# Configuration Management at Massive Scale: System Design and Experience

William Enck, Thomas Moyer, Patrick McDaniel, Subhabrata Sen, Panagiotis Sebos, Sylke Spoerel, Albert Greenberg, Yu-Wei Eric Sung, Sanjay Rao, and William Aiello

***Abstract*—The development and maintenance of network device configurations is one of the central challenges faced by large network providers. Current network management systems fail to meet this challenge primarily because of their inability to adapt to rapidly evolving customer and provider-network needs, and because of mismatches between the conceptual models of the tools and the services they must support. In this paper, we present the PRESTO configuration management system that attempts to address these failings in a comprehensive and flexible way. Developed for and used during the last 5 years within a large ISP network, PRESTO constructs device-native configurations based on the composition of *configlets* representing different services or service options. Configlets are compiled by extracting and manipulating data from external systems as directed by the PRESTO configuration scripting and template language. We outline the configuration management needs of large-scale network providers, introduce the PRESTO system and configuration language, and reflect upon our experiences developing PRESTO configured VPN and VoIP services. In doing so, we describe how PRESTO promotes healthy configuration management practices.**

***Index Terms*—Computer network management.**

## I. Introduction

CONFIGURATION management is among the largest cost-drivers in provider networks. Such costs are driven by the immense complexity of the supported services and infrastructure. For a large provider, thousands of enterprises with diverse services and configurations must be seamlessly and reliably connected across huge geographic areas and rapidly evolving networks. Moreover, the initial "turn-up" installation and subsequent support of a single customer may span many organizations and systems internal to the provider. The stakes for the supported enterprise are extremely high: an outage may result in loss of business, delays in "getting to revenue" since service turn up precedes revenue, failure to meet contractual obligations, or disruption of key organizational workflows.

Given the complexity and stakes, it may be surprising that the common configuration management practice involves pervasive manual work or ad hoc scripting. The reasons for this are multi-faceted. From a provider perspective, every customer is in some ways unique. Though service orders have a lot in common, many new installations made to realize those orders represent unique combinations of services and network configurations. Evolving requirements within customer networks, stale, incomplete, or imperfect information, and service interactions make both "turn-up" as well as ongoing maintenance of configurations complex and error-prone processes. Moreover, the devices in the network and the definition of services they support change at a dizzying rate. New firmware versions, customer requirements, or supported applications appear every day. Market demands further dictate that the time-to-market for new services is a critical driver of revenue: delays caused by tool configuration, extension, or development can mean the difference between profitability and loss. In short, there is an unserved need in provider networks for tools that address these complex and sometimes contradictory challenges while constructing service configurations.

In the PRESTO configuration management system [1], network designers specify services in a template language, which are subsequently combined by network operators to create specific router configurations. PRESTO develops network device configurations from composed collections of *configlets* that define the services to be supported by the target device. Our general-purpose scripting and configuration template language extracts specific information from external systems and databases to transform this information into complete device configurations as directed by the PRESTO compiler. Extensive interactions with diverse engineering teams charged with managing operational IP networks led us to the conclusion that, to gain wide buy-in and adoption, the PRESTO language must adhere closely to the complex and often low level configuration languages supported by network device vendors (e.g., for Cisco devices, the IOS command language). PRESTO empowers network designers, "power users" comfortable with native network device configuration languages, and automates the unambiguous translation of their design rules into precise network configurations. Specifically, the PRESTO system generates complete device-native configurations, which can then be downloaded into a device and archived by network operators.

In this paper, we present the motivation, design, and workflow of the PRESTO configuration management system. We outline the challenges faced by a large network provider in installing and maintaining millions of diverse devices for thousands of customers and organizations, and reflect on the failures of past network management systems to address these needs. We outline the PRESTO workflow and configuration

Manuscript received 4 April 2008; revised 30 October 2008. A preliminary version of this article appeared in the Proceedings of the 2007 USENIX Annual Technical Conference in Santa Clara, CA.

William Enck, Thomas Moyer, and Patrick McDaniel are with Pennsylvania State University (e-mail:{enck,tmmoyer,mcdaniel}@cse.psu.edu).

Subhabrata Sen, Panagiotis Sebos, and Sylke Spoerel are with AT&T Research (e-mail: {sen,psebos}@research.att.com, sspoerel@att.com).

Albert Greenberg is with Microsoft Research (e-mail: albert@microsoft.com).

Yu-Wei Eric Sung, and Sanjay Rao are with Purdue University (e-mail: {sungy,sanjay}@ecn.purdue.edu).

William Aiello is with University of British Columbia (e-mail: aiello@cs.ubc.ca).

Digital Object Identifier 10.1109/JSAC.2009.090408.

language and reflect upon our experiences implementing Virtual Private Network (VPN) and Voice-over-IP (VoIP) within PRESTO. In doing so, we describe how PRESTO promotes healthy configuration management practices.

A significant effort in network management involves the development of "greenfield" configs–configuration of new routers or services in a new installation. Hence, PRESTO's initial focus, and the focus of this paper, is on creating greenfield configurations. Support for long term maintenance of routers, called "brownfield" configuration, is currently evolving within the PRESTO framework. While providing generic support for brownfield configuration remains the subject of our ongoing work, PRESTO currently supports limited brownfield configuration and will be briefly discussed.

The PRESTO system evolved out of decades of experience in network management. Configuration management is about more than just getting routing and filtering correct. It must meld together many different services that exhibit subtle interactions and dependencies. Therein lies the challenge of configuration management in a provider network—*How do we glue together many information sources of myriad organizations in real time to build a functioning device configuration?* Revisited in the following section and throughout, it is the lessons gleaned from our experiences in meeting that goal that drive the PRESTO design.

The remainder of the paper proceeds as follows: Section II discusses motivation and requirements; Section III describes the PRESTO system; Section IV describes our experience using PRESTO for two real applications within enterprise networks supported by a large ISP; Section V discusses related work; and Section VI concludes.

## II. Configuration Automation

In this section, we discuss the need for automation by describing current best practices and their limitations. We then describe the challenges an automated configuration generation system must face in large provider networks.

A router configuration file provides the detailed specification of the router's configuration, which in turn determines the router's behavior. Often, the configuration file is a representation of the sequence of specific commands that, if typed through the command line interface, determine the wide set of interdependent options in router hardware and software configuration. In practice, this may represent thousands of lines of complex commands per router. These configuration files are text artifacts—described in a device specific command language, with a device specific syntax in a human and machine readable format, in some cases in XML. Note that a plethora of network devices beyond routers, e.g. Ethernet switches and firewalls, rely on configuration files of this type. To some degree, this state of affairs reflects natural technology evolution and the marketplace—networks started (and often still start) small and therefore often gravitated toward manual or (ad hoc) scripted configuration.

Today's configuration languages offer a myriad of complex options, typically described in precise low level device-specific languages, such as Cisco's IOS command language [2]. While the learning curve for such languages might be steep and the cost of inadequate learning severe (small slipups may cause large network outages), these languages are in extremely wide use for the entire lifecycle of network management – starting with configuration, but encompassing all other aspects, including performance, fault and security management. The tack that the PRESTO system takes is to leverage these "native" languages, and empower the user of these languages to enforce the precise translation of design intent into detailed device configuration.

Our interactions with network designers revealed that using templates to describe design intent is essentially universal. That is, designers create parameterized chunks of design configurations to describe intent. Accordingly, PRESTO provides full and flexible support for template creation in the native device configuration language.

### A. Need for Automation

Decades of experience in network management have taught us that manual configuration practices are limited in the following ways, i.e., *the configuration process is*:

• *costly, time-consuming, and unscalable:* There is a significant initial investment in the interpretation and documentation of network standards and device-specific interfaces in developing any new service or support for a device. The result of that investment is a "model" configuration document (sometimes termed an Engineering and Troubleshooting Guidelines (ETG) document) used by enablers[1] to manually configure each target device. Typically performed by a large network engineering organization, and depending on the complexity of the service or device, this process can take many person-months of effort to complete and is resource expensive. The subsequent manual application of the model configurations to customer networks is also costly—some large customers may have tens of thousands of network elements, and applying a new configuration to even a fraction of them verges on the intractable.

• *prone to misinterpretation and error:* Even under the best of circumstances, engineering guidelines will not be perfect. Because network designers cannot anticipate all possible target environments, the guidelines are necessarily ambiguous, sometimes imprecise, and often subject to multiple interpretations. Thus, different enablers may interpret the same rules differently or adopt different local conventions. Differences between interpretations can and often do result in configuration mismatch errors. Making matters worse, while some errors might be easier to detect, others might have no immediate effect. These latter configuration problems are the most vexing, as they may become manifest at periods of high load (possibly the worst possible time) or introduce undetected security vulnerabilities.

• *fraught with ambiguous, incorrect, changing or unavailable input data:* Configuration information is not only spread across multiple data sources, but may be incomplete and imperfect. For example, customer order databases may not reflect the latest needs of the customer, e.g., order updates may only exist

[1]*Enablers* are the personnel who implement a given service, either staff on-site or within a provider's organization.

as emails to human contacts and may not quickly (or ever) be reflected in a database. As another example, information such as IP address assignments may be missing at the time of initial configuration. Finally, rules might be ambiguous. We have, for example, encountered examples where a particular service mandated that a site may have dual routers with ISDN backup, but it was not obvious which router must be the backup and which the primary.

## *B. Requirements*

The pervasive practices and technical organizational problems detailed above makes automation in provider networks difficult. These issues lead to the following requirements, i.e., *the configuration process must*:

• *Support existing configuration languages:* While there have been many prior efforts at automating configuration generation, most have focused on developing abstract languages or associated formalisms to specify configurations in a vendor neutral fashions, e.g., IETF standard SNMP MIBs (Simple Network Management Protocol, Management Information Bases) [3], the Common Information Model (CIM) [4], and the Common Management Information Protocol (CMIP) [5]. These information models define and organize configuration semantics for networking/computing equipment and services in a manner not bound to a particular manufacturer or implementation. However, such generalized abstractions invariably introduce layers of interpretation between the specification and device. Gaps open between general abstract specifications and the concrete language of specific device configuration. It is very difficult to avoid extending or creating specialized common models to describe the realities of today's rapidly evolving devices and services. The artifacts of efforts to create standards or libraries often lag the marketplace. In truth, network experts often do not have the time or inclination to understand such abstractions, and today nearly universally find that working within native configuration interfaces is much more efficient for both initial installation and later maintenance.

• *Scale with options, combinations, and infrastructure:* Customer configurations are dependent on, in particular, selected service offerings, devices, firmware revisions, and local infrastructure. For example, consider a site connecting to a provider backbone. The seemingly simple customer order has many options—does the customer require multiple routers to connect to the backbone or just one? Should each have multiple links or one? Further, each router may have several WAN (Wide Area Network) and LAN (Local Area Network) facing interfaces, and each interface may admit specific naming conventions that depend on the router model and the WAN card. The physical local infrastructure (e.g., routers and network topologies), will often have major impact on the workflow and content of the configuration.

• *Support heterogeneous and diverse data sources:* Putting together a router configuration involves collecting all necessary router configuration information. Such configuration information may not be all available in one central repository at the time the information is needed, but rather maybe distributed amongst a variety of databases, which are populated by upstream workflows. Take, for example, the customer order database. The customer information may itself have arrived at different times, and may be split across various forms. In large operational networks, information regarding customer orders and the resulting router deployment and maintenance is potentially spread across various systems spanning many internal organizations. An automated configuration system has to be cognizant of the diversity of information sources (and quality of data). Importantly, these data sources typically have their own persistent databases (each representing large investments), and a configuration management solution faces a huge real-world hurdle if it were to attempt to replace or replicate these databases, or even add a new persistent database rather than extend an already existing one. To the greatest extent possible, a configuration management system should strive to be stateless if it is to succeed in diverse operational environments.

## III. The PRESTO System

Two kinds of users interact with PRESTO—domain experts, which define services, and provisioners, which combine services for specific routers. Domain experts initially codify the configuration services and options in *active templates*. These are the PRESTO equivalent of the ETG, where execution of the "active" template by the compiler directs the interpretation of the service definition for a particular environment. At installation time, provisioners obtain necessary configuration information from customers and other sources, that, when combined with the templates written by the domain expert, produce the end router configuration.

In practice, deployment is a multi-stage process including requirements reconnaissance, initial configuration creation, and device turn-up. PRESTO is concerned with the second part of the process, the creation of the configuration. The configuration creation process begins with a batch of one or more new configuration requests resulting from a network upgrade request or customer order. The requests are submitted to PRESTO as a collection of specification inputs, where the relevant environment data is provided as direct inputs or extracted from supplementary inventory and configuration databases. The data is cleansed and projected into a service or device (application-specific) data model created for the target configurations. Finally, an active template is executed for the set of specified target devices resulting in complete device-native configurations.

As discussed in Section II, automation is paramount to massive scale configuration management. The remainder of this section describes how PRESTO achieves automation and realizes the requirements laid out in Section II-B. We first present a template language based on *configlets* (modular sub-parts of configuration text) that allow domain-experts to design a scalable model for unambiguous configuration specification. We then show how a two step architecture can normalize and cleanse configuration inputs and promotes provisioner feedback for process standardization.

## *A. PRESTO Template Language*

The PRESTO template language lays out the foundation over which the rest of the system is built. This allows domain experts to leverage knowledge of flexible native configuration languages, e.g., Cisco IOS, while creating useful abstractions, defining services, which take the form of *configlets* in PRESTO. Interestingly, this is in direct opposition to traditional network management interfaces which provide a single abstraction to which any policy would have to adhere. PRESTO provides the following key characteristics:

• *Data Modeling:* Router configurations are modeled as an amalgamation of services. A per-application data model uses a relational database schema to capture router characteristics specific to the desired service rules.

• *Rich Template language:* Service rules are defined with respect to information available in the application specific database. Templates present a straightforward definition of service rules using data driven loops, conditionals, and variable substitution.

• *Support for Template Decomposition and Assembly:* PRESTO allows the architect to write multiple smaller templates targeted for very specific elements of a configuration, i.e., services.

Before discussing specific details, we provide an example scenario to aid in the understanding of language constructs and design motivations. Consider the configuration of a gateway router. The gateway connection may have one or more external connections. If there are multiple connections, they may be dispersed across multiple routers. Hence, these routers require configuration knowledge of the other routers participating in the gateway connection, e.g., IP addresses, to coordinate failover, e.g., HSRP [6]. The template language must support these relationships between connections on one router and between routers.

We now discuss each part of the template language in turn. After discussing the core language concepts, we introduce additional language features that enable better software engineering practices.

*1) Data Model:* The PRESTO template language revolves around the data model. The templates require access to small data chunks describing router properties. Furthermore, multi-router relationships dictate a need to perform quick lookups for peer specific information. Such a capability is required for instance when configuring one router (eg., a spoke) involves extracting information for another router (eg., the hub). A relational database provides just this capability: router properties are stored in table fields and accessed as variables; peer router properties are queried by specifying the router hostname. Hence, the data model becomes a database schema. We refer to this database as the provisioning database and provisioning relational database schema where necessary to remove ambiguity.

The schema definition is application dependent. Each application has different requirements on data accessibility. It is no surprise that defining the schema is the most delicate part of applying PRESTO to a new application, hence complete flexibility is required. Despite the supported flexibility, past experience has resulted in a few recommended guidelines. The data model should contain a `ROUTER` table indexed by a globally unique identifying value, e.g., the router hostname. One row will exist for each router in a provisioning request. The `ROUTER` table should contain the bulk properties; however, whenever multiple instances of a property occur, a new table should be created. For example, multiple LAN or WAN interfaces are semantically equivalent. Sub-router tables should use a multi-column primary key consisting of the `ROUTER` table index and a unique identifier, e.g., interface number. Upon querying the database for all LAN interface records matching a specific router hostname, the template language produces an interface loop. For example; suppose `LAN` is a table holding all LAN-facing inputs. Then

```
SELECT * FROM LAN WHERE ROUTER=THIS_ROUTER
```

selects each LAN-facing interface on a given router. Iteration specifics and syntax are discussed below.

*2) Variable Evaluation:* Variable substitution is integral to any template language; PRESTO is no exception. Variables are defined by the data model. Templates gain access to variables by querying the provisioning database. The returned record defines a variable namespace, or *context*, used to access the variable, e.g.:

```
<CONTEXT.VARIABLE>
```

Variables of this form are directly substituted in the template text.

Templates are written to produce a configuration file for one router. PRESTO begins template evaluation by querying the `ROUTER` table in the data model for the row corresponding to the current router. The returned record populates the `ROUTER` context, which consequently allows templates to use `ROUTER` variables at any point. The template creates new contexts by making a new database query; however, those variables are only accessible within the defined context scope. When a query returns multiple records, the template code within the context is repeated, producing a loop. For example, SELECT * FROM LAN WHERE ROUTER=THIS_ROUTER has the effect of configuring multiple LAN-facing interfaces.

*3) Iteration:* The PRESTO template language simulates iteration by executing database queries that return multiple records. The template designer creates a "for-each" loop by defining a new context name, an SQL-like query, and a scope. Each row returned by the query produces an iteration. For example:

```
[INT:SELECT (*) FROM (WAN_INTERFACE) WHERE
 (WAN_INTERFACE.HOSTNAME=<ROUTER.HOSTNAME>)]
interface serial0/<INT.SLOT>/<INT.PORT>
 bandwidth <INT.BANDWIDTH>
 ip address <INT.IP> <INT.MASK>
!
[/INT]
```

Here, `INT` is the name of the new context. The statement associates the `INT` context with the record returned by querying the `WAN_INTERFACE` table of the provisioning database for all fields (`(*)`) related to the current router hostname (note the use of `<ROUTER.HOSTNAME>` in the query). The text within the `INT` context scope, i.e., all text between the query statement and the context closing statement, `[/INT]`, is repeated for each returned record. Field names from each record are accessible as variables within the context, as shown. Note that new context definitions can be arbitrarily nested, but they cannot define scopes spanning multiple parent scopes. That is, the nested context's closing statement must occur before its parent closing statement. This constraint is consistent with loop structures in common programming languages.

*4) Conditional Logic:* Configuration statements are commonly dependent on router properties. For example, E1 (a standard widely used in Europe) line cards required slightly different interface specification than T1 (a standard widely used in the US) cards. The PRESTO template language supports the inclusion and omission of configuration options with conditional statements. All conditionals have a label, condition and scope, in general:

```
[COND LABEL CONDITION]
... template text
[/LABEL]
```

COND indicates a conditional statement; LABEL defines a label; and CONDITION contains relational operators that dictate if the template text between the condition statement and the closing statement, [/LABEL], is included. The template text can contain static strings, new contexts, or even more conditionals. The CONDITION itself supports arbitrary complexity of Boolean logic. Statements can be simple:

```
("<ROUTER.HAS_FEATURE_X>" eq "YES")
```

or more complex logic:

```
(("<ROUTER.HAS_FEATURE_X>" eq "YES") &&
(("<ROUTER.HAS_FEATURE_Y>" eq "YES") ||
  ("<ROUTER.FEATURE_Z>" ne "BASIC")))
```

*5) Data Transformation:* Configuration statements commonly require a transformation of an input variable. For example, an interface IP address may be specified as IP and mask, i.e., *x.x.x.x/y*, but the router configuration language requires the IP and mask coded separately, i.e., *x.x.x.x z.z.z.z*. In another case, the template designer may need to configure the network address corresponding to the input value. To accommodate such requirements, the PRESTO template language provides a mechanism for arbitrary extension.

A function added to the language interpreter module is referenced within a template as a context, variable, function, and argument:

```
<CONTEXT.NEW_VARIABLE:function(args)>
```

Upon execution, arguments are evaluated (if they are variables) and passed to the function. The function performs a specific manipulation and returns the result to a new variable in the specified context. The new variable's value is inserted into the template text, and it's value is retained for later use within the context.

To aid template design, the PRESTO template language contains a core set of application agnostic functions. Some functions provide generic computation abilities, e.g., `calc()` performs simple arithmetic, `sbsstr()` returns a substring specified by an offset and length, and `matchre()` provides regular expression substring matching. Other core utility functions perform useful conversions on common network values such as IP address. For example:

```
<INT.NETIP:computeOffsetMaskIP(<INT.IP>,0)>
```

computes the network address of an IP specified in *x.x.x.x/y* form. The function, however, can calculate any offset of the IP, a useful feature when network policy dictates devices on specific offsets, e.g., the gateway is commonly `.1`. Realizing a new PRESTO function involves including its code in the PRESTO language interpreter.

*6) Hidden Evaluation:* Configuration policy occasionally requires values resulting from complex computations. While additional domain specific functions provide ample mechanism, template designers are encouraged to keep domain knowledge within the templates themselves. The motivation is twofold. First, this reduces bloat of the core language. Second, as function definitions require programming, and most template designers do not possess the necessary skills, or are simply unwilling, to create new functions. Therefore, we have added only a small number of generic primitive operations to the core language in an application.

As described to this point, the template language is not conducive to performing complex computation within the templates themselves. All functions return text that is inserted into the end router configuration. Multi-step computations therefore become difficult, if not impossible. To overcome this issue, the language supports hidden evaluation:

```
[EVAL LABEL noprint]
... template statements
[/LABEL]
```

Statements within the LABEL scope produce no output.

Computation within EVAL blocks is not limited to simple multi-step functional transformations. In practice, we leveraged the hidden evaluation interface to provided a multitude of features. For example, database SELECT queries were used to lookup values in supplemental data tables. Values were assigned to higher level contexts, e.g., ROUTER, and used throughout the template. The EVAL blocks also proved useful to determine values that depended on multiple conditionals. The conditional logic was performed once, and the value was used many times thereafter.

*7) Template Assembly:* Managing one large template becomes unwieldy. Software engineering experience recommends modular code. Templates are no exception. Using many small templates, or `configlets` provides many beneficial side effects. It allows a template designer to concentrate on one feature at a time. For example, a configlet can be written for each network access type used for the WAN interface of a router. Later, depending on the router provisioning data, the correct configlet is chosen. By including configlets on demand, complicated conditional logic is avoided. Additionally, as configlets are only inserted where applicable, they can be written with certain assumptions in mind. This reduces complexity within the configlets themselves. Finally, as configlets can include other configlets, the template designer can exploit commonality between configlets.

PRESTO stores all configlets in a template library. Configlets can be included at any point. The language provides a special syntax for including configlets:

```
[INCLUDE FROM (FEATURE) WHERE
  (FEATURE.TYPE=SOME_TYPE)]
```

In our above example, the correct WAN interface configlet is included using the <ROUTER.ACCESS_TYPE> variable:

```
[INCLUDE FROM (WAN) WHERE
```

```
1  %% Find loopback0 and store the IP address in the ROUTER context for later use
2  [EVAL FIND_LOOPBACK0 noprint]
3  [LOOPBACK: SELECT (*) FROM (LOOPBACK) WHERE (LOOPBACK.ROUTER_ID=<ROUTER.ROUTER_ID>)]
4  [COND LOOPBACK0 ("<LOOPBACK.INTERFACE_NAME>" eq "loopback0")]
5  <ROUTER.LOOPBACK0:setValue(<LOOPBACK.IP_ADDRESS>)>
6  [/LOOPBACK0]
7  [/LOOPBACK]
8  [/FIND_LOOPBACK0]
9  %% Configure Back-to-Back Interface to Peer Router
10 [INCLUDE FROM (TEMPLATES) WHERE (B2B.TYPE=<ROUTER.B2B_TYPE>)]
11 %% Define the BGP configuration
12 router bgp <ROUTER.LOCAL_ASN>
13  network <ROUTER.LOOPBACK0> mask 255.255.255.255
14 [PEER:SELECT (*) FROM (ROUTER) WHERE (ROUTER.HOSTNAME=<ROUTER.PEER>)]
15  network <PEER.NETIP:computeOffsetMaskIP(<PEER.B2B_IP>,0)> mask 255.255.255.252
16  neighbor <PEER.B2B_IP> remote-as <ROUTER.LOCAL_ASN>
17  neighbor <PEER.B2B_IP> next-hop-self
18 [/PEER]
19 [WAN:SELECT (*) FROM (WAN) WHERE (WAN.HOSTNAME=<ROUTER.HOSTNAME>)]
20 %% If WAN port used as a full port configuration, include network and neighbor statements
21 [COND CONFIGURE_WAN ("<WAN.FULL_PORT_CONFIG>" eq "TRUE")]
22  network <WAN.NETIP:computeOffsetMaskIP(<WAN.IP>,0)> mask <WAN.MASK:computeMask(<WAN.IP>)>
23 %% The gateway is the second IP in the subnet (for this example)
24  neighbor <WAN.GW:computeOffsetMaskIP(<WAN.IP>,1)>
25  remote-as <WAN.REMOTE_ASN>
26 [/CONFIGURE_WAN]
27 [/WAN]
28  no auto-summary
29 !
```

Fig. 1. Example configlet defining the WAN routing protocol of a two-line two-router configuration

```
(WAN.ACCESS_TYPE=<ROUTER.ACCESS_TYPE>)]
```

*8) Example Configlet:* Once the data model and configlet organization are determined, writing the configlets themselves is straightforward. We now provide a quick example to show how each of the language primitives come together. Consider a network topology where the Internet edge has two access lines, each connected to one router, and the two routers establish load sharing of in and outbound traffic. One of the more complex configlets defines the WAN routing protocol, of which Fig. 1 shows an example.

The example begins by using an `EVAL` block (lines 2-8) to acquire the IP address on the first loopback interface. Storing this value at the beginning of the configlet makes the subsequent code cleaner and allows the value to be used at multiple places without executing multiple queries. Next, a configlet is included (line 10) to define the back to back connection between the two routers. Multiple connection types may be supported. Instead of using a large conditional statement to pick the right interface definition, the `INCLUDE` statement allows the relational database to perform the conditional logic and simplify the code the domain expert must specify.

The BGP block (lines 12-29) defines the WAN routing protocol configuration. First, the `PEER` context (lines 14-18) is used to queries for network addresses specific to the peer router. Next, the `WAN` context (lines 19-27) queries information specific to the WAN connection (the data model specifies that WAN interface specifics be placed in a separate table). The configlet selects the correct table row using the `HOSTNAME` foreign key. In this context, the Cisco IOS `network` and `neighbor` commands require available information to be translated using the `computeOffsetMaskIP()` function. In this case, the remote peer is always the second IP in the network, hence the domain expert can use the offset function to code the neighbor's IP directly.

The template language and data model provides a scalable mechanism to describe configuration policy in native configuration language; however, alone it does not meet the requirement of supporting heterogeneous and diverse data sources (see Section II-B). We next present a two-step architecture to overcome these difficulties.

## *B. PRESTO Architecture*

In an ideal world, an engineer receives a request for a group of related routers with all required input information available and correct. This would allow a straightforward execution of the active templates upon inputs. Unfortunately, the information required to configure a router is not always readily available. In large operational networks, the input data for the configuration task spans the outputs of multiple upstream workflows, which may arrive at different points in time. It is therefore important to be able to work with such partial information flows and to able to handle any inconsistencies across the flows. In such a scenario, ubiquitous flow-through or full automation becomes extremely difficult to realize.

PRESTO achieves nearly full automation using a 2-step process, see Fig. 2. The goal of automation is to minimize user efforts. PRESTO minimizes manual processes in two ways. First, it allows bulk requests, where applicable, to streamline the process of creating the initial router configuration code. Second, it requires only one point of user interaction at which point users provide the most minimal effort, e.g., to correct inaccurate or enter missing information, to allow automation to complete. In PRESTO, data processing proceeds in two steps, with user integration capabilities made available during Step 1, the results of which are passed to Step 2.

Specifically, PRESTO uses a 2-step architecture to request user input at the most ideal moment. The process begins with the batch submission of one or more router configuration requests to Step 1. Step 1 pulls together and parses information from available input data sources, for example a customer order database. The role of Step 1 is to normalize and tabulate the input information and, if possible, apply

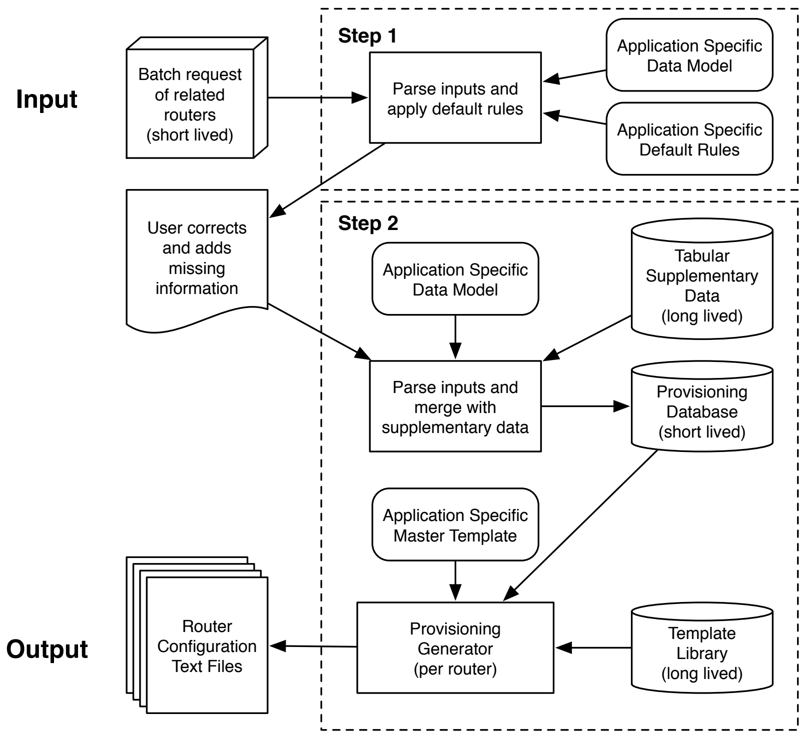
Input
Output
Step 1
Step 2
Batch request of related routers (short lived)
Parse inputs and apply default rules
Application Specific Data Model
Application Specific Default Rules
User corrects and adds missing information
Application Specific Data Model
Tabular Supplementary Data (long lived)
Parse inputs and merge with supplementary data
Provisioning Database (short lived)
Application Specific Master Template
Router Configuration Text Files
Provisioning Generator (per router)
Template Library (long lived)

Fig. 2. 2-step PRESTO Data Flow

defaults or inference rules to fill in missing information. Where defaults or inference are applied, the Step 1 output will flag or annotate the output, for (optional or mandatory) user inspection and possible refinement before resubmission to Step 1 to ensure correctness. Note that the design of PRESTO is intended to be input agnostic and can be adapted to a number of different modalities (via web services, via database invocations, or via stand-alone interfaces), and to operate on inputs of various forms and origins (database extracts, spreadsheets, XML forms). Whatever the implementation, the goal of Step 1 remains the same: output the complete and unambiguous information needed to configure all routers in the batch.

Step 2 parses the cleansed inputs from Step 1 and places the restructured data into a one-time database called the *provisioning database*. Then, for each router in the request, a *master template* is executed by the provisioning generator. This execution combines the short-lived provisioning database with up to two longer-lived databases: a template database and an optional supplemental database. The template database stores all pre-defined configlets for a particular service and is queried by the master template to include appropriate configlets, resulting in a configuration snippet or a completed "ready-to-load" configuration file. The creation of supplemental database[2] is service-dependent, and the database commonly includes persistent data already naturally expressed in tabular form, e.g., mapping from card names to interface type and number of ports. Note the separation between the short-lived provisioning data and the long-lived service rules database. Step 2 is stateless with respect to provisioning requests; it is simply a repository of configuration policy. This model allows PRESTO maintainers to develop, scrutinize, and rapidly deploy new service functionality to meet the demands of ever changing network technology, instead of dedicating resources to duplicate information stored elsewhere, e.g., in customer order databases.

[2]Supplemental database is not used in the current services deployed with PRESTO (see §IV), since decisions were made to include supplemental data in the inputs from Step 1.

By dividing the processing in these two steps, we isolate problems arising from inadequate information to Step 1, and provide the system and its users opportunities to repair fallout and resume automation before proceeding to Step 2, which then produces the desired output. However, sometimes processes require configurations before the complete set of inputs is available, e.g., a customer site with two router resiliency may deploy the routers at different times. In such cases, Step 1 ensures all inputs mandatory for a minimal router configuration are available, and Step 2 conditionally configures services depending on input completeness. Experience has demonstrated that performing such due diligence is not always a straightforward variable comparison. For example, a service may only be configured if a certain number of inputs are present. To accommodate such cases, we extended the PRESTO language enable checking of minimal requirements as well as verbose warning and error messages to inform the user of potential areas of concern.

## IV. Experience

The PRESTO system has benefited from the insights of network designers and engineers responsible for configuring network elements for large commercial connectivity services. A key measure of the value of such a tool, ultimately, is how useful and usable it is in practice for this target user

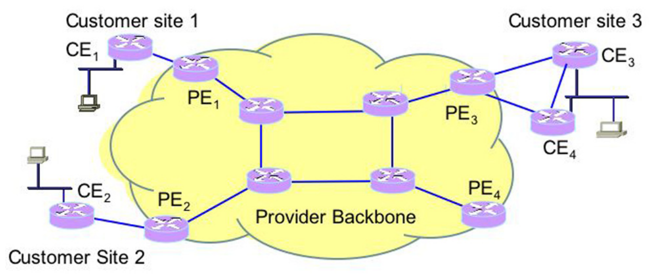

Fig. 3. Provider VPN

community. The PRESTO system is currently being used to automate configuration generation for a number of different commercial network services. In this section we first present a general design flow to adapt PRESTO for a new service. Second, we describe two unique provider-based services, VPN and VoIP, that are currently supported by PRESTO, and their PRESTO data models. Finally, we reflect on our continued experiences automating configuration for these services.

## A. Design a PRESTO tool for a New Service

Developing a PRESTO-based configuration tool for a new service involves knowledge engineering (codifying expert knowledge, initially only partially documented) and data modeling (identifying service-specific information and business rules), as well as front end user interface and back end PRESTO template development. The main tasks involved are:
(i) identifying all the service-specific information required for building the configs, and developing the resulting service-specific configuration data models (i.e., a provisioning data model and an optional supplemental data model).
(ii) understanding the workflow surrounding the provisioning process and available data sources and determining how the information in the above data model can be extracted.
(iii) collating the service-specific provisioning rules and building the service-specific templates based on engineering guidelines from the service designers and the associated data model for the template database to store the configlets.
(iv) defining the workflow of the PRESTO-based tool and developing service-specific code around the core service-agnostic PRESTO system.

## B. PRESTO for Virtual Private Networks

*1) Customer Edge Router Configuration for Provider VPNs:* Enterprise networks are increasingly using provider-based Virtual Private Networks (VPNs) to connect geographically disparate sites. In such a VPN, each customer site has one or more customer edge (CE) routers connecting to one or more provider edge (PE) routers in the provider network (see Fig. 3). Incoming customer traffic from a CE is encapsulated at the PE and carried across the provider network, decapsulated by a remote provider edge router and handed off to the customer CE router at a remote site of the same customer. Traffic belonging to different customers is segregated in the provider backbone, and the provider network is opaque to the customer. The predominant method for supporting provider-based VPNs uses MPLS [7] as the encapsulating technology across the provider backbone.

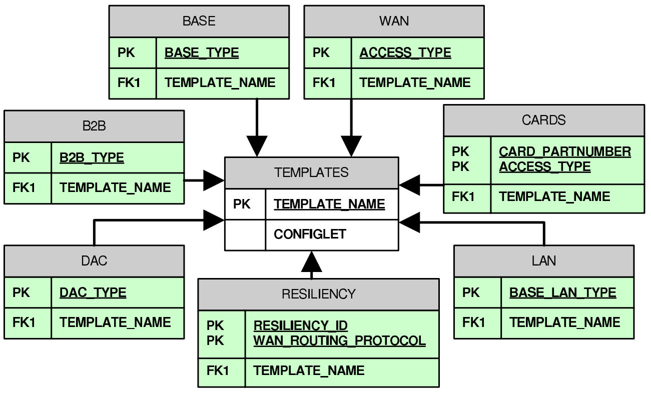

Fig. 4. Template data model for VPN service.

A critical part of supporting the VPN service involves configuring the CE routers. The tasks include configuring ACLs (Access Control Lists), interfaces, WAN (Wide Area Network) links and routing (e.g., OSPF and BGP). The key challenges pertain to heterogeneity and scale. VPN services enjoy a large and growing number of customers. A single customer can have hundreds to thousands of different sites. There are a wide range of features and options a customer can select that impact the configuration. There are different hardware elements (e.g., router models and interface cards), different access type options for the CE-PE connection (e.g., Frame Relay and ATM), different interface speed options, and so on. In addition to hardware options, customers choose from a range of resiliency options varying the number of routers, and access lines. This huge space of feature combinations and the need to support a steadily increasing slew of new features makes CE router configuration a prime candidate for PRESTO automation.

*2) PRESTO Data Models for CE Configuration:* Recall that PRESTO requires an application to define two mandatory data models—a provisioning data model for short-lived data, and a template data model for pre-defined configlets. The provisioning data model provides a repository of data specific to the current router request set. The VPN instantiation of the provisioning data model placed as many fields as possible in a central ROUTER table indexed by an unique ID assigned to each router to be provisioned. This contained router-specific information such as hostname, model number, software version, and available customer information. Whenever multiple instances of any type of data was required to build template iterations, a new table was created, as replication has risk in provisioning tasks. For CE configuration, this led to the creation of tables for WAN interfaces, dial backup information, and the logical interfaces that define VPN connections. In total, the resulting provisioning data model consisted of one main ROUTER table and 12 secondary tables, each containing foreign keys to reference the ROUTER table.

The data model for the longer-lived template database contained the configlets used to create the actual router configuration. As shown in Fig. 4, the data model consisted of 7 feature tables, all referencing the TEMPLATES table in the center. The master template operates on this template database, collates the appropriate configlets, and produces a completed router configuration file. To get a configlet, the master template uses relevant information from the provisioning database to derive the name of the target template from the feature table. The name is then used to key into the TEMPLATES table to retrieve the corresponding configlet. This extra layer of indirection enables records from multiple feature table to share a single configlet. As a result, if a configlet is changed, only the TEMPLATES table needs to be repopulated.

Configlets were grouped into feature tables by logical features, specifically BASE, LAN, WAN, CARDS, RESILIENCY, DAC, and B2B. The BASE table consisted of the configlet required for all routers, e.g., hostname, password, loopback, and motd commands. The LAN and WAN tables contained configlets for various types of interfaces, e.g., frame relay and ATM interfaces. The CARDS table contained configlets for interfaces that are line card-specific. The RESILIENCY table contained configlets defining the different resiliency options required by the VPN service. The DAC table contained configlets specific to various parts of the dial backup configuration. The B2B table defined special interface definitions used where CE routers are organized in back to back configurations. Defining these feature tables as primitives or building blocks allowed specialized configlets to be easily composed and promoted knowledge and code reuse. A total of 86 configlets containing 4569 lines of statements were created.

### C. PRESTO for Voice-over-IP Networks

*1) VoIP Router Configuration for Provider VoIP Networks:* Another key service that enterprise networks leverage is provider-based Voice-over-IP (VoIP) networks to facilitate voice communication between disparate sites. Similar to provider-based VPNs, the VoIP service for a customer has multiple VoIP sites connected via the IP backbone. Each customer VoIP site uses one or more VoIP Routers (VRs) to provide connectivity between the customer VoIP network and the provider backbone (via PE routers). The VoIP network consists of a heterogeneous set of VoIP Network Equipments (NEs) (e.g., firewall devices, VoIP servers, etc) connected to the VRs via Virtual Local Area Networks (VLANs), and has grown at a rapid pace to meet customer demand.

A key challenge of supporting the VoIP service involves configuring the VRs to accommodate a continuous growing set of NEs in the VoIP network. For each NE, the configuration tasks include configuring VLANs, virtual and physical interfaces, VRF (virtual routing and forwarding), WAN (Wide Area Network) and LAN (Local Area Network) links, and routing (e.g., BGP and OSPF). There are a wide range of NEs a customer can use and require support for, and configuration details of some tasks are different for NEs from different vendors. In order to efficiently create and maintain a disparate VoIP site with up-to-date NEs, automation of VR configurations is necessary for greenfield deployment of a new VoIP site, as well as incremental configurations for later additions and removals of NEs. These characteristics make VRs well-suited for PRESTO automation.

From the provider's perspective, there are two key differences between configuring VRs in the VoIP service and configuring CE routers in the VPN service. First, while CE router configurations are themselve heterogeneous due to the various hardware elements and resiliency options available, the VRs have homogeneous hardware components. The heterogeneity in VR configurations stems from the myriad of VoIP NEs that can be connected to and need to be supported by a VR. Second, in addition to handling complex greenfield configurations as in the case of CE routers, the need for effectively and correctly modifying parts of the configuration to reflect the additions and removals of NEs makes VRs a unique candidate for brownfield automation by PRESTO.

*2) PRESTO Data Models for VR Configuration:* While the designs of the provisioning and template databases for VR configuration are very similar to those for CE configuration, here we describe the data models for VR configuration and highlight their key differences.

In order to offer support for incremental configuration of VRs to handle NE additions and removals, a VR configuration was divided into smaller, self-contained *configuration snippets* which are to be executed on top of a *base configuration*. The base configuration is a standalone configuration file the router can immediately load as its running configuration. In contrast, configuration snippets are to be applied via the router's command line interface or via replacing parts of the router's running configuration. For example, different configuration snippets are created for the VR to communicate with different NEs. This modular design is specific to the service requirement of VoIP networks and is significantly different from the PRESTO output for VPNs, where a completed monolithic "ready-to-load" configuration file is always generated.

The provisioning request for a VR consists of a list of *configuration tasks* of various types. Each task contains information about the type of task requested and the necessary parameters required to complete the task. PRESTO processes this list in sequence, generates the configlet (a base configuration or configuration snippet) for each task, and finally pushes back a list of completed configlet to the enabler.

The provisioning database stored a provisioning request by having each router in the ROUTER table to be associated with a list of configuration tasks. Whenever PRESTO encounters a new type of task, a new table was created to store the information associated with the task. For example, in a list of 3 tasks, if the first and the second tasks involve creation of VLANs, and the third task requires creation of a LAN interface, a total of 2 tables are created to store the provisioning request. For VR configuration, this led to the creation of tables for VLAN interfaces, LAN interfaces, BGP route advertisements, etc. In total, the resulting provisioning data model consisted of one main ROUTER table, and 13 secondary tables, each corresponding to a supported type of configuration task and containing foreign keys to reference the ROUTER table.

The template data model for VR routers had a similar 2-level structure to the template data model for CE routers. As shown in Fig. 5, configlets were grouped into 4 task tables, all referencing the TEMPLATES table in the center. A total of 13 configuration tasks were supported. For each configuration task associated with a router, the master template queries the appropriate task table to obtain the name of the target template, uses the name to retrieve the corresponding configlet from the TEMPLATES table, and finally generates a completed configlet for the task. If the configlet corresponding

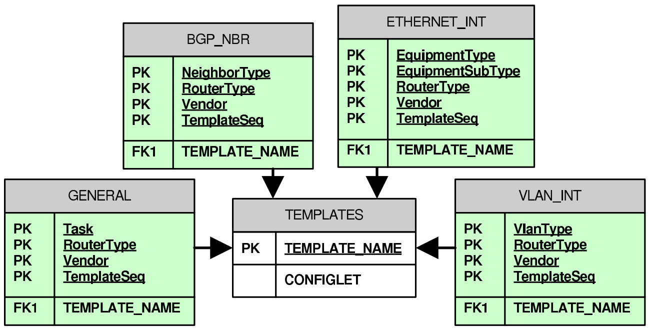

Fig. 5. Template data model for VoIP service.

to a task was indepedent of the NEs, it was stored in the `GENERAL` table; this included configlets for 10 tasks such as configuring the base commands (e.g., hostname, loopback, banner commands), adding static routes, and defining VLANs for NEs. The `BGP_NBR` table contained configlets for creating BGP sessions between VRs and PEs or between VRs themselves. The `ETHERNET_INT` table and `VLAN_INT` table contained NE-dependent configlets for configuring Ethernet and VLAN interfaces facing various NEs. A total of 64 configlets containing 2634 lines of statements were created.

## *D. Controlled Brownfield Support with PRESTO*

Thus far, we have focused on using PRESTO to generate greenfield configuration. Recently we have adapted PRESTO to provide support for brownfield configurations. In this section, we discuss how PRESTO was adapted for brownfield configuration in the context of configuring VoIP routers (VRs) due to their unique requirement for such a feature.

Recall that the design objective of configuration snippets was to allow incremental VR configurations to handle additions and removals of NEs. This involves making changes to VR configurations already in operation (i.e., brownfield configurations) when NEs are later connected to or disconnected from VRs. While highly important, supporting changes to brownfield configurations in a generic way is hard and has remained an open research problem. In particular, a tool needs to automatically infer the existing state of a brownfield configuration in order to generate configuration changes correctly and ensure no negative impacts to other parts of the configuration.

Instead, we offered limited support for changes to brownfield configurations in a controlled setting, with the key assumption that any prior changes, if any, was made only via PRESTO. By making this assumption, the range of allowed change configuration is tightly controlled by PRESTO, thereby eliminating the need for inferring configuration states. To achieve this, for each configuration snippet requested to reflect NE additions, PRESTO also produces the associated *backout* snippet to nullify the corresponding configuration snippet. Then, the backout snippet can be archived and executed later when the NE is removed. Fig. 6 shows an example configuration-backout snippet pair for configuring BGP's advertised routes. In total, 11 out of the 13 supported tasks have their backout snippets defined. The remaining 2 tasks pertain to core configuration and interface initialization, and they are

```
! ConfigletType = Config, Task = AdvertisedRoute
router bgp <TASK.ASN>
 network <TASK.IPADDRESS> mask <TASK.MASK> \
     route-map <TASK.ROUTEMAP>
!
! ConfigletType = Backout, Task = AdvertisedRoute
router bgp <TASK.ASN>
 no network <TASK.IPADDRESS> mask <TASK.MASK> \
     route-map <TASK.ROUTEMAP>
!
```

Fig. 6. Example configuration and backout snippets.

one-time configurations whose effects need not be rolled back later.

## *E. Deployment Status and Lessons*

We have implemented PRESTO systems for the VPN and VoIP services, and they are currently under deployment for the large ISP. Since deployment the PRESTO tool has been used for configuring a majority of the routers in the VoIP service.

In developing a PRESTO-based configuration tool, the results of defining configuration logic on top of native device languages were immediately apparent. Expressing such logic in the *lingua franca* of the configuration experts was simply a requirement of fully capturing their desires. On the other hand, the effects of heterogeneous hardware, impure data sources, and guideline ambiguity were not fully realized until the tool had matured. The remainder of this section describes how features within the PRESTO framework mitigate these challenges, using provider-based VPN configuration as the focus of our discussion.

*1) Configlet Modularity:* Connection resiliency is the primary configuration option for the customer edge of the provider-based VPN configuration. Depending on site requirements, the customer chooses from a vast array of router models and connection technologies (e.g., Frame Relay, ATM) as well as the number of routers and the number of access lines per router. In developing the configlets for resiliency, we found a basic tenet of software engineering to hold true, specifically: *develop in modular fashion*. However, the extent of modularity depends highly on the service rules required for an application.

For example, modularity was fundamental to the routing configuration (BGP). The BGP configuration describes the topology of the network, e.g., the `network` and `neighbor` statements in Cisco IOS. Such statements are needed for each router interface involved in BGP, but are the same for both physical and logical interfaces. Hence, we created one configlet containing `network` and `neighbor` statements and dynamically included it from other configlets defining different physical and logical interfaces.

Overall, configlet modularity played a significant role as the template library grew to support more options. In our initial designs, "parallel" configlets, i.e., configlets defining the same service, frequently contained duplicate code. Changing this code required burdensome duplication of changes and can be dangerous if one configlet is missed. Increasing the modularity of configlets decreased code duplication, which eased maintenance and increased reliability. Additionally, breaking up configlets provided significant code reuse, resulting in faster development of new features and hardware support.

*2) Reliable Data Sources:* Real deployment scenarios are often plagued with incomplete and inconsistent configuration requirements. PRESTO overcomes this challenge using the 2-step architecture; users refine and correct data inputs before generating the end router configuration. Recall that PRESTO's primary goal is to reduce manual intervention, therefore we desire as complete and correct a data source as possible. While one may attempt to automatically reconcile multiple data sources, doing so is non-trivial and may introduce user confusion or result in invalid configuration requests.

Through our experience with the CE configuration tool, we discovered a new data source requirement that exposed the necessity for PRESTO to mesh with existing systems and processes. Initially, we used a static input source that provided most, but not all information necessary to configure multiple routers related to a customer order. Step 1 parsed the input, producing a spreadsheet for completion by the enabler. While this model satisfied PRESTO's goal of not maintaining a new customer information database, existing configuration processes caused it to *unfairly push the data management problem onto the enabler*. For this application, enablers commonly only configured one router at a time due to process dependencies that impeded the availability of information required to create configurations. Hence the spreadsheets had to be maintained over significant durations. Based on our learning from that experience, the current version of the CE configuration tool interfaces directly with an existing customer requirements database and pushes back updates made by enablers. This experiences shows that data sources allowing enabler effort to be propagated back not only keep existing databases current, but are paramount to providing a usable interface.

*3) Resolving Guideline Ambiguity:* Natural language configuration guidelines are inherently ambiguous. Edge cases often lead to ad-hoc creation of local provisioning practices. As enablers resolve interpretation difficulties, they relay resolution techniques to coworkers; however, often solutions are not relayed back to a central location for validation and incorporation into the guidelines. Such inconsistencies cause obscure configuration errors that go long periods before detection. Requiring new configurations to use a central unambiguous guideline repository overcomes regional differences, and more importantly, it allows edge cases to be *fixed once and used forever*.

The development of the template library for the CE configuration tool revealed many ambiguities in the natural language guidelines. The rule discovery process exposed various inconsistencies between regional interpretation. After much debate, inconsistencies were resolved, and the guidelines were amended. Hence, the rule discovery process itself provided great value to the CE configuration process. The existential benefits continued as additional edge cases were discovered, and validated solutions were incorporated into the template library and used for all future configurations. In doing so, PRESTO closes the loop between engineers designing guidelines and enablers implementing them.

## V. Related Work

Several industrial products (for example, [8], [9], [10], [11], [12]) have emerged that offer support for configuration management. Many of these efforts have focused on developing abstract languages to specify configurations in a vendor neutral fashion, e.g., IETF standard SNMP5D MIBs [3], the Common Information Model (CIM) [4], and the Common Management Information Protocol (CMIP) [13], [5], [14]. These information models define and organize configuration semantics for networking/computing equipment and services in a manner not bound to a particular manufacturer or implementation. An example of the success of such an approach is the DSL Forum's TR-069 effort for DSL router configuration [15]. Yet, general router configuration via this approach is challenging given rapid technology evolution, driving network operators and vendors towards competitive and differentiated advantage, feature proliferation, and the need to continuously expand networks and features while maintaining backwards compatibility.

Boehm et. al. [16] present a system that raises the abstraction level at which routing policies are specified from individual BGP statements to a network-wide routing policy. The system includes support to generate the appropriate pieces of router configuration for all routers in the network. An approach to automated provisioning of BGP-speaking customers is discussed in [17]. These efforts focus on BGP, just one component of router configuration. Narain [18] seeks to bridge the gap between end-to-end network service requirements, and detailed component configurations, by formal specification of network service requirements. Such specification could aid synthesis of router configurations. In contrast to these efforts, our focus in PRESTO is on the synthesis of complete, precise, and diverse network configurations that are readily deployable.

Several initiatives have explored configuration management systems for desktop, and server environments [19], [20], [21], [22]. Networked and router environments often involve more complex options and inter-dependencies than desktop environments, and these solutions do not directly apply. That said, there is much potential benefit from cross-fertilization between these domains. Further, many of these works emphasized deployment of configurations and placed relatively little effort on deciding what the configuration of a node should be [21].

While the focus of PRESTO is the synthesis of configuration files, others have looked at important orthogonal issues related to configuration management. The Network Configuration Protocol (NETCONF) [23], [24] effort provides mechanisms to install, manipulate, and delete the configuration of network devices. The NESTOR project [25] seeks to simplify configuration management tasks which requires changes in multiple interdependent elements at different network layers by avoiding inconsistent configuration states among elements, and facilitating undo of configuration changes to recover an operational state. Others [26], [27] have looked at detailed modeling and detection of errors in deployed configurations.

Others have proposed a completely new management and control plane architecture [28]. Work by Greenberg et. al. lead to implementations and extensions of the "4D" architecture [29], [30], [31], [32]. While these systems reduce the complexity of network control and management, they require a complete "ground up" restructuring of the network elements which is very costly. Meanwhile, PRESTO works within the confines of existing router architectures.

## VI. Conclusions

The PRESTO system presented throughout represents a step toward realistic automation of massive scale configuration management. Central to the success of PRESTO are the satisfied mandates for the treatment of complex and evolving service definitions and customer requirements, dealing with the hugely diverse and sometimes unreliable data sources, and communication within the *lingua franca* of its user community.

PRESTO attempts to balance these requirements by providing malleable and composable *configlets* that encode configuration business logic directly in the target language. Our experiences developing a PRESTO-based configuration management tool for a VPN service clearly demonstrated the utility of the framework. In short, PRESTO promotes a healthy configuration management environment by actively cleansing existing customer information databases, identifying unaccounted for edge cases, and eliminated redundant fixes of such problems.

Note that PRESTO, in its current stage, accounts for greenfield and controlled brownfield configuration management. While highly important, a complete configuration management system must support generic change of live systems. Support for generic brownfield scenarios requires great care to avoid negative consequences, e.g., performance, security, and connection problems, and are the subject of ongoing work.

## Acknowledgments

We would like to thank Charles Kalmanek and Jennifer Rexford for their invaluable inputs during the initial design phases of PRESTO, the AT&T Labs engineering team for their support, and Brian Freeman for his many helpful suggestions on improving the paper.

## References

[1] W. Enck, P. McDaniel, S. Sen, P. Sebos, S. Spoerel, A. Greenberg, S. Rao, and W. Aiello, "Configuration management at massive scale: System design and experience," in *Proc. USENIX Annual Technical Conference*, Jun. 2007.
[2] Cisco Systems, Inc., *Cisco IOS Configuration Fundamentals Command Reference*, 2006, release 12.4.
[3] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A simple network management protocol (snmp)," http://www.ietf.org/rfc/rfc1157.txt, May 1990.
[4] Distributed Management Task Force, Inc., http://www.dmtf.org.
[5] "ISO 9596: Common management information protocol," ISO, 1998, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=29698.
[6] T. Li, B. Cole, P. Morton, and D. Li, "RFC 2281, Cisco hot standby router protocol (HSRP)," *Internet Engineering Task Force*, Mar. 1998, http://www.ietf.org/rfc/rfc2281.txt.
[7] E. Rosen, A. Viswanathan, and R. Callon, "RFC 3031, Multiprotocol Label Switching Architecture," *Internet Engineering Task Force*, Jan. 2001, http://www.ietf.org/rfc/rfc3031.txt.
[8] "Cisco IP solution center," http://www.cisco.com/en/US/products/sw/netmgtsw/ps4748/index.html.
[9] "Intelliden," http://www.intelliden.com/.
[10] "Opsware," http://www.opsware.com/.
[11] "Voyence," http://www.voyence.com/.
[12] Cisco Systems Inc., "Cisco works small network management solution version 1.5," http://www.cisco.com/warp/public/cc/pd/wr2k/prodlit/snms_ov.pdf, 2003.
[13] U. Warrier, L. Besaw, L. LaBarre, and B. Handspicker, "RFC 1189, common management information services and protocols for the internet (CMOT/CMIP)," *Internet Engineering Task Force*, Oct. 1990, http://www.ietf.org/rfc/rfc1189.txt.
[14] "ISO 9595: Common management information service," ISO, 1998, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=30585.
[15] "DSL forum TR-069," http://www.dslforum.org/aboutdsl/tr_table.html.
[16] H. Boehm, A. Feldmann, O. Maennel, C. Reiser, and R. Volk, "Network-wide inter-domain routing policies: Design and realization," Apr. 2005.
[17] J. Gottlieb, A. Greenberg, J. Rexford, and J. Wang, "Automated provisioning of BGP customers," in *IEEE Network Mag.*, Dec. 2003.
[18] S. Narain, "Network configuration management via model finding," in *Proc. Large Installations Systems Administration (LISA) Conference*, Dec. 2005.
[19] P. Anderson, "Towards a high-level machine configuration system," in *Proc. 8th Large Installations Systems Administration (LISA) Conference*, 1994.
[20] M. Burgess, "Cfengine: a site configuration engine," in *USENIX Computing systems, Vol 8, No. 3*, 1995.
[21] P. Anderson and E. Smith, "Configuration tools: Working together," in *Proc. Large Installations Systems Administration (LISA) Conference*, Dec. 2005.
[22] N. D. et al, "A case study in configuration management tool deployment," in *Proc. Large Installations Systems Administration (LISA) Conference*, Dec. 2005.
[23] "Network configuration (netconf)," http://www.ietf.org/html.charters/netconf-charter.html.
[24] R. Enns, "NETCONF configuration protocol," http://www.ietf.org/internet-drafts/draft-ietf-netconf-prot-12.txt, Feb. 2006.
[25] Y. Yemini, A. Konstantinou, and D. Florissi, "NESTOR: An architecture for network self-management and organization," *IEEE J. Select. Areas Commun.* , vol. 18, no. 5, pp. 758–766, May 2000.
[26] N. Feamster and H. Balakrishnan, "Detecting BGP configuration faults with static analysis," in *Proc. 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, May 2005.
[27] A. Feldmann and J. Rexford, "IP network configuration for intradomain traffic engineering," in *IEEE Network Mag.*, Sep. 2001.
[28] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4d approach to network control and management," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, pp. 41–54, 2005.
[29] H. Ballani and P. Francis, "Complexity oblivious network management," Cornell University, Tech. Rep., 2006. [Online]. Available: http://techreports.library.cornell.edu:8081/Dienst/UI/1.0/Display/cul.cis/TR2006-2026
[30] ——, "Conman: taking the complexity out of network management," in *INM '06: Proc. 2006 SIGCOMM workshop on Internet network management*. New York, NY, USA: ACM, 2006, pp. 41–46.
[31] ——, "CONMan: A Step towards Network Manageability," in *Proc. of ACM SIGCOMM*, 2007.
[32] T. S. E. Ng and H. Yan, "Towards a framework for network control composition," in *INM '06: Proc. 2006 SIGCOMM workshop on Internet network management*. New York, NY, USA: ACM, 2006, pp. 47–51.

**William Enck** is a Ph.D. candidate researching network and systems security in the Systems and Internet Infrastructure Security Laboratory in the Computer Science and Engineering Department at Penn State University. William received a B.S. (with highest distinction and honors) and M.S. in Computer Science and Engineering from Penn State in 2004 and 2006, respectively. His research efforts have included telecommunications security, specifically modeling and characterizing SMS vulnerabilities, systems and hardware security, and large-scale network configuration.

**Thomas Moyer** is a Ph.D. candidate researching network and systems security in the Systems and Internet Infrastructure Laboratory in the Computer Science and Engineering Department at Penn State University. Thomas received a B.S. in Computer Engineering from Penn State in 2006. His research interests include systems security, distributed system security, and large-scale network configuration.

**Patrick McDaniel** is an Associate Professor in the Computer Science and Engineering Department at the Pennsylvania State University and co-director of the Systems and Internet Infrastructure Security Laboratory. Patrick's research efforts centrally focus on network, telecommunications, and systems security, language-based security, and technical and public policy issues in digital media. Patrick was awarded the National Science Foundation CAREER Award and has chaired several top conferences in security including, among others, the 2007 and 2008 IEEE Symposium on Security and Privacy and the 2005 USENIX Security Symposium. Patrick is the editor-in-chief of the ACM Journal Transactions on Internet Technology (TOIT), and serves as associate editor of the journals ACM Transactions on Information and System Security and IEEE Transactions on Software Engineering. Prior to pursuing his Ph.D. in 1996 at the University of Michigan, Patrick was a software architect and program manager in the telecommunications industry.

**Subhabrata Sen** (M'01) received a Bachelor of Engineering (First Class with Honors) degree in Computer Science (1992) from Jadavpur University, India, and M.S. and Ph.D. degrees in Computer Science from the University of Massachusetts, Amherst, USA, in 1997 and 2001, respectively.

Dr. Sen is currently a Principal Member of Technical Staff in the Internet & Network Systems Research Laboratory at AT&T Labs—Research, where he has been since 2001. His research interests include IP network management, traffic analysis, network data mining, security and anomaly detection, peer-peer systems, and end-to-end support for streaming multimedia. He is a member of the ACM.

**Panagiotis Sebos** is a freelancer working on various diverse networking projects. His research interests include network design problems, network management, IP/Optical integrated networks, real time operating systems and user interfaces.

**Sylke Spoerel** received a Dipl.-Ing.(BA) in technical computer science from the technical academy Stuttgart, Germany in 1989. She completed the CCIE in 2000.

S.Spoerel is currently a Principal Member of Technical Staff in the Edge-based VPN Service development at AT&T LABs, where she has been since 2001. Her development interests include IP network management, virtual private networks and controlled interworking of services.

**Albert Greenberg** is an ACM Fellow, and a Principal Researcher at Microsoft, which he joined in Jan 2007. At Microsoft, he is working on data center networking, enterprise network management, and monitoring. From 1983 to 2007, Albert worked at AT&T Bell Labs, where he was named an AT&T Fellow and was awarded AT&T's Science and Technology Medal., and where he worked on packet and flow measurement and analysis, traffic matrix inference, anomaly detection, configuration management, IP/MPLS control plane monitoring, MPLS/GMPLS control and management, IP traffic and network engineering, IP fault management and troubleshooting, new route control architectures, database and systems applications, network security, scheduling, wireless and satellite networks, massively parallel computation, and parallel simulation.

**Yu-Wei Sung** was born in Taipei, Taiwan, in 1980. He received the B.A.Sc. degree in computer engineering from the University of Toronto, Toronto, Ontario, in 2004, and the M.S. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, in 2006. He currently pursues his Ph.D. degree at Purdue University.

From May 2002 to August 2003, he was an intern at the IBM Toronto Lab, where he worked on database analysis and J2EE demo development. Since summer 2006, he has worked on a system for automated network element configuration provisioning with AT&T Research, Florham Park, NJ. His research interests are in computer networks and distributed systems, with a focus on overlay networks, peer-to-peer systems, and enterprise network management.

Mr. Sung is a recipient of the University of Toronto Scholar Award and W.S. Wilson Medal from the University of Toronto, and Estus H. and Vashti L. Magoon Outstanding Teaching Assistant Award from Purdue University.

**Sanjay G. Rao** received the Bachelor's degree in Computer Science and Engineering from the Indian Institute of Technology, Madras in 1997 and the Ph.D from the School of Computer Science, Carnegie Mellon University in 2004.

He is an Assistant Professor in the School of Electrical and Computer Engineering, Purdue University, West Lafayette, where he leads the Internet Systems Laboratory. He was a visiting researcher in the Network Measurement and Management group at AT&T Research, Florham Park, New Jersey in Summer 2006. He played a leadership role in the End System Multicast project which pioneered live streaming using peer-to-peer systems, now a mainstream research area and an emerging commercial sector. His research interests are in Peer-to-Peer systems, and Network Management.

Prof. Rao has served on the Technical Program Committees of several workshops and conferences including ACM SIGCOMM, IEEE Infocom, and ACM CoNEXT.

**William Aiello** is a Professor at the University of British Columbia which he joined as Head of the Department of Computer Science in December of 2004. He received his B.S. in Physics from Stanford University and his Ph.D. in Applied Mathematics from M.I.T. in 1988. After a year post doctoral fellowship at M.I.T. he spent 9 years at Bellcore working in complexity theory, graph theory, parallel computing, routing theory and cryptography. In 1998 he joined AT&T Labs where he stayed for six years, the last five as Division Manager of Cryptography and Network Security Research. While at AT&T he designed several elements of DOCSIS and PacketCable, the cable industrys packet protocols and Voice over IP service, respectively; he was responsible for near- and long-term design and architecture issues for securing AT&T's VoIP service, and for long term research for securing IP backbone and enterprise networks. Current research interests include graph theory, network security, and cryptography.