ComplianceNet: Uma ferramenta de governança contínua no gerenciamento de redes baseadas em NETCONF

Modalidade: Código Fechado

João Pedro S. Pereira[1], **Diogo M. F. Mattos**[1],
Dianne S. V. Medeiros[1], **Nicollas R. de Oliveira**[1]

[1] LabGen/MídiaCom – TET/IC/PPGEET/UFF
Universidade Federal Fluminense (UFF)
Niterói, RJ – Brasil

Abstract. *This paper proposes ComplianceNet, a tool for continuous configuration governance in YANG-modeled networks. The tool continuously compares the configuration declared in a Git repository with the running configuration of network devices to detect configuration drifts. As its main technical contribution, ComplianceNet employs a comparison mechanism that aligns list elements using the keys defined in YANG models, eliminating false positives caused by the nondeterministic ordering of lists returned by network devices. The demonstration considers a realistic OSPF configuration drift scenario in which ComplianceNet correctly identifies the single genuine change across ten devices, achieving a 100% success rate in list reordering cases with an execution time of 0.0446 ms, whereas diff-based approaches report false positives. When a configuration drift is detected, the tool automatically creates a Merge Request, allowing the operator to decide whether to incorporate the detected change.*

Resumo. *Este artigo propõe o ComplianceNet, uma ferramenta para governança contínua de configurações em redes modeladas em YANG. A ferramenta compara continuamente o estado declarado em um repositório Git com a configuração em execução nos dispositivos de rede para detectar desvios (drifts) de configuração. Como principal diferencial técnico, o ComplianceNet emprega um mecanismo de comparação que alinha os elementos das listas pelas chaves declaradas nos modelos YANG, eliminando falsos positivos decorrentes da ordenação não determinística das listas retornadas pelos dispositivos. A demonstração considera um cenário realista de drift em configurações OSPF, no qual o ComplianceNet identifica corretamente a única alteração genuína entre dez dispositivos, obtendo 100% de acerto nos casos de reordenação com um tempo de execução de 0.0446 ms, enquanto abordagens baseadas em diff reportam falsos positivos. Quando um drift é identificado, a ferramenta cria automaticamente um Merge Request, permitindo ao operador decidir se incorpora a alteração.*

1. Introdução

O crescimento da complexidade das redes de computadores, impulsionado pela heterogeneidade de fabricantes e pela necessidade de operação contínua, torna cada vez mais difícil garantir que o estado real dos equipamentos corresponda ao estado formalmente aprovado pelos processos de mudança [Amlou et al., 2023]. Intervenções emergenciais, testes de campo e ajustes pontuais realizados diretamente via linha de comando

são práticas recorrentes na operação de redes. Essas intervenções podem introduzir divergências entre a configuração desejada, normalmente versionada em um repositório de controle de mudanças, e a configuração observada no equipamento. Tal divergência é conhecida como desvio (*drift*) de configuração.

Tradicionalmente, mecanismos de automação que não cobrem integralmente o *drift* de configuração comprometem a auditabilidade da rede e a conformidade com processos de governança de TI. Mudanças não rastreadas passam a existir fora de qualquer trilha de aprovação, dificultando a identificação de alterações realizadas diretamente nos equipamentos. Este problema agrava-se em redes multi-fabricante ou compostas por equipamentos com diferentes sintaxes de configuração nas quais comparações puramente textuais tornam-se pouco confiáveis. Nesse contexto, o protocolo NETCONF e os modelos YANG surgem como padrões abertos do IETF (*Internet Engineering Task Force*) para expressar a configuração de dispositivos de rede de forma estruturada e independente de fabricante [Wallin e Wikström, 2011]. Essa representação permite que ferramentas de automação operem sobre uma estrutura hierárquica e tipada da configuração, em vez de depender exclusivamente de CLI (*Command Line Interface*). Contudo, a adoção de uma representação estruturada não elimina todos os desafios da detecção de *drift*. Ferramentas amplamente utilizadas, como o *terraform plan*, realizam comparações posicionais entre atributos, desconsiderando as chaves definidas nos modelos YANG para identificar cada elemento. Como consequência, diferentes ordenações de uma mesma lista podem ser interpretadas como alterações de configuração, mesmo quando o conteúdo permanece inalterado. Esse comportamento produz falsos positivos de *drift* e dificulta a identificação de mudanças reais.

Este artigo propõe o ComplianceNet, uma ferramenta para governança contínua de configurações em redes baseadas em NETCONF/YANG. A ferramenta integra gerenciamento de mudanças, detecção contínua de *drift* e controle de versões em um fluxo único de operação. Seu principal diferencial é um mecanismo de comparação de configurações orientado pelas chaves declaradas nos modelos YANG, eliminando falsos positivos na detecção de *drift* decorrentes da reordenação de listas retornadas pelos dispositivos. Adicionalmente, no fluxo de detecção contínua de *drift*, a ferramenta cria automaticamente *Merge Requests* para que o operador possa decidir entre incorporar a alteração detectada ao estado desejado ou restaurar a configuração previamente aprovada. As principais contribuições deste trabalho são: i) a ferramenta ComplianceNet para governança contínua de configurações em redes baseadas em NETCONF/YANG; ii) uma estratégia para detecção de *drift* orientada pela semântica do modelo YANG; iii) a integração da detecção automática de *drift* ao processo de gerenciamento de mudanças baseado em *Merge Requests*.

O restante deste artigo está organizado da seguinte forma. A Seção 2 discute os trabalhos relacionados. A arquitetura e o funcionamento do ComplianceNet são apresentados na Seção 3. Os resultados são apresentados e discutidos na Seção 4, assim como a proposta de demonstração. Por fim, a Seção 5 conclui o artigo.

2. Trabalhos Relacionados

Trabalhos anteriores estabelecem as bases para a automação de configuração de rede baseada em NETCONF e YANG. Wallin e Wikström propõem e validam o uso de

NETCONF e YANG para simplificar a gestão de configuração em um ambiente com 2000 dispositivos, demonstrando bom desempenho frente a abordagens baseadas em *scripting* de CLI [Wallin e Wikström, 2011]. Bajpai *et al.* estendem o modelo de gerência para o contexto de roteadores domésticos, propondo o uso de NETCONF sobre TLS combinado ao mecanismo de *Call Home* para permitir que o gerenciador inicie a sessão de gerência com dispositivos localizados em redes protegidas por NAT [Bajpai et al., 2021]. Amlou *et al.* demonstram a programabilidade automatizada de redes utilizando modelos OpenConfig YANG em conjunto com NETCONF, avaliando tolerância a falhas em um cenário de automação [Amlou et al., 2023]. Esses trabalhos consolidam NETCONF e YANG como bases de automação, mas nenhum deles trata da detecção de *drift* entre o estado declarado e o estado observado ao longo do tempo.

Uma linha de pesquisa distinta concentra-se na verificação estática de arquivos de configuração para identificar erros antes de aplicar a configuração [Fogel et al., 2015]. Brown e Millstein propõem uma abordagem declarativa que deriva o plano de dados esperado a partir dos arquivos de configuração e permite consultas sobre propriedades de encaminhamento, identificando diversos erros de configuração em duas redes universitárias reais [Brown et al., 2023]. Os mesmos relatam a evolução do Batfish, descrevendo como o uso de Diagramas de Decisão Binária (*Binary Decision Diagrams* - BDDs) elevou em três ordens de grandeza o desempenho da análise em redes com milhares de nós. Essas ferramentas verificam a validade da configuração pretendida antes de sua aplicação, mas não comparam essa configuração contra o estado real do equipamento após a aplicação.

Focando a detecção de *drift* propriamente dita, Horton e Parnin propõem, uma estratégia para detectar *drift* de configuração em trechos de código Python causado por mudanças que quebram compatibilidade em dependências, identificando 248 instâncias de *drift* em um corpus público [Horton e Parnin, 2020]. Embora o domínio seja distinto da configuração de rede, o trabalho demonstra como o *drift* pode ser sutil e cumulativo. Mais próximo do domínio de infraestrutura, um relatório técnico recente descreve a detecção de *drift* em código *Terraform* por meio de *pipelines* do *GitHub Actions*, concluindo que a intenção de uma mudança deve permanecer como responsabilidade do operador mesmo em um fluxo automatizado [Martos, 2026]. O ComplianceNet compartilha essa conclusão, mas resolve uma limitação adicional. Comparações posicionais como as empregadas por ferramentas do tipo *terraform plan* são suscetíveis a falsos positivos quando um dispositivo retorna a mesma lista de elementos em ordem distinta entre leituras sucessivas. Além disso, o ComplianceNet atua na lacuna de integração contínua usando os protocolos padrão de gerenciamento para redes contemporâneas.

Uma tendência recente explora Modelos de Linguagem de Grande Escala (*Large Language Models* - LLMs) para facilitar a configuração de rede. Wang *et al.* propõem o NetConfEval, um conjunto de *benchmarks* para avaliar a capacidade de LLMs de traduzir requisitos em linguagem natural para configurações de baixo nível [Wang et al., 2024]. Hollósi et al. avaliam o uso de IA generativa para produzir configurações NETCONF de baixo nível a partir de modelos YANG [Hollósi et al., 2024]. Li *et al.* propõem o ConfAgent, um agente baseado em LLM para síntese de configuração de rede orientada a intenção [Li et al., 2025]. Esses trabalhos abordam o problema de síntese de configuração a partir de linguagem natural, complementar ao problema de governança contínua tratado pelo ComplianceNet. Dada a declaração da configuração desejada, manualmente ou

com auxílio de um LLM, o ComplianceNet garante que essa declaração continue refletindo a realidade do equipamento, cumprindo o valor operacional de um ferramenta de governança de redes.

3. ComplianceNet

A arquitetura do ComplianceNet, apresentada na Figura 1, é composta por dois fluxos complementares. O fluxo principal corresponde ao *pipeline* de integração contínua executado no GitLab, responsável por validar, aprovar e aplicar alterações propostas pelos operadores. Paralelamente, o fluxo secundário corresponde ao mecanismo de detecção contínua de *drift*, executado de forma independente para identificar alterações realizadas diretamente nos equipamentos fora do processo formal de gerenciamento de mudanças. O fluxo principal compreende as etapas de coleta das configurações dos dispositivos (1), registro das informações no inventário (2), geração do repositório de *templates* (3), validação e comparação das alterações propostas (4) e aplicação da configuração aprovada (5). Por sua vez, o fluxo secundário executa inspeções periódicas da infraestrutura (I), cria automaticamente um *Merge Request* quando identifica uma divergência (II) e, após a aprovação do operador, inicia um novo *pipeline* responsável por sincronizar a configuração implantada no equipamento com o estado desejado armazenado no repositório (III). Essa organização permite que o ComplianceNet trate tanto mudanças planejadas quanto alterações realizadas diretamente nos equipamentos.

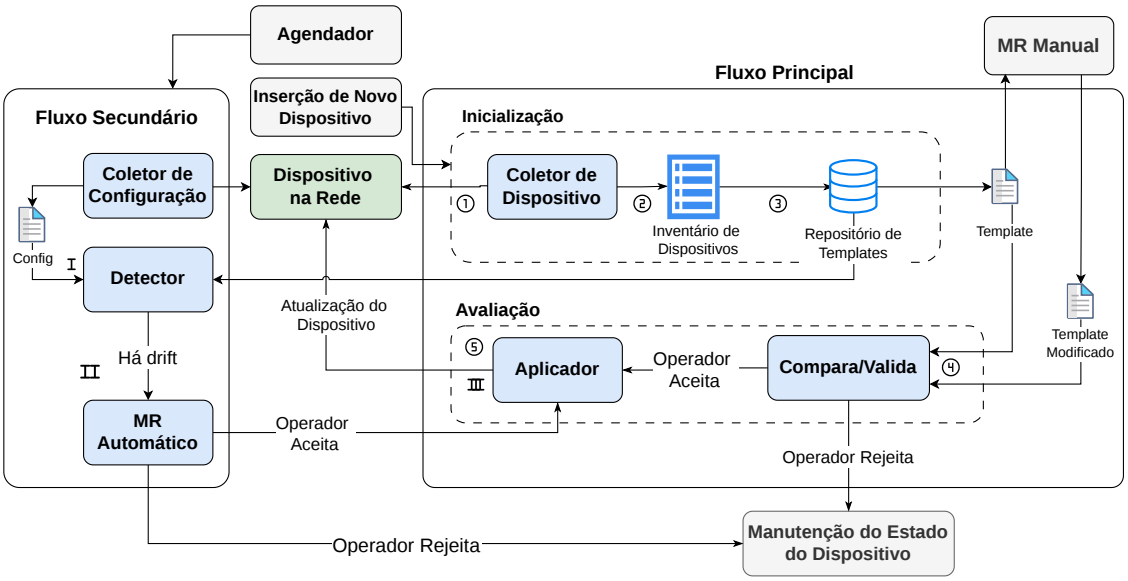

Figura 1. Arquitetura e fluxo operacional do ComplianceNet. O fluxo principal gerencia mudanças aprovadas por meio de *Merge Requests*, enquanto o fluxo secundário monitora continuamente os dispositivos, detecta *drifts* de configuração e cria automaticamente *Merge Requests* para sincronizar o estado declarado e o estado observado da rede.

A configuração desejada de cada equipamento é declarada em arquivos YAML (`host_vars/*.yaml`) versionados em um repositório Git, permitindo que todas as alterações sejam auditáveis, revisáveis e rastreáveis ao longo do ciclo de vida do dispositivo. Inicialmente, esses *templates* são gerados a partir das configurações coletadas dos dispositivos via NETCONF (Passo 1) e sincronizados com o inventário do NetBox[1] por

[1]https://github.com/netbox-community/netbox

meio do *Terraform*[2] (Passo 2), originando o repositório declarativo utilizado pelo ComplianceNet (Passo 3). Após essa etapa de inicialização, novas alterações passam a ser propostas por meio de *Merge Requests* criados manualmente na interface web do GitLab.

Com a abertura de um *Merge Request*, inicia-se a etapa *Compara/Valida* (Passo 4). Nesse componente são executadas verificações estáticas sobre os *templates*, incluindo *lint* de YAML, validação da sintaxe do *Terraform* e verificação da consistência dos *payloads* NETCONF. Em seguida, o mecanismo de comparação converte as configurações desejada e observada em estruturas de dados equivalentes e aplica o método proposto neste trabalho para identificar alterações reais entre elas. O resultado é então apresentado ao operador responsável pela governança das mudanças, que pode aprovar ou rejeitar o *Merge Request*. Caso aprovado, o *pipeline* prossegue para o componente *Aplicador* (Passo 5), responsável por implantar a nova configuração no equipamento.

O fluxo secundário é executado periodicamente para identificar alterações introduzidas fora do processo formal de gerenciamento de mudanças, como intervenções realizadas diretamente nos equipamentos por meio da CLI. Seu principal diferencial está no método de comparação empregado, que alinha os elementos de cada lista utilizando as chaves declaradas nos modelos YANG, em vez de compará-los pela posição ocupada na lista. Nos modelos YANG, a identidade de cada elemento é definida por suas chaves, e não pela ordem em que é retornado pelo dispositivo. Dessa forma, o alinhamento prévio dos elementos permite comparar apenas seus atributos efetivos, tornando a detecção de *drift* independente da ordem de serialização adotada pelo equipamento. A lista de interfaces de uma área OSPF, por exemplo, é alinhada pelo campo *interface-name*, em vez de ser comparada posicionalmente.

A implementação atual apresenta algumas limitações. Os equipamentos avaliados neste trabalho não oferecem suporte completo à escrita de determinadas configurações por meio da operação NETCONF *edit-config*. Nesses casos, o ComplianceNet aplica as alterações utilizando uma sessão SSH não interativa que executa a CLI nativa do equipamento, sem impacto no mecanismo de detecção de *drift*. Além disso, a versão atual da ferramenta não obtém automaticamente as chaves das listas diretamente dos modelos YANG. O mapeamento utilizado pelo método foi construído manualmente a partir da inspeção de respostas *get-config* obtidas via NETCONF, não sendo extraído dos arquivos YANG nem por meio da operação *get-schema*. Embora esse mapeamento seja atualmente manual, as chaves encontram-se formalmente especificadas nos modelos YANG, permitindo sua extração automática em trabalhos futuros.

4. Demonstração e Resultados

O funcionamento do ComplianceNet será demonstrado em uma rede simulada, cuja topologia é apresentada na Figura 2. A rede de demonstração é composta por dez equipamentos Nokia SR Linux interligados por adjacências OSPF, simulados por meio do ContainerLab[3]. Todos os equipamentos possuem suas configurações declaradas em arquivos `host_vars` e continuamente auditadas pelo *pipeline*. Durante a demonstração, serão apresentados a detecção automática de uma alteração realizada diretamente em um equipamento, a criação do *Merge Request* correspondente e o processo de aceitação ou

[2]https://developer.hashicorp.com/terraform
[3]https://containerlab.dev/

rejeição da mudança pelo operador. Dessa forma, a demonstração contemplará o ciclo completo de operação da ferramenta.

A demonstração será realizada por meio de um *laptop* dos próprios autores, a partir do qual será acessada a máquina virtual onde a ferramenta está hospedada. As funcionalidades do ComplianceNet serão demonstradas por meio dos seguintes passos: (*i*) alterar o identificador de roteador (*router-id*) da instância OSPF de um dos equipamentos via CLI; (*ii*) executar o estágio de *drift-detection*, no qual o método identificará corretamente essa única alteração; e (*iii*) interagir com o *Merge Request* criado automaticamente pela ferramenta, aceitando-o ou rejeitando-o. O vídeo demonstrativo da ferramenta está disponível publicamente[4].

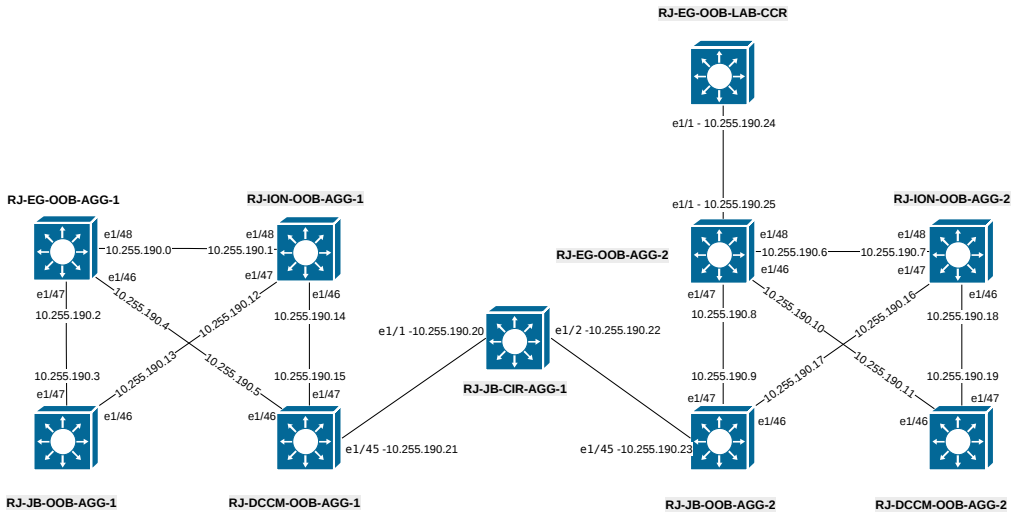

Figura 2. Topologia utilizada na demonstração do ComplianceNet. Os equipamentos podem ter suas configurações modificadas tanto pelo fluxo de gerenciamento de mudanças quanto por alterações manuais, utilizadas para demonstrar o mecanismo de detecção de *drift*.

Aplicam-se quatro métodos de comparação à mesma árvore de configuração obtida via NETCONF: o método proposto, uma comparação de dicionários por posição (*naive-dict-diff*), um *diff* textual sobre a representação serializada em JSON e uma comparação baseada na arquitetura do Terraform. No cenário em que ocorre apenas a reordenação da lista de interfaces OSPF, apenas o método proposto identifica corretamente a ausência de *drift*, enquanto os demais métodos produzem falsos positivos. Nos cenários em que há alterações reais (*router-id* OSPF e *admin-state*), todos os métodos identificam corretamente a presença de *drift*, conforme resumido na Tabela 1.

O método *Terraform-style* consiste em uma reimplementação aproximada do comportamento documentado na literatura sobre o Terraform [Martos, 2026], em que blocos declarados como *TypeList* são comparados de forma posicional. Essa estratégia, amplamente adotada na implementação de *providers* devido à sua simplicidade, pode produzir falsos positivos de *drift* quando ocorre apenas a reordenação dos elementos da lista, pois a identidade de cada elemento é inferida a partir de sua posição. Em contraste, nos modelos YANG, a identidade dos elementos é definida pelas chaves declaradas em cada lista.

[4]https://youtu.be/ySfzWMWEKq4

Dessa forma, o alinhamento prévio dos elementos por suas respectivas chaves torna a comparação independente da ordem em que são retornados pelos dispositivos, eliminando esse tipo de falso positivo.

Nos modelos NETCONF/YANG, os elementos da configuração podem ser classificados como campos escalares ou listas. Um campo escalar corresponde a um valor único associado diretamente a um nó da árvore de configuração, como uma cadeia de caracteres, um número ou um endereço IP. Em contraste, uma lista agrupa múltiplas ocorrências de uma mesma estrutura, sendo cada elemento identificado por uma ou mais chaves. No contexto da demonstração, o *router-id* da instância OSPF e o *admin-state* de uma interface são exemplos de campos escalares, enquanto a lista de interfaces associadas a uma área OSPF constitui uma lista identificada pela chave *interface-name*.

Tabela 1. Tempo de execução (média $\pm$ intervalo de confiança de 95%, $n = 300$ amostras por célula) e taxa de acerto para os diferentes métodos.

Método	Reord. OSPF		RID OSPF		Adm. State	
	Tempo (ms)	Acerto	Tempo (ms)	Acerto	Tempo (ms)	Acerto
ComplianceNet	0.0446 ± 0.0035	100%	0.0393 ± 0.0012	100%	0.0097 ± 0.0009	100%
Naive-dict	0.0578 ± 0.0026	0%	0.0531 ± 0.0013	100%	0.0123 ± 0.0027	100%
Text-diff	0.3761 ± 0.0491	0%	0.3972 ± 0.0232	100%	0.1027 ± 0.0076	100%
Terraform-style	0.0329 ± 0.0010	0%	0.0328 ± 0.0009	100%	0.0072 ± 0.0005	100%

A avaliação de desempenho isola o custo computacional do algoritmo de comparação do custo de comunicação com a rede, permitindo medir exclusivamente o desempenho do método proposto. Para isso, a configuração de cada equipamento é obtida via NETCONF apenas uma vez, sendo cronometrada exclusivamente a execução da função de comparação. O tempo total de execução do *pipeline* depende também da latência da comunicação com os dispositivos e da infraestrutura utilizada, não sendo representativo do custo computacional do método de detecção de *drift*. Para cada combinação de cenário e método, a função de comparação é executada 30 vezes consecutivas sobre o mesmo par de entrada. Considerando os dez equipamentos avaliados, cada célula reúne 300 amostras, a partir das quais são calculados a média e o intervalo de confiança de 95% utilizando a distribuição *t* de Student. Além do tempo de execução, é avaliada a capacidade de cada método em classificar corretamente cada equipamento quanto à presença ou ausência de *drift*. Como todos os equipamentos são submetidos ao mesmo tipo de perturbação em cada cenário, espera-se um comportamento uniforme entre eles, resultando em uma taxa de acerto determinística.

O cenário de Reordenação de Lista OSPF (Reord. OSPF) avalia o comportamento dos quatro métodos na ausência de qualquer alteração de configuração. Nesse cenário, nenhum valor é modificado, nem no equipamento nem na configuração declarada no *Git*. A comparação é realizada sobre a lista de interfaces de uma área OSPF: embora os elementos e seus respectivos valores sejam idênticos, a ordem em que aparecem difere. Enquanto a declaração armazenada no *Git* preserva a ordem de inserção, o equipamento retorna a lista segundo sua representação interna. Como não há alteração de conteúdo, o resultado esperado é a ausência de *drift*. O método proposto identifica corretamente essa situação nos dez equipamentos. Em contraste, os demais métodos, por realizarem comparações posicionais, interpretam a mudança de ordem como uma alteração de configuração e re-

portam falsos positivos de *drift*.

Os cenários de alteração do *router-id* OSPF (RID OSPF) e do *admin-state* (Adm. State) introduzem uma alteração real em um campo escalar. No primeiro, o valor do *router-id* é modificado para simular uma reconfiguração realizada diretamente no equipamento. No segundo, o estado administrativo de uma interface é invertido, simulando o desligamento de uma porta via linha de comando. No cenário de alteração do *admin-state*, a interface avaliada em cada equipamento é escolhida dinamicamente, utilizando a primeira interface retornada pela resposta NETCONF. Essa estratégia é necessária porque a topologia física difere entre os dez equipamentos e nenhuma interface específica está presente em todos eles. Em ambos os cenários, a divergência corresponde exclusivamente à alteração do valor de um campo escalar, e não à reordenação de elementos em uma lista. Nesses casos, todos os quatro métodos identificam corretamente a presença de *drift*.

A inspeção do *diff* lado a lado confirma que os falsos positivos observados no cenário Reord. OSPF decorrem exclusivamente da ordem de retorno da lista de interfaces. O dicionário Python utilizado para gerar os arquivos `host_vars` preserva uma ordem distinta daquela utilizada pelo equipamento em sua resposta NETCONF, embora ambas representem exatamente o mesmo conjunto de configurações. Consequentemente, um mecanismo de detecção de *drift* baseado em *naive-dict-diff*, *diff* textual ou comparação posicional produziria alertas permanentes de *drift* em todos os equipamentos, mesmo na ausência de qualquer alteração real. O alinhamento dos elementos pelas chaves definidas nos modelos YANG elimina esse tipo de falso positivo. Quando um *Merge Request* é gerado em decorrência de um *drift*, o operador responsável pela governança pode rejeitá-lo caso conclua que a alteração não deve se tornar o novo estado desejado da rede. Nesse caso, a configuração previamente armazenada no repositório pode ser reaplicada diretamente pela interface web do GitLab por meio da opção *New Pipeline*, sem necessidade de comandos em terminal. A execução desse novo *pipeline* restaura a configuração desejada no equipamento, reestabelecendo a conformidade entre o estado declarado no repositório e a configuração efetivamente implantada.

5. Conclusão

Este artigo propôs o ComplianceNet, uma ferramenta de governança contínua que detecta *drift* de configuração em equipamentos de rede por meio de um método baseado nas chaves dos modelos YANG obtidas via NETCONF, evitando comparações posicionais e textuais. A ferramenta abre automaticamente *Merge Requests* quando identifica uma divergência genuína, mantendo a decisão final sob responsabilidade do operador. A validação em dez equipamentos reais demonstrou que abordagens posicionais e textuais produzem falsos positivos devido apenas à ordenação não determinística das listas retornadas via NETCONF, enquanto o método proposto distingue corretamente a ausência de divergências de alterações reais. O ComplianceNet integra detecção contínua de *drift*, gerenciamento de mudanças e controle de versões em um único fluxo de governança. Os resultados demonstram que o uso da semântica dos modelos YANG reduz falsos positivos de *drift* sem alterar o fluxo de gerenciamento de mudanças. Como trabalhos futuros, pretende-se estender a classificação de risco das mudanças, ampliar a cobertura da detecção de *drift* para outros equipamentos e protocolos e incorporar observabilidade contínua de métricas de rede.

Referências

Amlou, A., Abane, A., Merzouki, M., Oucheggou, L. A., Maasaoui, Z. e Battou, A. (2023). Automated network programmability using openconfig yang models and netconf protocol. Em *2023 20th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA)*, p. 1–5.

Bajpai, V., Krejčí, R. e Poulopoulos, L. (2021). Managing home routers with netconf over tls and netconf call home.

Brown, M., Fogel, A., Halperin, D., Heorhiadi, V., Mahajan, R. e Millstein, T. (2023). Lessons from the evolution of the batfish configuration analysis tool. Em *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, p. 122–135, New York, NY, USA. Association for Computing Machinery.

Fogel, A., Fung, S., Pedrosa, L., Walraed-Sullivan, M., Govindan, R., Mahajan, R. e Millstein, T. (2015). A general approach to network configuration analysis. Em *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, p. 469–483, USA. USENIX Association.

Hollósi, G., Ficzere, D. e Varga, P. (2024). Generative ai for low-level netconf configuration in network management based on yang models. Em *2024 20th International Conference on Network and Service Management (CNSM)*, p. 1–7.

Horton, E. e Parnin, C. (2020). V2: fast detection of configuration drift in python. Em *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, ASE '19, p. 477–488. IEEE Press.

Li, S., Gan, Z., Liu, J., Gao, C., Li, F., Wu, S., Hu, P. e Li, F. (2025). Confagent: Towards intelligent network configuration via llm agent. Em *2025 IEEE/ACM 33rd International Symposium on Quality of Service (IWQoS)*, p. 1–10.

Martos (2026). Detecting and classifying configuration drift in terraform with github actions.

Wallin, S. e Wikström, C. (2011). Automating network and service configuration using netconf and yang. Em *Proceedings of the 25th International Conference on Large Installation System Administration*, LISA'11, p. 22, USA. USENIX Association.

Wang, C., Scazzariello, M., Farshin, A., Ferlin, S., Kostić, D. e Chiesa, M. (2024). Netconfeval: Can llms facilitate network configuration? *Proc. ACM Netw.*, 2(CoNEXT2).