Shadow Configuration as a Network Management Primitive

Richard Alimi, Ye Wang, Y. Richard Yang
Laboratory of Networked Systems, Yale University
New Haven, CT, USA

ABSTRACT

Configurations for today's IP networks are becoming increasingly complex. As a result, configuration management is becoming a major cost factor for network providers and configuration errors are becoming a major cause of network disruptions. In this paper, we present and evaluate the novel idea of *shadow configurations*. Shadow configurations allow configuration evaluation before deployment and thus can reduce potential network disruptions. We demonstrate using real implementation that shadow configurations can be implemented with low overhead.

Categories and Subject Descriptors: C.2.1 [Computer Communication Networks]: Network Architecture and Design – *Network communications*; C.2.3 [Computer Communication Networks]: Network Architecture and Design – *Network Operations*

General Terms: Algorithms, Design, Management.

Keywords: Network Management, Network Diagnostics

1. INTRODUCTION

Modern IP networks are becoming increasingly complex to configure, as these networks continue to evolve to offer multiple services (*e.g.*, both routing and access control), integrate equipment from multiple vendors, and conduct continuous performance and feature tuning. As a result, it is difficult to generate and maintain the configuration even for a moderately-sized network. A recent survey [40] found that configuration errors are a large portion of operator errors, which are in turn the largest contributor to failures and repair time. Another survey [29] found that more than 60% of network downtime is due to human configuration errors. It further showed that more than 80% of IT budgets are allocated towards maintaining the status quo, a percentage that will only increase due to "increased complexity, lower budgets, and continued business demand."

One way to reduce configuration errors is to use configuration generation tools (*e.g.*, [2]) and/or validate the configuration files using static analysis or simulation (*e.g.*, [15,17,22,37,53]). Although these tools can be quite useful, for example, it has been noted that the configuration analysis tool NetDB provides AT&T significant cost savings [47], these tools are inherently limited in the problems that they can detect. In particular, since configuration files alone do not determine the behaviors of a network, analyzing only the configuration files based on an abstract model of the network and equipment behaviors may leave many problems undetected.

Recognizing the limitations of static analysis and simulation tools, some network operators and equipment vendors build test networks. For example, Cisco has built the NSite [11] facility to test network configurations before actual deployment. However, for most companies, the cost of maintaining a testbed sufficiently similar to the operating network is prohibitive.

Given the limitations of these existing approaches, configuration modifications are frequently deployed into the operating networks without realistic testing. As a contrast, software developers depend mostly on debuggers and actually running their programs before deployment. They run unit and regression tests for correctness and conduct stress tests to validate the programs under load. It would be difficult to imagine the extent of software errors if programs were deployed after only passing through analysis or simulation tools without actually running on the target platform. However, there is no such capability for IP network configuration [38].

In this paper, we propose such a novel capability called *shadow configurations*. With shadow configurations, a network operator may specify two configuration files for a router: one real (current) and one shadow (alternate). The shadow configuration files on a set of routers form a shadow configuration that the network operator intends to replace the current configuration files. The operator can test the shadow configuration files on the actual network without enabling them as the network's real configuration. Running on the existing network infrastructure, this capability is low cost, and thus may be utilized in daily operations. During the testing process, the current network configuration is still running and forwards real traffic; the shadow configuration carries only testing traffic and will not cause disruptions to the operation of the current configuration, even if there are errors in the shadow configuration. The operator conducts correctness and performance tests on the shadow configuration. Our usage of the term "shadow" is motivated by computer graphics, where instead of directly modifying the current display buffer, the display system often uses a shadow buffer to compute the next frame to be displayed.

In particular, by running a set of configuration files directly on the actual network to which they will be applied, a shadow configuration allows a network operator to evaluate the integrated effects of alternate configuration files, router software implementation (including protocol mis-implementations!), the physical network status, and dynamic information such as imported external route advertisements. Many integrated effects on routing are naturally summarized by the forwarding information base (FIB) at each router. We take advantage of the compact FIB representation and develop techniques to analyze the FIBs for configuration validation and adjustment.

Further exploring its benefits, we show how shadow configuration allows a network operator to evaluate, before actual deployment in the real network, whether a set of configuration changes will have the desired effect on network performance. Such realistic performance evaluation reduces the dependency on unrealistic models or assumptions of router processing or the network. Also, the availability of the on-going real traffic in the actual network allows the operator to duplicate a controlled portion of it as testing traffic in addition to generated testing traffic. This reduces the need to generate realistic testing traffic patterns. One potential issue of conducting testing on the shadow configuration is that if we naively send both shadow and real traffic, the combined traffic may overload some network links. Thus, we develop a novel technique,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCOMM'08, August 17–22, 2008, Seattle, Washington, USA.
Copyright 2008 ACM 978-1-60558-175-0/08/08 ...$5.00.

referred to as packet cancellation, to allow both real and shadow traffic to be forwarded in parallel without overloading the network.

After the operator is satisfied with the new configuration, she can simply quickly and smoothly swap the real and shadow configurations with minimal network disruptions. We develop a commitment capability for shadow configurations to reduce the effects of churn and convergence. This usage pattern can be viewed as "two-phase commitment" for network configurations.

To demonstrate feasibility, we extend the Linux kernel and implement necessary components to support shadow configurations in both Quagga [41] and XORP [23] software routers. We show that shadow configurations can be implemented efficiently, with only 12 additional lines of code on the kernel's forwarding fast path for packets not using packet cancellation, and *no* code changes to routing processes. The FIB memory increase to support both real and shadow configurations is less than 35% for the worst-case router under a variety of shadow configurations for a large US tier-1 ISP; the average is much smaller, less than 7%. At run time, our shadow-enabled forwarding engine under heavy traffic has no more than 1.2% CPU usage overhead with a shadow configuration installed.

We also demonstrate the usage of shadow configurations. We show in real implementation that the commitment ability avoids the transient routing convergence period under router maintenance, shutdown and OSPF weight changes. We demonstrate our packet cancellation technique in a usage scenario where the operator tests the impact of a new configuration on a streaming video application. In this case, the combined (raw) shadow and real traffic intensity can be as high as 1.05 times the capacity of some links. However, packet cancellation shields real traffic from shadow traffic while at the same time, the measured performance of the shadow video streams is close to the case when it is using the network alone (difference is less than 1%).

In summary, we have made the following contributions:

- We propose the novel capability of shadow configurations.
- We develop novel techniques for configuration analysis, evaluation and management.
- We provide an implementation and demonstrate that the shadow configuration capability can be implemented with low overhead.

2. MOTIVATING USAGE SCENARIOS

To drive our system design, we conducted a survey of operator configuration usage scenarios. Below, we list several key usage scenarios that we would like to support using shadow configurations. The objective of the list is not to be complete, but to motivate our design.

Equipment Maintenance, Testing: A network operator may need to shutdown a running router or link for maintenance. For example, many hardware and software updates suggest that a router or network interface card be taken offline during the process. To prepare for a shutdown, the operator makes the shadow configuration the same as the real configuration except that the link or router to be shutdown does not appear in the shadow configuration. The operator evaluates the shadow configuration, makes potentially necessary adjustments, and then commits it as the real configuration.

As another example of an equipment shutdown usage scenario, a surveyed operator commented that he needs to periodically shutdown a primary link to test if its backup link is operational and will be used after network reconvergence. Since the capacity of the backup link may be lower than the primary link, such tests may cause network disruptions. With shadow configurations, he can just shutdown the primary link in the shadow configuration and test if the backup link will be used in the shadow configuration after reconvergence.

After the maintenance or addition of a new network device, the operator includes the device in the shadow configuration, evaluates the effects, and makes adjustments before switching to the new configuration. This step can be particularly useful as multiple surveyed network operators commented that it is common for issues to arise after a maintenance upgrade.

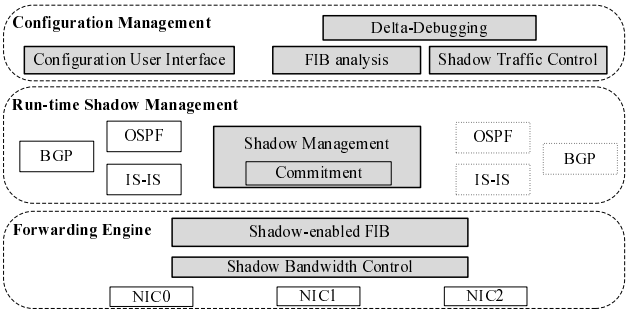

Figure 1: System architecture for network management with shadow configurations.

Configuration Parameter Tuning: Many network operators need to tune configuration parameters to address performance or security issues. For example, a network operator may conduct traffic engineering to improve network performance, and many traffic engineering techniques (*e.g.*, [18, 19, 39, 42]) require the modification of configurable parameters (*e.g.*, OSPF weights or egress point selections). However, such parameter adjustments may cause disruptions due to human error or routing reconvergence. As another example, a network operator may change its network access control list. However, such changes may lead to network disruption due to mis-configurations or unexpected interaction with routing. Shadow configurations support such tuning of parameters and testing correctness and performance.

Network Diagnosis: One problem with network diagnosis is that it is difficult to conduct root-cause analysis (*e.g.*, end-to-end performance violations). Although many network diagnosis techniques have been proposed lately (*e.g.*, [5, 13, 25, 28, 31, 32, 43, 45, 46]), a major limitation of network support is that they cannot easily conduct unit or "destructive" testing [47] as is done in software debugging. Shadow configurations allow a network operator to construct a shadow network on a subset of the network, and compare the differences in the real and shadow configurations to help with root-cause analysis. In particular, the delta [54] testing technique for software debugging can be particularly helpful to the automation of configuration debugging.

Feature/Service Testing: A network operator may be reluctant to enable new features (*e.g.*, queue management or scheduling algorithms) or services (*e.g.*, VoIP) on her operational routers due to concerns of unknown performance impacts, as many factors affect network performance [8, 35, 49, 55]. Shadow configurations allow the operator to conduct an evaluation in the shadow configuration. She can finally commit the shadow configuration as the real one once the integration is verified to work correctly.

3. SYSTEM OVERVIEW

We now present an overview of our system. The key components in our system are shown in Figure 1. We focus on a high-level overview in this section. Details and implementation of several components will be discussed in the following sections.

We divide our system into three layers: (1) forwarding engine; (2) run-time shadow management layer; and (3) configuration management.

3.1 Forwarding Engine

Foundation

The key component is a forwarding engine supporting both real and shadow configurations. In this discussion, we focus on the

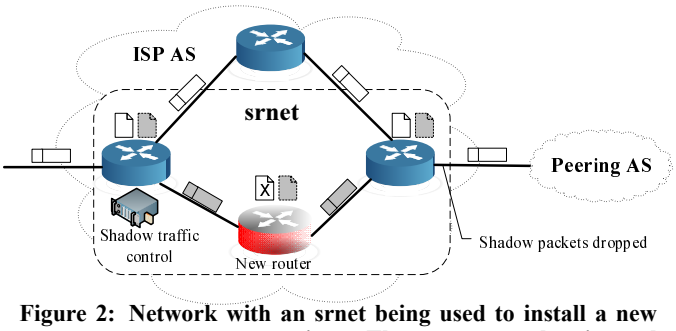

Figure 2: Network with an srnet being used to install a new router to support new services. The new router has its real configuration disabled during installation.

forwarding information base (FIB) for presentation, but note that the forwarding engine handles other items such as access control lists (ACLs) as well.

Let $\{1,\cdots,N\}$ be the set of routers in a configuration. Let $C = \{C_1,\cdots,C_N\}$ be their configuration files. In abstraction, the control plane converts the configuration files into a configuration for the forwarding plane: $C \Rightarrow \{fib_i\}_i$, where fib_i is the FIB at router i. The FIB entries at an interface maps a destination IP address to an output interface.

We refer to a set of connected routers running a shadow configuration as a *shadow-running network* or *srnet* for short. In this paper, we consider only the case where a srnet belongs to a single autonomous system (AS). A srnet is likely to be the whole IP network of the AS, but can be only a subnet. The latter possibility gives flexibility such as incremental deployment. A router i inside a srnet has two configuration files: C_i^r for the real configuration and C_i^s for the shadow. In the forwarding engine, it will then have two FIBs, (fib_i^r, fib_i^s), for the real and shadow respectively.

A link (interface) may leave a srnet, and we refer to such a link as a border link. The FIB at such a border link will need to contain ingress and egress policies for how to handle incoming and exiting shadow packets. Figure 2 shows a network containing a srnet.

When a packet arrives at a border link of a router i, the router uses the ingress policy to determine whether it should apply fib_i^r or fib_i^s. We refer to a packet forwarded using the shadow configuration as a *shadow packet*, and a packet forwarded according to the real configuration as a *real packet*. Router i uses a *shadow bit* in the IP header to indicate whether it is a shadow packet or a real packet.

When another router j receives a packet, it checks whether the packet is a real packet or a shadow packet, and uses the corresponding forwarding table. If it is a shadow packet and is leaving the srnet, the egress policy is applied (*e.g.*, dropped).

Shadow Bandwidth Control

With both shadow and real traffic using the same network, we need a shadow bandwidth control component to regulate the bandwidth sharing. In particular, testing the performance of a shadow configuration should not cause disruption to the real traffic. We focus on network bandwidth, but one could also consider processing bandwidth. For example, per-packet processing such as IP lookup may be the bottleneck.

We support three modes of shadow bandwidth control:

- Priority: real traffic has higher priority than shadow;
- Partition: each configuration is allocated a portion of bandwidth;
- Packet cancellation.

Priority and partition modes can be useful, for example, when the payload must be carried end-to-end to include the responses of end hosts and servers or when evaluating deep-packet inspection. In the partition mode, the network operator can specify non-work conserving scheduling for shadow packets to provide "scaled-down" testing bandwidth and arrival processes.

Packet cancellation is designed to allow an operator to conduct stress tests on the shadow configuration to reveal issues under higher load. The operator can certainly try to wait for a time period when the real traffic is low. However, there may not exist such a time period, or the real traffic load that the operator would like to duplicate for testing happens only when the real traffic is relatively high. Packet cancellation has the following two objectives:

- Performance of the real traffic is not severely degraded;
- Performance measurements taken from shadow packets should be close to the measurements that would be observed if the shadow configuration were the only active configuration.

Packet cancellation is presented in Section 5.

3.2 Run-time Shadow Management

Our next layer provides two main functions:

- It provides a run-time and management environment for real and shadow configurations and routing processes (*e.g.*, multiplexing of control packets, CPU and bandwidth management). We discuss one implementation in Section 7, including a technique for exchanging information with routers outside a srnet (*e.g.*, with BGP) that presents a single consistent view to the outside world.
- It provides a commitment capability to smoothly swap the configurations, which is important for many usage scenarios. This is especially desirable because the convergence process is a major source of disruption: reconvergence after a configuration change can cause network outages [1] or additional configuration errors [30]. We present our commitment protocol in Section 6.

3.3 Configuration Management Layer

This layer provides multiple utilities to take advantage of and control the capability of shadow configurations. We have implemented the following tools:

- Configuration user interface (cui): the operator is presented with two command-line terminals, one real and the other shadow. Using this interface, a network operator issues traditional router commands such as `traceroute` and `ping`. Additional commands are provided to control our commitment protocol.
- Shadow traffic control (stc): the operator is allowed to specify shadow traffic (*e.g.*, real traffic to be duplicated to the shadow configuration and intensity of generated shadow traffic) and collect traces.

We have also investigated other useful tools:

- Shadow configuration analysis using FIB (scaf): a tool to detect routing loops and reachability issues. We give more detail on this tool in Section 4.
- Shadow regression tester (srt): a tool to play test cases (*e.g.*, reachability of important applications at important locations).
- Configuration delta debugging (cdb): a tool based on the observation that by comparing the FIB and performance of the real configuration with the shadow configuration, we can automate a large fraction of configuration diagnosis.

4. SHADOW ANALYSIS USING FIB

In this and the next two sections, we present the details of shadow configuration analysis using FIB, packet cancellation, and shadow commitment. They are presented in this order as this is a common order in many usage scenarios.

4.1 Objectives and Overview

With the availability of a shadow configuration, the network operator can analyze $\{fib_i^s\}_i$ *before* they become installed for real packet forwarding.

In particular, we investigate how to detect forwarding loops using the collection of FIB states. As made evident by measurement results [24] and online detection algorithms [51], forwarding loops happen frequently in real networks. Since routing loops can cause unnecessary load and dropped packets, detecting loops caused by a new configuration before its actual deployment can have great value. Our system computes the set of destination addresses as well as routers present in the loop, providing the network operator with detailed information from which she can debug the problem.

We also detect reachability issues, another common type of configuration errors [33,53]. Some reachability issues can be extremely challenging to detect using any static analysis or simulation tools because they depend on software implementation. For example, the Cisco document [10] reports a common OSPF routing problem before Cisco IOS Release 12.1(3) related with forwarding addresses. The reachability issue noted was caused by the software implementation of a Cisco-specific optimization, and thus can be difficult to isolate using only configuration files. By looking directly at the FIB states, our system can bypass detailed modeling and abstractions, and provide the network operator useful reachability information to help debug the problem.

Note that for presentation clarity, we consider only unicast addresses; we assume that there exists a unique nexthop in each FIB for a single destination address. Also note that it is straightforward to add other forwarding mechanisms (*e.g.*, label switched paths) to our analysis.

4.2 Representative IP Addresses

A major complexity in reachability and forwarding loop analysis is that FIB lookup in modern routers is implemented using longest prefix matching, and different routers in the same network may have different sets of destination IP prefixes.

To use existing efficient graph algorithms to check reachability and forwarding loops, we first pre-process FIBs to compute *representative* IP addresses. With representative IP addresses, FIB analysis is done on individual IP addresses, without the need to handle longest prefix matching.

Consider a simple example where each FIB table in the network consists of the following destination IP prefixes: a default route (*i.e.*, 0.0.0.0/0), 10.1.0.0/16, and 10.1.0.0/24. Then if we verify that there is no reachability or routing loop problem for each IP address in the set {0.0.0.0, 10.1.0.0, 10.1.1.0, 10.2.0.0}, then the network has no reachability or routing loop problem.

The algorithm *findrepip* (Figure 3) computes the set A of representative IP addresses for a network. The algorithm computes the set A_i of representative addresses for each FIB fib_i^s. The set A for the whole network is obtained by merging the A_i's. To make the merging efficient using a priority queue, the algorithm maintains each A_i to be sorted.

When processing each entry in fib_i^s, the algorithm adds to A_i up to two addresses. The first is the beginning address of the destination prefix associated with the entry, and the second is the beginning address of the *next* range that could come after the entry's destination prefix.

4.3 Computing Reachability and Loops

Once the set of representative addresses is found, each address can be analyzed using standard graph algorithms to detect reachability issues and forwarding loops:

1. Reachability: (1) set of routers R_a that can reach address a; and (2) set of routers W_a with FIB entries for address a but cannot reach address a;
2. Forwarding loops: sets of routers L_a participating in forwarding loops for address a.

```
findrepip({fib_i^s}_i) – Compute representative address set A
01. foreach fib_i^s do
02.   A_i ← ∅ //sorted rep addr for fib_i^s
03.   foreach entry e in fib_i^s do
04.     A_i ← A_i ∪ {min{e.addr_range}}
05.     if max{e.addr_range} ≠ 222.255.255.255 then
06.       A_i ← A_i ∪ {1 + max{e.addr_range}}
07.     endif
08.   endfor
09. endfor
10. // Merge rep addr into single sorted list
11. A ← priority_queue_merge({A_i}_i)
12. return A
```

Figure 3: Algorithm for computing representative addresses given $\{fib_i^s\}$.

5. SHADOW PACKET CANCELLATION

With a consistent and reachable forwarding state, the network operator next might ask, "If I adopt this alternate configuration on my network, how will it perform?" Such a question is important when deploying new services such as voice or streaming media, or when the operator may want to evaluate the likely impacts of the new configuration on service level agreements.

At this point, the reader might suggest that since the operator already has the FIBs of the shadow configuration, she may compute or simulate the performance characteristics using a traffic demand matrix. This is certainly a feasible approach and our system can support it. Such computation- or simulation-based approaches, however, implicitly rely on a model for packet processing inside each router for features such as QoS or any particular queue management techniques. New techniques such as traffic shaping or differentiated services would require modifications to the model [12]. On the other hand, enabling direct measurements allows processing within the routers to be treated as a black box.

5.1 Overview

Recall that the objectives of packet cancellation are that (a) both real and shadow traffic are forwarded according to their original queue management schemes, and (b) shadow packets are (typically) only delayed by other shadow packets while real packets are (typically) only delayed by other real packets.

This mode uses two techniques: packet (payload) cancellation and a virtual clock. The key insight is that the payload of shadow data packets may not always need to be transmitted; that is, when the focus of an evaluation is on network performance metrics such as delay, the shadow data packets then are not intended to be received by end hosts. Thus, we need only to (1) retain the information relevant to forwarding the traffic within the network, and (2) know the correct payload size so that gathered performance measurements remain meaningful.

Given the preceding insight, we allow a router to append the header of a shadow packet to a real packet before it is transmitted over the link. The input interface at the receiving router removes the appended shadow header, and processes it accordingly. If the shadow traffic is delayed too much by the real traffic, we can append multiple shadow headers to catch up with the delay.

5.2 Shadow Data Packet Cancellation

We now describe how our scheme processes shadow data packets. At the output interface, shadow packets and real packets are separated into two queues, Q_s and Q_r. This also allows the shadow configuration and real configuration to define different queue sizes and queuing disciplines. When it is time to transmit the next packet, the line card applies the algorithm shown in Figure 4.

Specifically, if Q_s is empty, send $head(Q_r)$, the head of the real packet queue; otherwise, extract the headers of the shadow packets that should be transmitted and combine them with $head(Q_r)$.

```
pktsched() – packet cancellation and scheduling.
01. if not empty(Q_r) then
02.     p ← dequeue(Q_r) // Select real packet
03.     // Append shadow packet headers
04.     for 1...MAX_CANCELLABLE do
05.         if not virtual_clock_expired(peek(Q_s))
06.             break
07.         p ← append(p, ip_hdr(dequeue(Q_s)))
08.     endfor
09.     transmit(p)
10. elseif not empty(Q_s) then
11.     // Send shadow packet if available
12.     if virtual_clock_expired(peek(Q_s))
13.         transmit(dequeue(Q_s))
14. endif
```

Figure 4: Packet cancellation and scheduling.

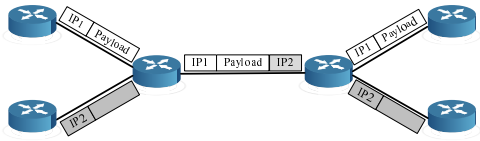

Figure 5: Shadow packet header combined with a real packet for transmission on a single link.

We may extract multiple (up to *MAX_CANCELLABLE*, set to 3 in our implementation) shadow packets to "piggyback" on a real packet due to packet payload sizes and previous delay of shadow packets. To determine whether a shadow packet should be transmitted or piggybacked, the shadow queue maintains a virtual clock. The virtual clock estimates whether the transmission of a shadow packet should be started (*virtual_clock_expired*) if there were only shadow traffic.

Note that it is important that when extracting headers from a shadow packet, we extract all IP headers to allow the scheme to work properly when tunnels or VPNs are configured. If any IP header that must be interpreted is encrypted, the scheme may not work. The TCP/UDP header, if it exists, should also be extracted since it may be required for packet filtering (*e.g.*, in Cisco's policy routing, NetFlow sampling, and firewalls). In a simple IP network without tunnels or VPNs, the extracted headers will consist of a single IP header and a TCP/UDP header, and will typically be 40 or 28 bytes in size.

There is one additional piece in the scheme. It must be possible for the incoming interface at the receiving router to determine whether it is receiving a single packet or combined packet. If the link-layer payload is larger than length indicated by the IP header, the router strips off the appended headers, verifying their IP version, header length, and optionally the checksum.

Figure 5 shows how a shadow payload can be canceled with a real packet for transmission over a link. The shadow header is extracted at the receiving interface and forwarded independently.

With packet cancellation, it is possible that the full size of the transmitted frame becomes larger than the next interface's MTU, causing the packets to be silently dropped at the next hop. To handle this, one could simply decrease the MTU to accommodate the additional canceled packets. To avoid additional fragmentation, one could instead increase the MTU, but internally process packets (*i.e.*, handle fragmentation) at the routers according to the original MTU.

Further consideration is required when operating on Ethernet. To provide intuition, the algorithm in Figure 4 might fill in all "whitespace" left by real traffic with full shadow packets, causing the link utilization to approach 100% and causing large delay variations. One simple way to solve the problem is to always transmit only the shadow header and set a timer to throttle shadow queue transmission rate when the real queue is empty. In our implementation, we found that the available timers are too inaccurate to retain the appropriate packet delay variations. Thus, we adopt the heuristic that even when the real queue is empty, only the shadow packet header is transmitted if link utilization is above a certain threshold (we use 85%). Since a previous hop may have trimmed a shadow packet, it may be necessary to expand the packet and zero-fill the payload when below the threshold.

5.3 Shadow Control Packet Forwarding

We previously considered only shadow data packets. Packet cancellation cannot be applied to shadow control packets, such as route advertisements, SNMP messages, or ICMP packets. For safety and because control packets can originate from many places (routing processes, ARP, ICMP, etc.), we opt to explicitly mark a shadow packet that *can* be canceled (*e.g.*, in generated testing traffic) with a *PD* bit, indicating that its payload is dropable. We process shadow control packets using a separate queue.

5.4 Overhead and Perturbation Analysis

FIB Lookup

One potential bottlebeck is FIB lookup instead of bandwidth. Since a combined packet received in packet cancellation mode contains multiple headers that might require separate lookups, it is crucial that the router be able to support this additional processing burden.

Forwarding engines in many routers are designed to handle worst-case scenarios where all incoming packets have the minimum size. In particular, assume that a router can support $\alpha\frac{L}{K_{min}}$ packets per second where L is the link speed in bytes per second; K_{min} (typically, $K_{min} = 40$ bytes) is the minimum packet size; and $\alpha \leq 1$ is the efficiency factor.

Let k_r denote the packet sizes of real traffic and k_s the packet sizes of shadow traffic. Let α_r be the link utilization caused by real traffic and α_s that of shadow traffic. To sustain lookup, we need:

$$\mathbb{E}\left[\frac{\alpha_r L}{k_r}\right] + \mathbb{E}\left[\frac{\alpha_s L}{k_s}\right] < \alpha\frac{L}{K_{min}}.$$

Using the packet size distribution in [26], we can compute α_s given α_r and α. For $\alpha \geq 0.7$ and $\alpha_r \leq 0.8$, we have $\alpha_s \geq 0.75$, meaning the link utilization for shadow traffic can reach up to 75% while still being supported by the forwarding engine.

Performance Measurement Accuracy

Our packet cancellation scheme tries to remain as consistent as possible with the original forwarding behaviors for both shadow and real packets. This is important since the operator must have confidence that the measurements obtained on real and shadow traffic are indicative of the measurements that would be observed if only real or only shadow traffic were present in the system.

To better understand our scheme, consider a basic model: packets have uniform sizes, all packets have space reserved for an additional shadow header, and packets do not arrive in the output queue when a transmission is in progress. Then, we can show that there will be no delay or loss perturbations for either real or shadow packets.

CLAIM 1. *For any packet p, $d^r(p) = d^s(p) = 0$ where $d^r(p)$ (*resp.*, $d^s(p)$) is the end-to-end packet delay perturbation for a real (*resp.*, shadow) packet.*

CLAIM 2. *For any packet p, $l^r(p) = l^s(p) = 0$ where $l^r(p)$ (*resp.*, $l^s(p)$) is the packet loss perturbation for a real (*resp.*, shadow) packet.*

6. SHADOW COMMITMENT

As we discussed in Section 3, with a consistent and reachable forwarding state and satisfactory performance, the network operator may then decide to apply the shadow configuration as the network's actual configuration. We define the objective of the commitment process to be swapping the shadow and real configurations at

all routers within the srnet. Swapping allows the network to rollback if an error occurs or the operator finds the new configuration unacceptable.

6.1 Overview

Although there are several previous studies on updating FIBs across routers (*e.g.*, [20,21,56]), our shadow configuration commitment problem is distinct from these previous problems. For example, many types of changes and routing processes may be involved in a configuration change, so routing-protocol specific techniques (*e.g.*, [21]) may not apply.

Our protocol is inspired by the simple and clean map dissemination protocol proposed by Lakshminarayanan *et al.* in [34]. We address additional issues in our specific context including integration with version control of distributed configuration files, rollback of configurations, and simplicity of router maintenance.

To integrate with configuration version control (*e.g.*, CVS), before each commitment, the operator broadcasts two tags to each router: C_{old} identifies the real configuration before swap, and C_{new} the shadow configuration before swap. An additional functionality of the tags is to mark packets to avoid forwarding loops during the swapping period; this is inspired by the map dissemination in [34]. After commitment, the tags should be removed for simplicity of router maintenance.

Consider the scenario when routers always tag packets (*e.g.*, with global map sequence numbers [34]), and the network operator powers on a new router. After reading its local configuration file, a routing process (either shadow or real) must communicate with the corresponding routing processes of its neighboring routers. However, since the router does not know which tag denotes the real configuration and which denotes the shadow, it may not be able to tag routing messages correctly such that they are demultiplexed to the correct routing processes at its neighbors. One could design various ways to work around this problem (*e.g.*, designating globally constant tags or a protocol to allow a router to query tags), but they introduce extra complexity. Our commitment protocol chooses to remove the tags after commitment so that the shadow bit has well-defined semantics (0 indicates current and 1 indicates shadow) during normal operation.

6.2 Protocol Operation

The protocol proceeds in four phases. Messages to the routers are sent first using the real configuration, then the shadow configuration in the case where the real configuration is non-operational.

Phase 1: During the first phase, the operator sends a TAG DISTRIBUTION message containing two tags to each router. The two tags are temporary network-wide identifiers for the configurations: C_{old} identifies the real configuration before swap, and C_{new} the shadow configuration before swap. Upon receiving these tags, each router creates a lookup table to remember the mapping. To report its configuration file to version control (`diff` is enough) and to make sure that all routers have received the tags, each router responds to the TAG DISTRIBUTION message with an acknowledgment. The operator waits to receive an acknowledgment from each router.

To prevent links from being oversubscribed while commitment is in process, testing traffic marked with the *PD* bit (discussed in Section 5) is immediately dropped by routers as of this phase. This is done by adding an output filter rule.

Phase 2: During the second phase, every router knows the tags, so the operator sends a TAG PACKET message to all routers causing them to start marking packets with tags. Since routers do not receive the TAG PACKET message simultaneously, some packets are marked with tags and some use the shadow bit during this phase. Packets generated at the router by a configuration are marked with that configuration's tag, and received packets already marked with tags are forwarded according to the appropriate configuration. Tags are added to packets received without tags: if the shadow bit is unset, it uses the tag of current real configuration (currently C_{old}); otherwise, it uses the tag of the current shadow configuration (currently C_{new}). If a router has not received the TAG PACKET message but receives a packet with a tag, it additionally triggers the router to transit to a state as if it had received the TAG PACKET state. This indirect triggering can speed up this phase.

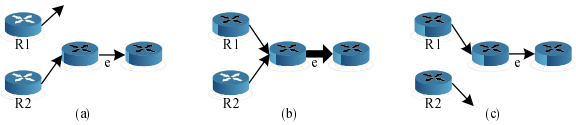

Figure 6: Scenario showing how a transient state can cause temporary congestion. White routers have not yet swapped; black routers have swapped.

Before moving to phase 3, the network must wait for the following two conditions to become true: (1) no routers are still marking packets using the shadow bit; (2) no packets using the shadow bit are in transit.

At the second half of the Phase 2, the two conditions are satisfied. For the first condition, the operator needs to receive an acknowledgment from each router. After the first condition is true, the operator satisfies the second condition by waiting for a short time (*e.g.*, the estimated upper bound of link latency) until all packets have been processed by the next router in their path.

Phase 3: During the third phase, since no packets will be using the shadow bit, the routers can safely swap the configurations. The operator transmits a SWAP message to the routers. Each router swaps the real and shadow configuration after receiving the message, and sends an acknowledgment back to the operator. Note that the tags associated with each configuration are *not* swapped. Also note that ingress routers that have received the SWAP message now tag unmarked packets with C_{new} instead of C_{old}.

Phase 4: In the last phase, the operator sends a MARK SHADOW BIT message to each router, allowing them to again mark packets using the shadow bit. To report success, each router sends an acknowledgment back to the operator.

6.3 Error Handling and Rollback

There are potential error conditions during commitment. Link or router failures cause the routing and forwarding processes (*e.g.*, fast rerouting) to automatically start to react and bypass the failed equipment. The presentation below is focused on error conditions leading to the disruption to our commitment protocol.

Transient States: We define a transient state as a state where some data packets use the old configuration and others use the new configuration. A potential pitfall of a transient state is that the utilization of some links may be higher than it would be under either of the two configurations. Consider an example shown in Figure 6. Routers R_1 and R_2 will both change forwarding paths in the new configuration. In Figure 6(a), neither has swapped and only R_2 forwards through link e. After R_1 has swapped in Figure 6(b), the link is used by both routers, possibly causing temporary congestion. Once R_2 swaps in Figure 6(c), the transient state ends and the final router is no longer using link e. Note that such transient states also can happen under some circumstances with other approaches such as the map dissemination approach [34].

Recovery and Rollback: During phase 1, if any one router reports an error or the controller does not receive acknowledgments from all routers, the commitment should abort. As a soft state design, if a router does not receive TAG PACKET before its local timeout, it should change back to the normal state. During phases 2 and 4, if the operation of any router is unsuccessful or times out, the operator will retry the phase. Routers can remain in their current states, as this is not a transient state. It is straightforward if the operator chooses to rollback to the original configuration since the tags are already distributed and only phases 2, 3, and 4 of the protocol need be executed.

The only phase in which a transient state can happen is phase 3. Here, it is important for the state not to be permanent. Consider what can happen during phase 3. If acknowledgments are received from all routers, the transient state has already ended and no rollback is necessary. If at least one acknowledgment is missing, there are two possible reasons: a router did not receive or process the SWAP command, or the SWAP is processed but the acknowledgment was not delivered. We would like to detect the first case. Since an error may have occurred at such routers (*e.g.*, a routing process crashed), it may not be possible to query them directly. Thus, the operator queries the router's neighbors. If the router in question is tagging its forwarded traffic (recall that only real packets are present) with C_{old}, then there exists a router that has not processed the SWAP message, and the srnet should rollback. Note that even if a router crashes during commitment, both the real and shadow configurations of other routers within the srnet reconverge appropriately.

PROPOSITION 1 (SAFETY). *Packets never alternate back and forth between configurations. Thus, the commitment protocol does not create any additional forwarding loops. Also, control packets such as route advertisements are delivered correctly even while the commitment protocol is executing.*

PROPOSITION 2 (LIVENESS). *If for every router, commitment control messages are delivered in finite time, and the router either responds to the messages or is recovered offline in finite time, the srnet returns to normal operation, and the transient state is no longer present.*

7. SHADOW IMPLEMENTATION

To demonstrate feasibility, we have implemented a fully operational system supporting shadow configurations. We now discuss in detail Layers 1 and 2 of our system architecture. The related components are shown in Figure 1.

7.1 Objectives

There are three primary objectives fulfilled by our implementation: (1) identify operating system configuration entities with the shadow and/or real configurations; (2) keep CPU and memory overhead low by merging configuration entities where possible; (3) reduce code changes (*e.g.*, to routing processes and network tools) after introducing shadow configurations.

7.2 Supporting Shadow Configurations

A key issue in implementing support for shadow configurations is associating entities maintained within the operating system (*e.g.*, FIB entries, filtering rules, interfaces, neighbor entries, and packets) with the appropriate configuration. To demonstrate that this can be done with minimal effort, we present an implementation consisting of extensions to the Linux Kernel (version 2.6.22.9). Our design is able to support both XORP [23] (version 1.4) and Quagga [41] (versions 0.98.6 and 0.99.9) without any source code changes to either software package. Either can be used interchangably above our shadow-enabled kernel, which illustrates support in heterogenous environments.

Figure 7 shows the major components of the implementation. We emphasize that different routers may choose different implementation as long as the messaging format (*i.e.*, how shadow data packets and shadow control packets are encoded) is standardized.

Separating Configurations: Each entity is associated with a particular configuration. Entities corresponding to the current real configuration are applied to transit traffic and routing processes that communicate with the outside world, while entities corresponding to the current shadow configuration are being evaluated by the operator.

We append data structures for necessary entities with a mask, where each bit position corresponds to a particular configuration.

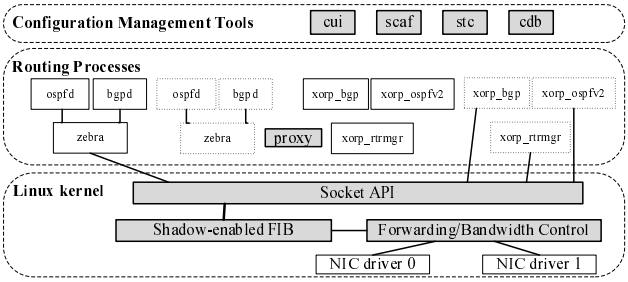

Figure 7: Implementation of router supporting shadow configurations: Shaded parts are new or modified.

If an entity appears in more than one configurations, multiple bits are set in the mask.

One installed configuration is considered as the real while another is considered the shadow. This mapping is maintained in a simple two-entry translation table, allowing the commitment's swap operation to simply swap the entries in the translation table.

Shadow-enabled FIB: We merge entries in FIB table for both configurations to reduce memory overhead. FIB entries use a mask to indicate the configurations to which the destination subnet belongs.

We extend the FIB lookup, insertion, and deletion algorithms to handle the merged FIB table. If the forwarding behaviors (*e.g.*, next hops) in the two configurations are different, we record the difference inside the entry.

Other similar kernel tables, such as neighbor entries, filtering rules, and interface addresses are handled similarly.

Socket API: Extending the kernel tables is not enough. When a userspace program, such as a routing process or a testing tool communicates with the kernel, it uses the socket API. We extend the kernel's socket data structure to reference the configuration to be used when transmitting packets and demultiplexing incoming packets. Routing processes in different configurations can safely bind to the same IP addresses and ports.

Packets: Our current packet format supports IPv4 and ARP, but the same methodology can be applied to IPv6 or other Layer 3 protocols. During normal operations, each packet needs two bits: a shadow bit S, and a PD bit to indicate whether the payload can be dropped. Both S and PD are always 0 for transit traffic. For IPv4 packets, S uses the low bit of the version field, and the PD uses the unused flag bit. Such a mapping causes shadow packets to be automatically dropped by routers that are not shadow-aware. Two additional bits are needed during commitment: TP indicates whether a tag is present, and TG indicates the tag. We store TP in the highest bit of the TOS field and TG in the next highest bit. We use the highest four bits of the ARP header's operation field to mark ARP packets. Note that it is also possible to encode some or all of this information in a shim header.

Packets received by the kernel are demultiplexed according to the translation table (and the tag assignment during commitment). A reference to the appropriate configuration is stored in the packet's data structure for usage in key parts of the TCP/IP stack such as the routing cache and FIB lookups, ICMP errors, and UDP/TCP demultiplexing.

Shadow-aware Programs: Since we also would like provide support for existing programs, we allow a default configuration to be defined for a process, and the attribute is inherited by child processes. Sockets created by a process initially belong to the process's default configuration. We can then launch any program within the desired configuration.

A shell is started for each configuration to enable an operator to apply changes to a particular configuration. The shell indicates whether its configuration is currently defined as the real or shadow.

Routing Processes and Tools: In most implementations, routing processes are normal user processes. Changing networking configurations in the Linux kernel is primarily done using `netlink`

sockets. By starting a routing process in the appropriate shell, its sockets are associated with that configuration and the kernel interprets the changes to entities as applying to that configuration. We configure Quagga and XORP such that two instances can be running concurrently, allowing both a shadow and real configuration to be deployed.

The same technique is applied to common network testing tools such as `ping`, `traceroute`, and homegrown scripts, allowing them to operate without modification. We use this approach with our custom traffic generation program and measurement program used in our evaluations.

It is possible that some vendors add shadow-awareness directly to userspace processes (*e.g.*, to use a shared RIB to further reduce memory overhead or supporting additional features in traceroute), while others may want to reduce code changes.

Connection to Outside: Our implementation uses proxies to handle control plane connectivity to outside of a srnet. Such connectivity is necessary to support incremental deployment and interdomain scenarios. These simple proxies can handle not only normal operations but also shadow commitment.

Consider the example of eBGP. Suppose without shadow configurations a BGP routing process b has a BGP peer e in another domain; that is, b has a TCP connection at port 179. With shadow configurations, corresponding to b, there may be two BGP processes b^r and b^s for the real and shadow configurations. We introduce a proxy b^p for b. Then b^p peers with the external BGP peer e (by listening at the IP address and BGP port 179). The process b^p forwards each incoming BGP message from e to both b^r and b^s, which can then apply its ingress filtering policies. Whenever b^r sends a BGP message to e, it is forwarded to b^p which forwards to e.

We use a novel transaction rollback technique to handle commitment with visible external effects. Specifically, the proxy keeps a log of forwarded messages. Whenever b^s sends a BGP message to e, it is stored locally by b^p. If the network swaps the real and shadow configurations, b^p computes the differences of the messages of b^r and b^s, rolls back the unnecessary impacts of b^r (*i.e.*, withdraw different routes), and then installs the effects of b^s without disconnecting the external BGP connection.

Shadow-aware Interfaces: It is necessary for routers to drop shadow packets and untag transit packets (in the case of commitment) before exiting an srnet. We enable a *shadow-aware* attribute on each interface that participates in the srnet.

Since our evaluation environment utilizes ARP, there is one additional complexity during commitment. Egress traffic should not be delayed or possibly dropped while it waits for the new configuration to query for the MAC address of the peering router outside of the srnet. Thus, we configure the kernel to accept unsolicited ARP replies and duplicate any received ARP reply to the shadow configuration for interfaces with the *shadow-aware* attribute disabled.

8. EVALUATIONS

We first present our methodology, then present our results in two parts. In the first part, we present results that show that the overhead of supporting shadow configuration is very small. In the second part, we demonstrate the effectiveness of shadow configurations in three usage scenarios.

8.1 Methodology

Implementation: We use our implementation as described in Section 7.

Configurations: We use the configuration files of the two operating networks in Table 1. US-ISP is a large US tier-1 ISP.

We use the configurations of US-ISP only for evaluation of FIB size overhead. The rest of our experiments use a small illustrative topology and an emulation of the Abilene backbone. We use Emulab's [52] 3 Ghz PCs with 1 Gbps and 100 Mbps Ethernet links. We take additional steps to load configuration data into our emulation of the Abilene backbone. Configuration commands are translated to both XORP and Quagga syntax. Then BGP routes from Abilene's July 2007 BGP RIB dumps are injected as static routes at virtual egress points, `dummy0` interfaces, at the appropriate routers. Routes for the University of Utah are removed so as not to interfere with the Emulab addresses configured on the routers. Since the versions of XORP and Quagga used did not support IS-IS, we translated Abilene's configurations to use OSPF.

Network	#Nodes	#Directed Links	Syntax
Abilene	9	26	Juniper
US-ISP	-	-	Cisco

Table 1: Network configurations used.

Data Traffic: We use CAIDA [6] packet traces in our evaluations. When using these traces on our emulation of the Abilene backbone, we remove packets for destination addresses not appearing in the BGP routes accepted by Abilene.

Performance Measurements: To obtain performance measurements under packet cancellation, we use a custom utility similar to `iperf` that timestamps generated packets just following the IP header and sends using raw sockets. The timestamp is not lost during packet cancellation. We modify the kernel to deliver canceled packets to raw sockets. The server computes delay between sending and receiving time, and uses linear regression to subtract off mean delay and account for clock drift.

8.2 Overhead

Since we intend that shadow configuration be used in production networks, the overhead of supporting it should be small. One reason we chose Linux is to see the overhead in a general platform. We consider (a) data path forwarding overhead due to additional complexity to support a shadow configuration; (b) FIB storage overhead due to addition of a shadow configuration; and (c) FIB update overhead due to addition of a shadow configuration.

Data Path Forwarding Overhead: Our results show that there is truly a negligible overhead on the data forwarding path due to the additional complexity of supporting a shadow configuration. For this test, we use a particular traffic load both with the standard Linux kernel, and then again with our shadow-enabled kernel. When employing our shadow kernel, we load a shadow configuration but do not generate shadow traffic.

We use a topology with 3 routers with 1 Gbps Ethernet links; there is a sending, intermediate, and receiving router. The sending router uses the Linux kernel's `pktgen` module to generate 300-byte packets so we can stress-test the intermediate router's forwarding path. Our implementation doesn't use any additional memory copies for real packets, so larger packet sizes do not add overhead in our shadow kernel.

The sending router transmits packets for 30 seconds with randomly generated destination IP addresses in the range 10.0.0.4-10.255.255.255 to ensure that FIB lookups (on the intermediate router) are rarely handled by the routing cache. The intermediate router configures one default route for 10.0.0.0/8 to route to the receiving router, and also adds additional 9306 randomly generated entries from 10.0.0.0/8 with a prefix length distribution matching the global BGP tables published by the Route Views Project on January 18, 2008 [44]. Note that there are no prefix lengths shorter than 8. Also, 9293 routes are added in shadow configuration, with 60% of the prefixes shared with the real configuration.

The comparison in CPU utilization between our shadow kernel and the standard kernel are shown in Figure 8. The machines are hyperthreaded, so we increase the data rate until the CPU handling the input interface interrupts reaches 100% utilization. The reported value is the overall CPU utilization including both CPUs. Our implementation does not noticeably increase CPU utilization

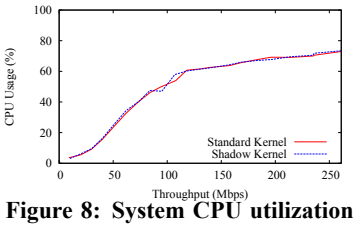

Figure 8: System CPU utilization for varying traffic rates (300-byte packets).

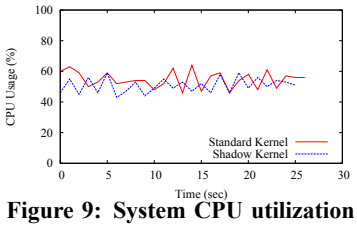

Figure 9: System CPU utilization for FIB updates (100 Mbps, 300-byte packets).

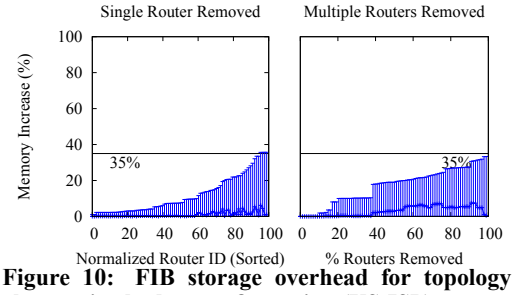

Figure 10: FIB storage overhead for topology changes in shadow configuration (US-ISP).

as compared to the standard kernel. Note that the full 1 Gbps capacity is not reached due to the smaller packet sizes. Our implementation can achieve the same rates cited by [4] for 1430-byte packets.

FIB Storage Overhead: One concern is that the number of FIB entries will be increased. However, for most networks, the network prefixes are relatively fixed, and thus should appear in both real and shadow configurations. Accordingly, the number of entries and IP prefix lookup costs do not increase significantly. We incur a storage overhead only if the shadow and real configurations specify different next-hop behaviors, since otherwise only a single FIB entry is required.

Scenario	Changed FIB Entries	Memory Overhead
Remove NEWY↔WASH	13074	4.7%
Remove LOSA	4467	1.6%
Remove KANS	19874	7.2%

Table 2: FIB storage overhead (Abilene).

Table 2 shows the increase in FIB size across all routers for the Abilene network due to configuration changes made in the shadow configuration. Both the real and the shadow configurations have more than 90,000 FIB entries. We observe these topology changes lead to a small overall storage increase. Although in the theoretical worst case the storage may double, in our real implementation the increase is less than 8% due to the sharing between real and shadow next-hops. We anticipate that this sharing is common.

To evaluate the scenarios for a larger network, we use the configuration of US-ISP, a large tier-1 ISP. We use its backbone topology, OSPF link weight configuration, and external routes to compute the FIB size at each router. Each router has a few hundreds of thousands of FIB entries. The presented memory overhead is based on data structure sizes in the Linux kernel implementation.

Figure 10 shows the results for two scenarios. The vertical bar denotes the maximum and minimum per-router memory overhead, and the dark points denote the average memory overhead over all routers. In the first scenario, we show the memory overhead when only a single router at a time is removed from the network in the shadow configuration. We observe that in the worst cases, the routers with the worst FIB overhead have their FIB storage increased by no more than 35%. These "worst" routers are often stub routers with low connectivity in the topology. Thus, one way to reduce their storage, if necessary, is forwarding entry aggregation or virtual address mapping. The average is much lower, under 5% in most cases. Next, we show the FIB memory overhead as routers are removed one-by-one in the shadow configuration. There is no case in which the router with the worst FIB overhead has its FIB storage increased by 35%. The average overhead is much lower than the worst case.

FIB Update Overhead: Since we also extend the FIB insertion and deletion routines to handle shadow configuration, we also evaluate the performance when the FIB is being frequently updated. We use the same setup as the prior experiment on FIB data forwarding processing overhead, but we also randomly add and delete between 1 and 100 routes in the real configuration in 10.0.0.0/8 each second at the intermediate router as it is forwarding traffic.

Figure 9 shows the results. Again, there is no noticeable difference between supporting shadow configuration or not. Note that when running this experiment without the FIB updates, the CPU utilization for both our shadow kernel and the standard kernel fluctuates much less, but both remain nearly identical for the duration of the experiment.

8.3 Usage of Shadow Configurations

We now demonstrate the effectiveness of shadow configuration in three usage scenarios.

Equipment Maintenance: A usage scenario of shadow configuration is equipment maintenance. We use this scenario to demonstrate the performance of our commitment protocol.

In this experiment, we use the Abilene topology and configurations, and generate transit traffic according to the CAIDA traces from peering routers configured at New York, Seattle, and Atlanta. Emulab's delay nodes are used to model propagation delays.

In this scenario, we bring the Kansas router down for maintenance and return it to service when finished. The real configuration is initially cloned to the shadow configuration. Next, we disable OSPF in the shadow configuration on the Kansas router, wait 10 seconds, then commit at time 48. The network operator may then safely perform upgrades, and restart it when finished. Once the shadow configuration with Kansas enabled converges, the configurations are again swapped, causing the Kansas router to again forward transit traffic.

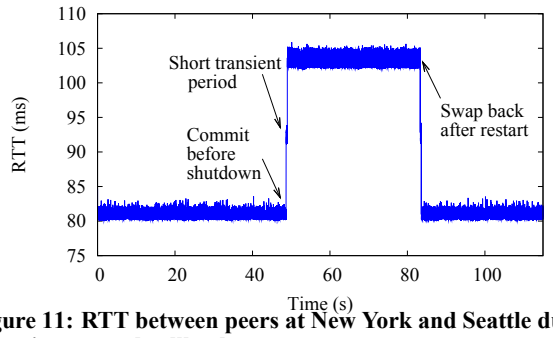

Figure 11: RTT between peers at New York and Seattle during commitment and rollback.

Figure 11 shows the round-trip time between the peering routers at New York and Seattle. Note that there are three modes of operation at 82 ms, 92 ms, and 102 ms due to the Abilene routers asynchronously executing the swap in phase 3 of the commitment protocol. This arises because the ICMP echo request follows a different path than the reply, due to tagging at the ingress routers. The intermediate transition phase lasts for a short time, but the packet forwarding behavior during this transition phase is clean and controlled. Importantly, there are no packet losses.

Our commitment protocol is executed over serial consoles to each router. We are currently developing a protocol to access the routers' configuration terminals using both the shadow and real configurations such that the protocol is resistant to misconfiguration in one of the two configurations.

Parameter Tuning: The next usage scenario for shadow configurations we evaluate is parameter tuning. We update a set of OSPF link weights simultaneously. The real configuration uses Abilene's normal link weights, and we then change *all* of the link weights in the network to be the inverse of the bandwidth (*i.e.*, all equal in the Abilene case) using two methods: (1) manual configuration and (2) shadow configurations.

To perform the manual configuration, we update the link weights using parallel Telnet sessions, which takes about 4 seconds. With shadow configurations, we update the link weights in the shadow configuration, wait 20 seconds for convergence, and then execute the commitment protocol.

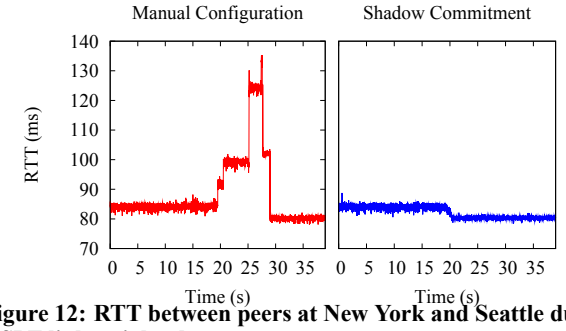

Figure 12: RTT between peers at New York and Seattle during OSPF link weight change.

We immediately notice in Figure 12 that using the shadow configuration avoids the reconvergence process. Under manual configuration, the round-trip time between the peer routers at Seattle and New York fluctuate between 83 ms and 135 ms before settling on the converged value of 80 ms. Using shadow configurations provides a quick and smooth transition since convergence takes place in the shadow configuration prior to commitment.

New Service Testing: The last usage scenario we evaluate is testing of new services. We use this scenario to demonstrate our packet cancellation technique and show that (1) there is little effect on transit traffic and (2) performance measurements on shadow traffic are indicative of its true performance.

	Real	Shadow
Config	Abilene configuration	Abiliene configuration with 4 link weights adjusted for load balancing
Traffic	Transit traffic generated from CAIDA traces with 30% utilization on bottleneck link	Duplicated real traffic and UDP streaming video with 6 servers and 12 clients

Table 3: New service testing experiment setup.

Setup: In this scenario, a network operator is testing a streaming video application under a new set of OSPF link weights. Our setup is shown in Table 3.

UDP packet traces are constructed using a high-definition movie trailer and the VideoLAN [48] VLC software. The movie trailer alternates between complex scenes (using up to 22 Mbps) and a black background with text (using 450 Kbps).

With this setup, there exist time intervals when the combined (raw) real and shadow traffic intensity exceeds link capacity on some links, meaning bandwidth partitioning is not effective for obtaining accurate performance results.

Delay nodes are removed from the Emulab experiment since we want to observe small-scale variations over multihop flows. We also use 100 Mbps links to more easily observe delay variation given the resolution of our measurement tools.

Safety for Transit Traffic: Our experiments show that the shadow traffic has little effect on the real traffic when packet cancellation is enabled. We show the measured performance for two paths. Figure 13 shows the delay variation for traffic from Seattle to Chicago. The real traffic performance with packet cancellation enabled overlaps the performance when only real transit traffic is present, while the delay variation rises sharply up to about 15 ms without packet cancellation. Similar behavior is observed between Salt Lake City and Atlanta (Figure 14), where the round-trip time increases from under 1 ms up to 20 ms without packet cancellation. Round-trip time is largely unaffected with packet cancellation enabled.

Shadow Performance Accuracy: We next show that packet cancellation provides accurate performance measurements despite the presence of real transit traffic. In our experiment, there are multiple streaming sessions that have incorrect measurements when packet cancellation is not enabled. For example, the throughput measurement for the video stream from Houston to Chicago (Figure 16) shows the correct value of 22 Mbps. Without packet cancellation, the measurements incorrectly show that only 18 Mbps is supported.

Multiple video streams in our experiments also show that loss rates with packet cancellation are indicative of the true value. Figure 15 shows the loss rate of streams served by Salt Lake City. Without packet cancellation, it is erroneously reported to be up to 14%, while packet cancellation correctly has no losses.

Fine-grained Accuracy: Finally, we show in more detail how real traffic is protected and performance characteristics of shadow traffic are preserved under packet cancellation. We use a simple illustrative topology shown in Figure 5 and the CAIDA traces. Figure 17 shows CDFs of delay variation for both real and shadow traffic. The observed performance for real traffic is largely unchanged as we increase shadow traffic until raw total traffic intensity reaches link capacity (100%). Similarly, delay variations for shadow traffic closely approximate its actual behavior.

9. RELATED WORK

The importance of configuration management has motivated many recent studies and proposals on this topic (*e.g.*, [3, 7, 14, 36]). Due to space limitation, we review only the most directly related work.

Static Analysis and Simulation Tools: These studies are very useful for configuration validation in many settings (*e.g.*, [15, 17, 22, 37, 53]). Shadow configurations provide a complementary tool and have several advantages such as scalability. A particular advantage is that it does not depend on an abstract model of the real network and therefore will not miss configuration errors caused by the inconsistency between the real network and the model.

- Since static analysis and simulation tools often depend on an abstract model of the real network, they may miss configuration errors caused by the inconsistency between the real network and the model (*e.g.*, forgotten network equipment or network connectivity).
- The final configuration depends on the whole network processing environment: the hardware, firmware, and software features (including the bugs!) of the routers. Typical networks are heterogeneous networks consisting of equipment from multiple vendors with distinct hardware, firmware and software features. As an example of the complexity, a survey [36] of 31 production networks found that over 200 different software versions were running on multiple hardware platforms. As another example, some routers may also offer special non-standard features (*e.g.*, Cisco-specific BGP decision steps in addition to the conventional BGP decision process [53]). As yet another example, the Cisco document [10] reports a common OSPF routing problem related with forwarding addresses. The reachability issue was caused by a bug in Cisco IOS before Release 12.1(3).
- The interactions of multiple services can be a source of configuration errors. Today's networks are complex and certain behaviors may only arise when two features interact. As a simple example, the routing protocol can compute a backup path but all packets rerouted to the backup path can be dropped by a packet

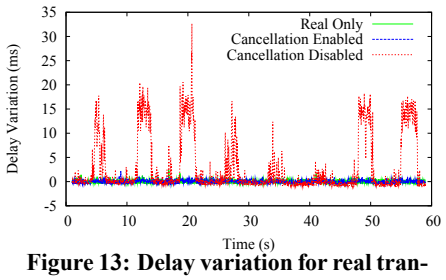

Figure 13: Delay variation for real transit traffic (Seattle→Chicago).

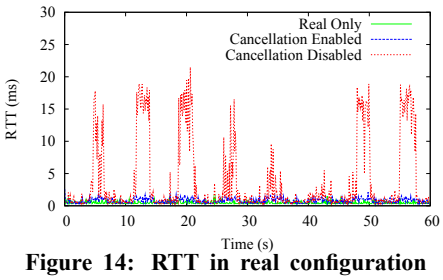

Figure 14: RTT in real configuration (Salt Lake City→Atlanta).

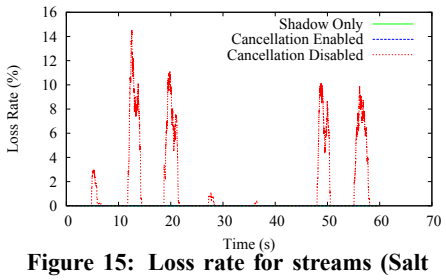

Figure 15: Loss rate for streams (Salt Lake City server).

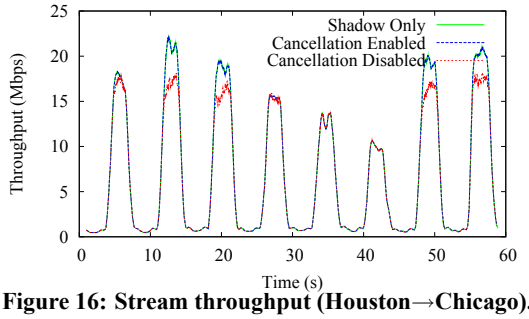

Figure 16: Stream throughput (Houston→Chicago).

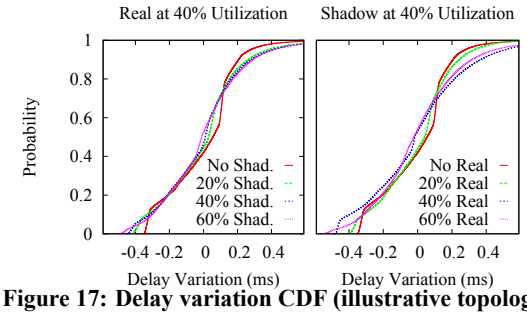

Figure 17: Delay variation CDF (illustrative topology).

filter. Most tools typically focus on a single piece or set of functionality (*e.g.*, routing, access control, or QoS).

- The configuration evaluation system should scale as the network size grows. Many simulators can have limited capacity as compared to the network it is simulating, making it difficult and inefficient to run tests with large demands.
- On the forwarding plane, the performance depends on traffic demand pattern, FIBs, hardware capability, and software implementations. If a tool conducts performance evaluation based on a very coarse-grained model (*e.g.*, a link is characterized by two simple numbers such as propagation delay and bandwidth), performance problems may not be revealed.

Comparison with Virtualization: One possibility to address some of the problems of static analysis or simulation is to utilize the recently proposed network virtualization techniques such as PlanetLab slices [9], VINI [4], CABO [16], or Juniper's Logical Router feature [27].

A key difference between shadow configurations and these virtualization techniques such as PlanetLab and VINI is objective. Specifically, the main objective of the aforementioned virtualization studies is on building general, flexible experimental facilities for researchers. Thus, the focus is more on providing capabilities such as flexible (overlay) experimental topology constructions, and flexible routing and forwarding (or programmability on the network elements in general). Also, as application-oblivious experimental facilities, general virtualization would need to consider each virtual network as independent and thus a key issue is to provide isolation among them.

The objective of shadow configurations, on the other hand, is to provide an effective and easy-to-use capability for network operators in the specific but important domain of configuration testing. A configuration may have multiple virtual networks (*e.g.*, VPNs) constructed inside according to the configuration files in the shadow configuration, but this is transparent to us. In a specific domain, general flexibility may not be necessary, and the overhead may be too high in production networks.

Conceptually, one can consider the real and the shadow configurations as two virtual networks. From this perspective, our main contribution is a novel usage of virtualization for the important domain of network configuration management. Instead of forcing users to adapt to the underlying technique (*i.e.*, asking operators to explicitly construct virtual experiments for configuration management), we use the user interface that a network operator is already using to provide an easy-to-use tool. The most important design objective of shadow configurations is simplicity and ease of use, as this is the objective prompted in many settings (*e.g.*, the collective plea of LISA 2006 Configuration Workshop). In addition, we enrich the domain of application of virtualization by creating two correlated virtual networks for a single task. Our novel capabilities of *packet cancellation* and *configuration commitment* provide examples where it may be beneficial to extend beyond general virtualization where virtual networks are typically considered independent.

One recent work that also applies virtualization to network configuration management is VROOM [50]. Their objective is to reduce logical topology change by allowing dynamic binding between virtual routers to physical routers. This objective is different from our objective, which in one sentence is to discover issues in a new configuration before its real deployment.

10. CONCLUSION AND FUTURE WORK

In this paper, we presented the novel idea of shadow configurations. We developed novel techniques such as packet cancellation and shadow commitment to substantially improve the capability of configuration evaluation and management. There are many avenues for future exploration. In particular, we would like to conduct an extensive evaluation on using shadow configuration for automating configuration debugging and network diagnostics. We would also like to study their usage in carrier-grade installations.

11. ACKNOWLEDGMENTS

Richard Alimi and Y. Richard Yang were supported in part by grants from the U.S. NSF. We would like to thank Yuan Dong and Ehab Al-Shaer for greatly beneficial initial discussions. The main algorithm of FIB analysis was contributed by Hao Wang. He has also made many other key observations. We are also grateful to Jim Aspnes, Matthew Caesar, Charles Kalmanek, Karthik Lakshminarayanan, Erran Li, Jennifer Rexford, Zhong Shao, Avi Silberschatz, and Xiaowei Yang for valuable suggestions.

12. REFERENCES

[1] M. Agrawal, S. Bailey, A. Greenberg, J. Pastor, P. Sebos, S. Seshan, J. van der Merwe, and J. Yates. Routerfarm: Towards a dynamic, manageable network edge. In *ACM SIGCOMM INM*, 2006.

[2] P. Anderson and A. Scobie. Large scale Linux configuration with LCFG. In *Proc. of ALS*, Berkeley, CA, 2000.

[3] H. Ballani and P. Francis. Conman: A step towards network manageability. In *ACM SIGCOMM*, Kyoto, Japan, Aug. 2007.

[4] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI veritas: Realistic and controlled network experimentation. In *ACM SIGCOMM*, Pisa, Italy, Sept. 2006.

[5] M. Caesar, L. Subramanian, and R. Katz. A case for an Internet health monitoring system. In *HotDep*, 2005.

[6] CAIDA: Cooporative association for Internet data analysis. `http://www.caida.org/`.

[7] X. Chen, Z. M. Mao, and K. van der Merwe. Towards automated network management: Network operations using dynamic views. In *ACM SIGCOMM INM*, 2007.

[8] B.-Y. Choi1, S. Moon, Z.-L. Zhang, K. Papagiannaki, and C. Diot. Analysis of point-to-point packet delay in an operational network. In *Proceedings of IEEE INFOCOM '04*, Hong Kong, China, Apr. 2004.

[9] B. Chun and *et al.* Planetlab: an overlay testbed for broad-coverage services. *ACM CCR*, 33(3):3–12, 2003.

[10] Cisco Systems. Common routing problem with OSPF forwarding address. `http://www.cisco.com/warp/public/104/10.pdf`.

[11] Cisco Systems. Network solutions integrated test environment: Delivering on the promise of innovation. URL: `http://www.cisco.com/application/pdf/en/us/guest/netsol/ns522/c2072/cdccont_0900aecd80458f98.pdf`, 2006.

[12] M. Crovella, C. Lindemann, and M. Reiser. Internet performance modeling: the state of the art at the turn of the century. *Performance Evaluation*, 42:91–108, 2000.

[13] A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot. Netdiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data. In *In the Proceedings of CoNEXT'07*, Dec. 2007.

[14] W. Enck, P. McDaniel, Subhabrata, P. Sebos, S. Spoerel, A. Greenberg, S. Rao, and W. Aiello. Configuration management at massive scale: System design and experience. In *USENIX*, Santa Clara, CA, June 2007.

[15] N. Feamster and H. Balakrishnan. Detecting BGP configuration faults with static analysis. In *NSDI*, May 2005.

[16] N. Feamster, L. Gao, and J. Rexford. How to lease the Internet in your spare time. *ACM CCR*, 37(1):61–64, 2007.

[17] A. Feldmann. Netdb: IP network configuration debugger/database. Technical report, AT&T Research, July 1999.

[18] A. Feldmann and J. Rexford. IP network configuration for intradomain traffic engineering. *IEEE Network Magazine*, pages 46–57, Sept./Oct. 2001.

[19] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional IP routing protocols. *IEEE Communication Magazine*, Oct. 2002.

[20] P. Francois and O. Bonaventure. Avoiding transient loops during IGP convergence in IP networks. In *IEEE INFOCOM*, Miami, FL, Apr. 2005.

[21] P. Francois, M. Shand, and O. Bonaventure. Disruption free topology reconfiguration in OSPF networks. In *IEEE INFOCOM*, Anchorage, AK, May 2007.

[22] H. Hamed, E. Al-Shaer, and W. Marrero. Modeling and verification of IPSec and VPN security policies. In *IEEE ICNP*, Boston, MA, Nov. 2005.

[23] M. Handley, E. Kohler, A. Ghosh, O. Hodson, and P. Radoslavov. Designing extensible IP router software. In *NSDI*, May 2005.

[24] U. Hengartner, S. Moon, R. Mortier, and C. Diot. Detection and analysis of routing loops in packet traces. In *IMW*, Marseille, France, Nov. 2002.

[25] L. Huang, X. Nguyen, M. Garofalakis, J. Hellerstein, M. Jordan, A. D. Joseph, and N. Taft. Communication-efficient online detection of network-wide anomalies. In *IEEE INFOCOM*, Anchorage, AK, May 2007.

[26] W. John and S. Tafvelin. Analysis of Internet backbone traffic and header anomalies observed. In *IMC*, Aug. 2007.

[27] Juniper Networks. Intelligent logical router service. URL: `http://www.juniper.net/solutions/literature/white_papers/200097.pdf`, Oct. 2004.

[28] S. Kandula, D. Katabi, and J.-P. Vasseur. Shrink: A tool for failure diagnosis in IP networks. In *MineNet*, Aug. 2005.

[29] Z. Kerravala. As the value of enterprise networks escalates, so does the need for configuration management, Jan. 2004.

[30] Z. Kerravale. Configuration management delivers business resiliency. The Yankee Group, 2002.

[31] E. Kiciman and L. Subramanian. A root cause localization model for large scale systems. In *IEEE INFOCOM*, Anchorage, AK, May 2007.

[32] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. IP fault localization via risk modeling. In *NSDI*, San Francisco, CA, May 2005.

[33] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. Detection and localization of network black holes. In *IEEE INFOCOM*, Anchorage, AK, May 2007.

[34] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica. Achieving convergence-free routing using failure-carrying packets. In *ACM SIGCOMM*, Kyoto, Japan, Aug. 2007.

[35] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level Internet path diagnosis. In *SOSP*, Bolton Landing, Oct. 2003.

[36] D. A. Maltz, G. Xie, J. Zhan, H. Zhang, G. Hjalmtysson, and A. Greenberg. Routing design in operational networks: A look from the inside. In *ACM SIGCOMM*, Portland, OR, Aug. 2004.

[37] S. Narain. Network configuration management via model finding. In *Proceedings of LISA*, pages 155–168, 2005.

[38] S. Narain. Overview of configuration validation. Presentation at LISA 2006 Configuration Workshop, Dec. 2006.

[39] A. Nucci, S. Bhattacharyya, N. Taft, and C. Diot. ICP link weight assignment for operational tier-1 backbones. *IEEE/ACM Transactions on Networking*, Aug. 2007.

[40] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why Internet services fail and what can be done about these? In *USENIX USITS*, Oct. 2003.

[41] Quagga Software Routing Suite. `http://www.quagga.net/`.

[42] B. Quoitin, S. Uhlig, and O. Bonaventure. Using redistribution communities for interdomain traffic engineering. In *QoFIS'02 LNCS 2511*, Oct. 2002.

[43] M. Roughan, T. Griffin, M. Mao, A. Greenberg, and B. Freeman. Combining routing and traffic data for detection of IP forwarding anomalies. In *SIGMETRICS*, New York, NY, June 2004.

[44] University of Oregon Route Views Project. `http://www.routeviews.org/`.

[45] A. Soule, K. Salamatian, and N. Taft. Combining filtering and statistical methods for anomaly detection. In *IMC*, Aug. 2005.

[46] M. Steinder and A. S. Sethi. A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53:165–194, 2004.

[47] L. Tatman. Incorporating routing analysis into IP network management. URL: `http://www.agilent.com/labs/features/2003_wp_roca.pdf`, May 2003.

[48] VideoLAN - VLC media player. `http://www.videolan.org/`.

[49] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush. A measurement study on the impact of routing events on end-to-end internet path performance. *ACM CCR*, 36(4), 2006.

[50] Y. Wang, J. van der Merwe, and J. Rexford. VROOM: Virtual ROuters On the Move. In *HotNets-VI*, Atlanta, GA, Nov. 2007.

[51] A. Whitaker and D. Wetherall. Forwarding without loops in icarus. In *IEEE OEPNARCH*, 2002.

[52] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *OSDI*, Boston, MA, Dec. 2002.

[53] G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford. On static reachability analysis of IP networks. In *IEEE INFOCOM*, Miami, FL, Apr. 2005.

[54] A. Zeller. Yesterday, my program worked. today, it does not. Why? In *ESEC / SIGSOFT FSE*, pages 253–267, 1999.

[55] B. Zhang, T. S. E. Ng, A. Nandi, R. Riedi, P. Druschel, and G. Wang. Measurement-based analysis, modeling, and synthesis of the internet delay space. In *IMC*, Aug. 2006.

[56] Z. Zhong and *et al.* Avoiding transient loops through interface-specific forwarding. In *IWQoS*, 2005.