prototype version, we adapted a Java telnet implementation [6] based on Java sockets to contact the routers. The router passwords were available to log into the router and perform configuration. We wrote our sequencing routines as part of the IPNC tool [5, 13]. Our implementation allowed us to determine information about neighbors to the routers, and the ability to do one-hop indirect telnet, perform traceroute from the routers, generate the correct router commands and perform the required interactions with the routers.

The treegen Technique

We implemented the reverse-traceroute method from the treegen family of techniques. We implemented two variants of the method. In the first one, which we call trace-parallel, after the order is determined, the routers are updated with maximum possible parallelism; in other words, at any stage, all leaves of the tree are updated in parallel. In the second variant that we call trace-seq, the routers are updated sequentially. In both cases, the traceroutes from the routers are executed in parallel to obtain the order of update speedily. We restricted ourselves to three hops of output from traceroute. The routers we used did not allow an easy (non-interactive) way of specifying the maximum number of hops, and this had to be worked in from our implementation.

The reachable Technique

Implementation of the reachable technique required indirect telnet support. We performed pings in sequence, and hope to move to a parallelized implementation soon. Coming up with a suitable timeout to account for routing stabilization is non-trivial; we approximated with retrying the configuration operation after waiting five seconds upon a failure, and this worked for us. We elaborate more on this later in a section on Running Time Measurements.

Experiments and Results

Experiments were conducted on networks of various topologies. Measurements on the three basic topologies (line, tree, and circle) shown in Figure 4 are reported below. The other topologies we experimented with were made up of these basic topologies, as are most networks. The main results from other topologies we tested were in agreement with what we present here.

The routers in our experiments were Cisco routers (25xx, 26xx, 36xx, and 40xx series), running various versions of IOS (all version 11.3 or higher). The interfaces were a mix of ethernet, high speed serial, serial, and token ring interfaces. For our experiments, we enabled the routers to run OSPF, put all routers in the same OSPF area, and changed the OSPF area authentication parameter on the routers. (See Appendix 2 for a brief description of OSPF.) In this case, a change at a router does not immediately make itself or other routers unreachable. The protocol takes

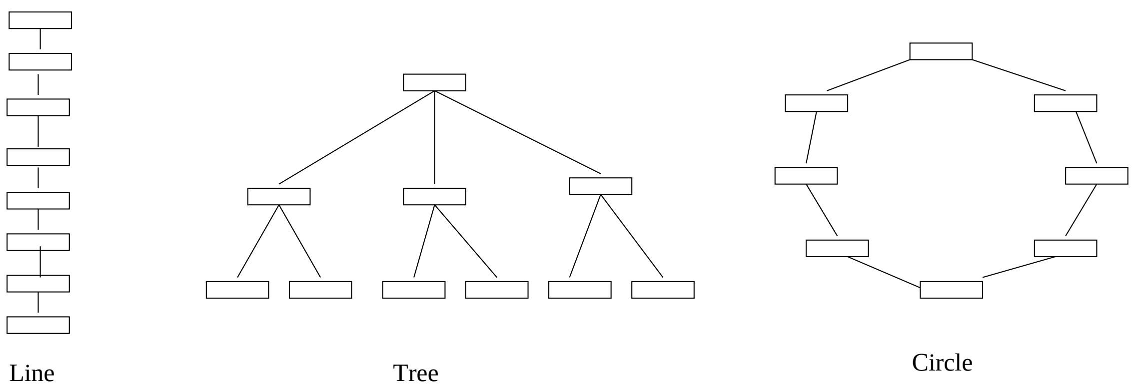


Figure 4: Network topologies for reported experiments.

some time to settle depending on other parameters (like hello and dead intervals) that we could change. Keeping the hello interval fixed at a low value (e.g., 1-2 seconds), a high value of the dead interval holds the network routing virtually unchanged for a long time, while a low value for the dead interval propagates the effect of the parameter changes quickly. For each of the methods random, trace-seq, trace-parallel, and reachable, we obtained measurements on (1) the number of routers that were successfully configured, (2) the time it took to determine the sequence order and (3) the time it took to perform the configuration. In the case of reachable since the sequencing is interwoven with the configuration, we only obtained one number: the total time to complete the reconfiguration. We now present our results under various categories.

Number of Routers Updated

The trace-seq, trade-parallel, and reachable techniques were always successful in updating all the routers. This was true for all values of hello/dead interval we tested.

An interesting situation in this case is with the random technique. With random, since the routers are chosen in an arbitrary order and no sequencing is performed, some of the routers may fail to get configured.

Depending on the hello and dead intervals and the random order, up to 60% of the routers failed to get updated by random. An illustration of this effect is shown in Figure 5, which presents a plot for a random sequence. The plot in the figure must be taken with some caution. By using automated techniques rather than manual interaction with the router for configuration, quick router configurations were possible. (More details of the time for configuration appear below.) The effect from Figure 5 may be vastly different for manual configurations in that many more routers may fail to get updated, since each router takes longer to be updated when done manually. Second, the random

Method	Line (8 routers)			Tree (10 routers)			Circle (8 routers)		
	t_{seq}	t_{conf}	t_{total}	t_{seq}	t_{conf}	t_{total}	t_{seq}	t_{conf}	t_{total}
trace-seq	16.70	18.92	35.62	27.86	19.63	47.49	23.52	15.06	38.58
trace-parallel	18.16	17.81	35.97	27.87	6.14	34.01	19.83	9.82	29.65
reachable	–	–	58.92	–	–	69.46	–	–	56.05

Table 1: Average time to perform sequencing and configuration for the three topologies and the three techniques; t_{seq} is the time to determine the sequence, t_{conf} is the time to do configuration, and t_{total} is the total time taken. The times are for all routers in the network to be updated. All units are in seconds.

Method	Line (8 routers)			Tree (10 routers)			Circle (8 routers)		
	σ_{seq}	σ_{conf}	σ_{total}	σ_{seq}	σ_{conf}	σ_{total}	σ_{seq}	σ_{conf}	σ_{total}
trace-seq	2.31	0.52	1.79	3.43	0.41	3.35	0.06	0.05	0.11
trace-parallel	0.04	0.65	0.61	3.49	0.12	3.59	4.61	0.15	4.69
reachable	–	–	0.75	–	–	1.15	–	–	0.74

Table 2: Standard deviations in time to perform sequencing and configuration for the three topologies and the three techniques; σ_{seq} is the standard deviation in the time to determine the sequence, σ_{conf} is the standard deviation in the time to do configuration, and σ_{total} is the standard deviation in the total time taken. The times are for all routers in the network to be updated.

Method	Line (per router)			Tree (per router)			Circle (per router)		
	t_{seq}	t_{conf}	t_{total}	t_{seq}	t_{conf}	t_{total}	t_{seq}	t_{conf}	t_{total}
trace-seq	2.09	2.37	4.45	2.79	1.96	4.75	2.94	1.88	4.82
trace-parallel	2.27	2.23	4.50	2.79	0.61	3.40	2.48	1.23	3.71
reachable	–	–	7.37	–	–	6.95	–	–	7.01

Table 3: Average time (per router) to perform sequencing and configuration for the three topologies and the three techniques; t_{seq} is the time to determine the sequence, t_{conf} is the time to do configuration, and t_{total} is the total time taken. These numbers are derived from Table 1 by dividing the total time by the number of routers, and is presented here for convenience. All units are in seconds.

order is significant, and the plot implicitly says more about the time for the routing to settle rather than about the random technique itself. Third, the graph may look different if a parameter other than dead interval were chosen for the *x* axis. We chose the dead interval since it is a good control variable for how soon the configuration changes affect the routing.

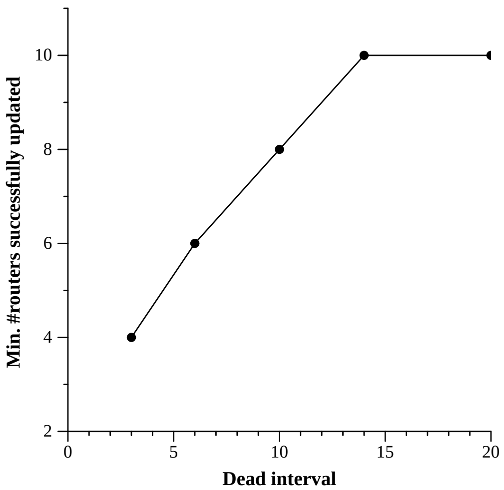


Figure 5: Minimum number of routers (out of 10) updated by random as a function of OSPF dead interval using a hello-interval of 1 second on the tree network.

Running Time Measurements

Tables 1-3 show various statistics related to the time taken by our techniques for determining the sequence and performing the configuration changes. All values are derived from 3-4 runs. (We did many more runs, but they used different topologies and different number of routers. We observed similar results from those experiments as reported here.) Table 1 shows the total time for all routers in the network to be updated and Table 2 shows the standard deviation of the observed times. For convenience, we divide the total time numbers from Table 1 by the number of routers in the network to get per router timing numbers in Table 3. The topologies have different number of routers (eight each for the line and circle, and ten for the tree); in addition, the routers and interfaces used in the topologies were also slightly different.

The first interesting result is that automation makes the total time taken to do the configuration very small (on the average, 0.6-2.23 seconds per router for trace-parallel as seen from Table 3). This enables rapid reconfiguration and makes our tool very useful. As a point of reference, our (informal) study shows that manual reconfiguration of authentication by experienced administrators may take about 30-45 seconds per router (*excluding* times for sequencing), resulting in 5-7.5 minutes for reconfiguring 10 routers. This estimate for manual configuration excludes the time to generate the configuration commands, but includes the time to type in the commands at the router's configuration prompt. In the case of OSPF authentication, this typically translates to 4-6 commands per router. In the automated case, the commands are automatically generated and sent to the router by our tool, and that time is included in the reported numbers.

The automation also lends to easy parallelization of the configuration operation, when possible. The performance improvement resulting from the parallelization is visible from the t_{conf} value for the trace-parallel technique, which shows a 34.6% and 68.8% improvement over the corresponding t_{conf} time for the trace-seq technique for the circle and tree topologies, respectively. No parallelism is possible in the case of the line. The traceroutes in the case of the trace-seq and trace-parallel methods were done in parallel, and the t_{seq} time is dominated by the amount of time it takes to get a 3-hop output from the traceroute done at the router (since all other processing on this output is minimal and in-memory). For random, when it is successful in completing the configuration, the $t_{conf} = t_{total}$ and is approximately the t_{conf} value for trace-seq. From the table, it is difficult to break up the time taken for configuration alone by reachable. The time for non-configuration activities (including pings, timeouts, and indirect telnet) for reachable can be estimated, however, by

t_{total}(reachable) − t_{conf}(trace-seq)/t_{total}(reachable)

to be 67-73% for the three topologies.

The comparatively low values for standard deviation from Table 2 suggest that the timing measurements did not vary significantly.

Interestingly, the timing values for trace-seq and trace-parallel were rather predictable and independent of the hello/dead interval values (and hence, the time taken for route changes). However, the same was not the case for reachable. The time for reachable depends on the order in which the routers are updated, and how soon the routes change. In our implementation of reachable, the order in which the routers were tried for update depended on the hash function used by Java (since the router names were initially put into a hash table). Figure 6 shows how the time for configuration for ignored: reachable changed with the dead interval in the network. The increase in time when the routes change more rapidly (i.e., at lower values of dead interval) comes from the increase in the number of indirect telnets performed and the timeouts when performing pings (when the routing is settling down). As mentioned in Appendix 1 and the implementation section, in our experiments, when there was a failure, retrying after a timeout of five seconds provided a router as per the condition in Proposition 1 in Appendix 1.

In rare circumstances (in two cases out of several tens of tests we ran), we observed that a ping to a router succeeded, but the routing changed between the ping's success and completion of the configuration. Such occurrences may be more frequent in some networks. However, it does not matter to the reachable technique from the point of view of correctness, since it assumes that the router was unreachable. It does, however, increase the running time of the technique.

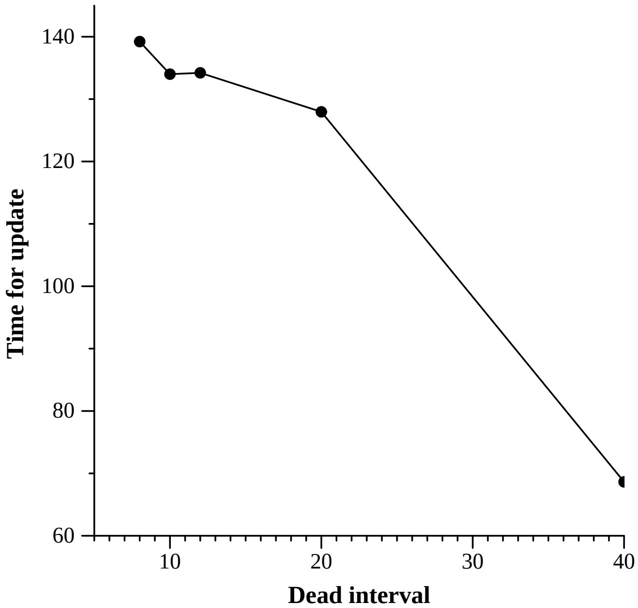


Figure 6: Time taken by reachable to do configuration as a function of the dead interval using a hello interval of 2 seconds on the tree network.

Asymmetry in Routing

In most of our topologies, the routing was symmetric. In the circle case, both paths around the circle could be taken by the packets. We did not explicitly manipulate the routing tables, and left the cost of both paths to be the same (for OSPF purposes). The dynamic nature of the routing combined with the sequencing techniques we used resulted in no configuration problems for the topology using our techniques. In our experimentation, we concentrated on topologies and scenarios that we felt were representative of real networks, and hence did not choose configurations like in Figure 2.

Conclusions

In this paper we have studied the problem of IP network configuration. In particular, for the first time, we have studied the issue of sequencing updates of critical parameters in a network of IP routers. We have presented several techniques to sequence the configuration of the routers to ensure continued connectivity to the routers during the time that it takes to perform the configuration. We have implemented our techniques, and have presented experimental results on the validity of the techniques and measurements of their performance. We have shown that automated configuration significantly speeds up time for configuration (from an estimated 45 seconds per router while done manually to 0.6-2.3 seconds when automated), and our sequencing techniques make configuration fast and reliable. Our automated techniques also make the configuration process convenient for the administrator. Some of our sequencing techniques are also part of the IPNC tool [5]. We believe that this significantly advances the state of the art in a field dominated by manual configuration of routers. We believe that such tools and techniques for automated network configuration will prove valuable to the users.

The network configuration tool with appropriate enhancements is currently available from Lucent Technologies (http://www.lucent.com) and ISPsoft, Inc. (http://www.ispsoft.com).

Acknowledgements

We would like to thank Subrata Mazumdar in particular, and other members of the IPNC team at Bell Labs for helpful discussions.

Author Information

P. Krishnan (who is called ‘‘Krishnan’’ or ‘‘PK’’) is currently with ISPsoft, Inc.; pk@ipsoft.com. He received his Ph.D. in Computer Science from Brown University, and his B. Tech in Computer Science from the Indian Institute of Technology, Delhi. Prior to joining ISPsoft, he was with the Networking Research Center at Bell Labs, Lucent Technologies. His research interests include IP networking and management, the development and analysis of algorithms, prefetching and caching, and mobile computing.

Tejas Naik is currently with Bell Labs, Lucent Technologies Inc. He received his Master’s degree in Computer Engineering from University of Southern California and B.E. in Electronics Engineering from South Gujarat University, India. Prior to joining Bell Labs, he was with QLogic Corporation. His research interests include IP network and services management and IP traffic management. Reach him electronically at tnaik@research.bell-labs.com.

Ganesan Ramu is currently with CoSine Communications Inc., Redwood City, CA. He received his B.S. in Computer Science and Engineering from University of Madras, Chennai, India. His prior work in Network Management area includes his association with the Networking Research Center at Bell Labs, Lucent Technologies and Network Management Business Unit, Cisco Systems. His current interest is in the analysis of various aspects of IP routing protocols implementation.

Roshan Sequeira holds a Bachelor’s Degree in Electronics and Communications Engineering from Mangalore University, India. He has worked for the Network Management Buisness Unit at Cisco Systems and the Network and Services Management Department at Bell Labs. He is currently with ISPsoft Inc.

Bibliography

[1] A. Aho, J.Hopcroft, J. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley Longman Inc., April 1978.

[2] C. Cunha, *Trace Analysis and Its Applications to Performance Enhancements of Distributed Information Systems,* Ph.D. thesis, Boston University, 1997.

[3] J. Case, M. Fedor, M. Schoffstall, and J. Davin, *A Simple Network Management Protocol,* IETF Network Working Group, STD 15, RFC 1157, May 1990.

[4] J. Case, R. Mundy, D. Partain, B. Stewart, *Introduction to Version 3 of the Internet-standard Network Management Framework,* IETF Network Working Group, RFC 2570.

[5] The IP Network Configurator, http://www.lucent.com/OS/ipnc.html .

[6] M. Jugel and M. Meisner, *The Java Telnet Applet,* http://www.first.gmd.de/persons/leo/java/Telnet/index.download.html .

[7] P. Krishnan, D. Raz and Y. Shavitt, "The Cache Location Problem," to appear in the *IEEE Transactions on Networks*.

[8] D. Levi and J. Schoenwaelder, "Definitions of Managed Objects for Scheduling Management Operations," IETF Network Working Group, *RFC 2591*, May 1999.

[9] D. Levi and J. Schoenwaelder, "Definitions of Managed Objects for the Delegation of Management Scripts," IETF Network Working Group, *RFC 2592*, May 1999.

[10] John T. Moy, *OSPF: Anatomy of an Internet Routing Protocol,* Addison-Wesley Longman Inc., December 1997.

[11] John T. Moy, "OSPF version 2," *Internet RFC 2328 Standard*, April 1998.

[12] W. R. Parkhurst, *Cisco Router OSPF: Design and Implementation Guide,* McGraw Hill, July 1998.

[13] B. Sugla and P. Krishnan, "IP Network Configurator: Advancing the State of the Art in IP Network Deployment and Operations," *Lucent Technologies' whitepaper*, November 1998.

Appendix 1: Success Criterion for the reachable Algorithm

The success criterion for the reachable algorithm is based on the following *communication property.* In most cases, nodes that have already been reconfigured, and that are (topologically/logically) connected will start communicating with each other, and be reachable from each other. Given the correctness assumption from earlier, such nodes will continue communicating with nodes that do not need to be reconfigured and that are connected to them. Further, if any one of these nodes is reachable from the management station, all of them will be reachable.

While it is not essential that the communication property hold true, it often is, especially for routing protocol parameter updates. In such cases, the following proposition holds.

Proposition 1: Let the communication property be true. If there remain a set of routers to be reconfigured, there always exists a router in this set such that either that router or one of its neighbors is reachable.

Proposition 1 implies that when the communication property is true, algorithm reachable will always be successful in its reconfiguration operation. The validity of Proposition 1 is not hard to deduce; we present an informal proof below using Figure 7.

Let a be a type of node that does not need to be reconfigured (i.e., a node of type $a \not\varepsilon S$), b_u be a type of node that has already been reconfigured (i.e., a node of type $b_u \varepsilon U$), and b_{tbu} be a type of node that has not yet been reconfigured (i.e., a node of type $b_{tbu} \varepsilon TBU$). (We use the set definitions from the earlier description of algorithm reachable for simplicity in exposition.) Take any node x from set TBU (i.e., of type b_{tbu}) and conceptually trace the routing path[5] from the NOC to x. (We do not need to know this path; it is only used for the proof.) Map the nodes on the path to their types; for the example in Figure 7, we get a string of the form "$ab_ub_uab_{tbua}b_ub_{tbu}$". By the communication assumption, the nodes corresponding to the initial string matching the regular expression "$(a{*}b_u{*}){*}$" (in this case ab_ub_ua) are reachable from the NOC, and hence a neighbor of the first b_{tbu} (i.e., the neighbor z of node y) is reachable. If the initial string "ab_ub_ua" were empty, then the first node of type b_{tbu} is directly reachable. Note that it is not the node x we selected initially in our proof that is

[5] The term routing path is used loosely here. A formal proof would consider the path packets would take in trying to reach b_{tbu}, even if packets do not complete the journey.

Figure 7: Reachability snapshot for Proposition 1.

reachable; all the proposition says is that there exists *some* node that is reachable or whose neighbor is.

The communication property does hold for the update of most routing protocol parameters. Even in such cases, the routing takes some time to settle down, and the communication property holds after the routing has settled down. The algorithm has to incorporate timeouts and retries to take care of this phenomenon. Typically, the time taken for the routing to settle down for a given network after each reconfiguration can be estimated to be some time t. If there are routers remaining to be updated, and we are unable to find any, retrying once after waiting for time t should provide a router as per Proposition 1. Although t can be pretty large in theory, in practice it is reasonably small, and the number of times you need to wait is expected to be infrequent. Another possible problem that could arise in practice is that a router is reachable but becomes unreachable before the configuration action is completed. We describe our experiences with these issues in time measurement section.

Appendix 2: Routing and OSPF – A Brief Overview

Routers are critical network elements of IP networks. The most important function of an IP router is to route data packets. A forwarding table inside a router is used to decide which route a packet should take. There are several ways to add entries to a forwarding table. Typically, a protocol builds this table dynamically. These protocols are called routing protocols. A routing domain is a collection of routers and all routers in a routing domain run the same routing protocol. One or more such routing domains constitute an Autonomous System (AS). A unique number identifies this AS to the rest of the autonomous systems. This number is called AS number. Routing protocols designed to run among AS are called Exterior Gateway Protocols (EGP). Routing protocols designed to run inside an AS are called Interior Gateway Protocols (IGP). One of the most popular IGPs is the Open Shortest Path First (OSPF) routing protocol [10, 11].

OSPF is a hierarchical routing protocol. It can run in the entire AS. The AS can be divided into OSPF areas. A unique number identifies each area. An area identified by number zero is also called the OSPF backbone. Grouping of routers in an area is a network planning and design issue, which is beyond the scope of this paper. A router is an Area Border Router (ABR) if it is part of more than one area. Entries in a forwarding table, i.e. routes, are generally aggregated at the area level by the ABR so that the processing burden and storage requirements of the routers in the other areas are reduced. For OSPF to work correctly, at least one ABR in an area must be connected to the OSPF backbone.

Some OSPF parameters used in this paper are described below.

- **Hello Interval**. A router running OSPF sends a ''hello'' packet periodically to inform its neighbors that it is alive. The fixed interval between consecutive hello packets is called the *hello interval*.
- **Router Dead Interval**. If a router does not receive a hello packet during a fixed time interval from a neighbor, it declares that neighbor dead. This interval is referred as the *router dead interval*.
- **Area Authentication**. A technique used to ensure that the router sending OSPF information is trusted and the OSPF packet has not been altered.