Engenharia de Software em Redes Programáveis: A Abordagem ProgNet

Moises S. de Sousa[1], **Newton M. Kamimura**[1], **Alexandre Kaihara**[1], **Lucas Bondan** [1,2], **Marcos F. Caetano**[1], **Davi A. Casseb**[1], **João J. C. Gondim**[1] **e Marcelo A. Marotta**[1]

[1] Universidade de Brasília – UnB

[2]Rede Nacional de Ensino e Pesquisa – RNP

moisesssousa98@gmail.com, newton.kamimura@ifsp.edu.br

lucas.bondan@rnp.br, {mfcaetano,davi.casseb,gondim,marcelo.marotta}@unb.br

Abstract. *Redes programáveis, impulsionadas por SDN e P4, oferecem flexibilidade, mas carecem de padrões e metodologias sólidas. A falta de interfaces e testes padronizados em SDN e a programação de baixo nível em P4 geram desafios como conflitos de código, vulnerabilidades e baixa reutilização. Para superar essas limitações, o ProgNet utiliza a modularidade do GNU Radio para organizar redes SDN e P4 de forma intuitiva. Adotando uma abordagem de cima para baixo, alinha o desenvolvimento de software aos objetivos da rede, evitando problemas de desempenho. Um estudo de caso demonstra a reutilização de componentes e a otimização de latência e vazão, validando a viabilidade da modularidade e interfaces padronizadas.*

Abstract. *Programmable networks, driven by SDN and P4, offer flexibility but lack solid standards and methodologies. The absence of standardized interfaces and tests in SDN, along with low-level programming in P4, leads to challenges such as code conflicts, vulnerabilities, and low reusability. To address these limitations, ProgNet leverages GNU Radio's modularity to organize SDN and P4 networks more intuitively. By adopting a top-down approach, it aligns software development with network objectives, preventing performance issues. A case study demonstrates component reuse and optimization of latency and throughput, validating the feasibility of modularity and standardized interfaces.*

1. Introdução

Redes programáveis emergem como uma solução para superar as limitações de dispositivos de hardware dedicados, permitindo que funções de rede sejam desenvolvidas e implementadas em software. Tecnologias como *SDN* [Liatifis et al. 2022] e *P4* [Larsen et al. 2023, Doriguzzi-Corin et al. 2024] desempenham um papel central nesse avanço, viabilizando a criação de aplicações que gerenciam diferentes aspectos da rede, desde o controle centralizado até o processamento detalhado de pacotes. Essa combinação traz benefícios como flexibilidade no desenvolvimento de protocolos, redução de barreiras à inovação e rápida adaptação às demandas dinâmicas da rede [Doriguzzi-Corin et al. 2024]. No entanto, o desenvolvimento de aplicações para *SDN* e *P4* apresenta desafios significativos, especialmente pela falta de práticas consolidadas de engenharia de software e modelos seminais para essas tecnologias.

Embora essas tecnologias possam ser empregadas de forma complementar, o desenvolvimento simultâneo de aplicações de alto e baixo nível enfrenta problemas, como dificuldades na integração e a falta de garantias de eficiência, modularidade e interoperabilidade. A engenharia de software oferece fundamentos essenciais para mitigar esses problemas, promovendo melhores práticas de projeto, modularização e testes. Contudo, tanto o *SDN* quanto o *P4* carecem de abstrações e arcabouços padronizados, o que aumenta a propensão a erros e compromete a qualidade do software [Elkhail and Cerny 2019]. Além disso, a ausência de paradigmas para o desenvolvimento integrado limita a criação de soluções autônomas e escaláveis para toda a pilha de rede [Kellerer et al. 2019].

Neste artigo, propõe-se *ProgNet*, um ambiente de programação inspirado no GNU Radio que utiliza uma abordagem baseada em blocos para simplificar o desenvolvimento de aplicações de rede. Cada bloco representa uma função de rede, permitindo a manipulação modular de elementos da arquitetura e a análise integrada de seu impacto no desempenho geral. O *ProgNet* também promove a reutilização de soluções por meio de uma biblioteca compartilhada, facilitando a colaboração e a inovação. Para validar essa abordagem, utiliza-se o *Lightweight Fog Testbed* (LFT), uma estrutura modular que simula cenários reais, oferecendo suporte para avaliação de desempenho, confiabilidade e escalabilidade de aplicações [Doe and Smith 2022, Brown and Taylor 2020].

O restante deste trabalho está organizado da seguinte forma: na Seção 2, apresentamos os conceitos de engenharia de software aplicados a redes programáveis e os trabalhos relacionados à tais áreas; na Seção 3, detalhamos o *ProgNet* e sua implementação; a Seção 4 discute um estudo de caso; por fim, as conclusões e perspectivas futuras são apresentadas na Seção 5.

2. Programação de Redes por Áreas do Conhecimento

A ampla adoção de redes programáveis baseadas em *software* levou a um aumento significativo no desenvolvimento de código por programadores independentes [Lima et al. 2021a], incluindo contribuições dentro de comunidades de código aberto [Lima et al. 2021b]. Tal adoção leva a demandas no desenvolvimento de software, como garantir um código auto-contido, modular, minimizar erros, reduzir custos e diminuir o tempo necessário para a criação e evolução de uma aplicação de rede [Tornhill and Borg 2022][Saraswat et al. 2019].

As demandas de desenvolvimento motivam a necessidade de implementar práticas de engenharia de *software* para garantir a qualidade das funcionalidades e do código. Tradicionalmente, a engenharia de *software* organiza o processo de desenvolvimento em cinco áreas de conhecimento principais: Requisitos de Software, Projeto de Software, Desenvolvimento de Software, Teste de Software e Manutenção de Software [Bourque and Fairley 2014].

No contexto das redes programáveis, cada uma das cinco áreas da engenharia de *software* oferece conhecimentos valiosos que podem remodelar e aprimorar o desenvolvimento de aplicações de rede. Abaixo, são exploradas cada área de conhecimento a partir de uma perspectiva de engenharia de software, fornecendo uma breve descrição, contextualizando-a em redes programáveis, destacando conceitos-chave, possíveis usos e discutindo desafios abertos.

Os **Requisitos de Software** definem as características essenciais que o *software*

deve apresentar para resolver problemas do mundo real. A Engenharia de Requisitos envolve a gestão sistemática desses requisitos, com foco na elicitação, análise, especificação e validação. Em redes programáveis, os requisitos especificam o comportamento do sistema, incluindo segurança, escalabilidade e desempenho.

Já o **Projeto de Software** abrange tanto o processo de definição da arquitetura do *software* quanto da especificação das interfaces e características dos componentes, assim como os resultados desse processo. Durante a fase de projeto, os requisitos de *software* são analisados para produzir uma estrutura que guiará a construção. Essa estrutura é articulada por meio de uma arquitetura de referência – uma descrição de como o *software* é decomposto em componentes – e pela especificação dos componentes de forma suficiente para a construção. Isso ajuda a padronizar os sistemas e melhora a compreensão, além de aumentar a reutilização entre projetos [Kreutz et al. 2015].

O **Desenvolvimento de Software** refere-se ao processo de transformação de um projeto detalhado em um *software* funcional, incluindo atividades como codificação, teste, depuração e integração. Na construção de *software* para redes programáveis, as linguagens de programação desempenham um papel central. Esse desenvolvimento apresenta desafios significativos em redes programáveis. Por exemplo, garantir que o código desenvolvido seja minimalista o suficiente para operar em ambientes de rede de alta capacidade exige uma atenção cuidadosa ao desempenho e à utilização de recursos. A otimização de códigos para reduzi-los e torná-los menos complexos demanda conhecimentos das várias áreas do saber, como métodos numéricos e teoria da computação.

A fase de **Teste de Software** é uma atividade crítica que visa verificar se o *software* atende aos requisitos especificados e funciona conforme o esperado. Em redes programáveis, o teste é particularmente desafiador devido à natureza distribuída e dinâmica dos sistemas. No campo de testes, "*Fuzz Testing SDN*" propõe o uso de técnicas de *fuzzing* para identificar vulnerabilidades e assegurar a robustez de aplicações SDN [Saraswat et al. 2019]. Essa técnica se mostra promissora, mas ainda enfrenta desafios na simulação de tráfego realista e na avaliação do impacto em diferentes cenários de rede.

Por fim, a **Manutenção de Software** engloba as atividades necessárias para modificar um *software* após sua entrega, visando corrigir defeitos, melhorar o desempenho ou adaptá-lo a um ambiente operacional em mudança. Uma das principais dificuldades na manutenção de *software* em redes programáveis é garantir que as mudanças introduzidas não afetem negativamente o funcionamento geral do sistema. Isso requer um planejamento cuidadoso e a utilização de ferramentas que possam ajudar a avaliar o impacto das mudanças antes de implementá-las. Por fim, a documentação clara e detalhada é fundamental para facilitar a manutenção, especialmente em sistemas complexos que envolvem múltiplas camadas de *software* e hardware.

A integração de práticas de Engenharia de *software* (SwEng) no desenvolvimento de redes programáveis tem sido explorada por diversas propostas. Trabalhos como "*Software Engineering Challenges for SDN*" destacam desafios significativos no uso de princípios de SwEng em SDN, abordando desde ao levantamento de requisitos até a validação e testes de sistemas distribuídos [Lima et al. 2021a]. Essas iniciativas reconhecem que a natureza dinâmica e distribuída de redes programáveis exige adaptações em

metodologias tradicionais. Outra abordagem notável é apresentada em "*Applying Agile Development to SDN Applications*", que explora como metodologias ágeis podem ser adaptadas para o desenvolvimento de aplicações SDN [Lima et al. 2021b]. Este trabalho enfatiza ciclos curtos de desenvolvimento, favorecendo uma entrega incremental de funcionalidades, mas carece de uma análise detalhada entre a garantia de manutenção do código e a sua longevidade.

3. ProgNet

Na seção anterior, identificou-se problemas que afetam a criação e o gerenciamento de softwares para redes programáveis, como a complexidade da interconexão de softwares, a adaptação a diferentes topologias e a necessidade de uma interface intuitiva. Para enfrentar esses desafios, propõe-se o ProgNet, um *framework* modular e extensível que facilita a construção e reutilização de softwares para redes programáveis. Dessa forma, ProgNet tem como foco as fases de construção, teste e manutenção de software citadas anteriormente (Seção 2).

Uma das premissas de sua existência é que a arquitetura do ProgNet precisa ser modular e permita a criação, configuração e interconexão de blocos de rede de maneira simples. Para tornar o desenvolvimento de software de redes modular, pode-se usar a intuição de que cada pedaço de código pode ser transformado em um bloco de rede. A intuição por trás dessa suposição baseia-se na forma como uma função de programação é estruturada, considerando que suas entradas são parâmetros e suas saídas são estruturas de dados que podem ser usadas como entradas para outro bloco de rede. Para que isso seja alcançado, usuário do ProgNet possui duas formas de criar seus blocos dentro do sistema, como descritas a seguir.

3.1. Ferramentas utilizadas

Para implementar o ProgNet, duas ferramentas de software essenciais são analisadas: o *LFT* (*Lightweight Fog Testbed*) e o *GNU Radio*. O *LFT* funciona como uma plataforma de teste projetada para simular e avaliar ambientes de computação em névoa [Yang et al. 2022]. Ele oferece um ambiente controlado e adaptável para avaliar aplicações e serviços em contextos de computação distribuída e descentralizada. O *GNU Radio*, um *framework* de código aberto, facilita o desenvolvimento de sistemas de rádio definido por software (*SDR* ou *Software Defined Radio*). Ele emprega uma abordagem baseada em blocos para simplificar o *design* e a implementação de fluxos de processamento de sinal complexos [Ferreira et al. 2012].

Essa abordagem facilita o design e a implementação rápidos de fluxos de sinal por meio de uma interface gráfica intuitiva, simplificando o desenvolvimento de sistemas *SDR*. Ao combinar essas capacidades, o ProgNet pode oferecer uma base sólida para o desenvolvimento e aprimoramento de redes programáveis, avançando significativamente tanto no *design* quanto na funcionalidade.

3.2. Programação de blocos personalizados

Nesse modelo, o usuário tem à disposição bibliotecas *Python* como referência e um código padrão para a criação de blocos simples. Durante esta etapa, associada ao Projeto de Software, o usuário deve identificar os requisitos, definindo as funcionalidades específicas que o bloco precisa atender dentro da rede programável.

A etapa de desenvolvimento ocorre quando o usuário implementa toda a lógica de programação necessária para criar os blocos que atenderão aos requisitos levantados. Após o desenvolvimento, o bloco é submetido a um processo de testagem, onde o sistema verifica erros de sintaxe no código *Python* e alerta o usuário sobre problemas identificados. Contudo, essa verificação inicial se limita a erros de sintaxe, não abrangendo validações do funcionamento lógico ou operacional da rede.

Quando o código é compilado sem erros, o bloco é considerado gerado, podendo ser conectado a outros blocos existentes ou futuros. Esta integração é parte do processo de Manutenção de Software, pois possibilita a evolução contínua da rede programável por meio da adição de novos blocos.

3.3. Uso de blocos pré-fabricados

Na segunda abordagem, o usuário tem acesso a blocos pré-fabricados, com funções já definidas. Aqui, o Requisito de Software principal é identificar quais blocos atenderão às necessidades da rede programável. Como o nível de programação é zero, a etapa de desenvolvimento se resume à configuração das variáveis disponíveis nesses blocos. Porém, o controle dessas variáveis exige conhecimento prévio sobre conceitos de redes, pois erros de configuração podem comprometer o funcionamento. Por exemplo, ao usar um bloco UDP, o usuário deve configurar o tamanho dos pacotes corretamente, respeitando as limitações do protocolo. Configurar pacotes com tamanhos superiores a 512 bytes, por exemplo, resultaria em erro.

Assim como na primeira abordagem, há um processo de testagens durante a validação dos parâmetros configurados. O ProgNet verifica inconsistências e notifica o usuário sobre problemas, permitindo a correção antes de gerar o bloco. Uma vez validado e compilado sem erros, o bloco está pronto para ser utilizado, garantindo a manutenção e evolução contínua da rede programável.

3.4. Montagem e validação da rede programável

Com os dois métodos de geração de blocos explicados, o usuário pode montar sua rede programável utilizando a quantidade de blocos que julgar necessária, combinando blocos personalizados e pré-fabricados. Essa etapa está fortemente ligada ao Projeto de Software, pois exige planejamento e definição clara de como os blocos se conectarão para atender aos objetivos da rede.

Após a junção dos blocos, a rede é submetida a um processo de compilação, que inclui uma etapa de Teste de Software. Caso existam inconsistências, como a saída de um bloco X não corresponder à entrada esperada do bloco Y, o sistema gera mensagens de erro no console, permitindo a correção. Alternativamente, se não houver erros, a saída será composta pelos dados gerados pela rede, que deverão ser analisados para validar seu funcionamento em conformidade com os conceitos de redes.

A rede construída pode ser ajustada ou ampliada ao longo do tempo, conforme surgirem novos requisitos ou necessidades de melhoria. A integração de novos blocos, ajustes nas configurações e validação de desempenho fazem parte do processo de Manutenção de Software, garantindo que a rede continue operando de forma eficiente e alinhada às demandas do usuário. Com essa abordagem estruturada, o ProgNet se torna uma poderosa

ferramenta para a criação de redes programáveis, alinhada às boas práticas de construção de software.

4. Caso de estudo

Para instrumentalizar as propriedades desta proposta, foi elaborado um cenário de testes na ferramenta *LFT* atuando com o *GNU Radio*, onde uma rede emulada foi construída para aferir as funcionalidades. Desse modo, seguindo todos os conceitos de um abordagem estruturada preconizada pelo ProgNet o cenário abaixo foi elaborado:

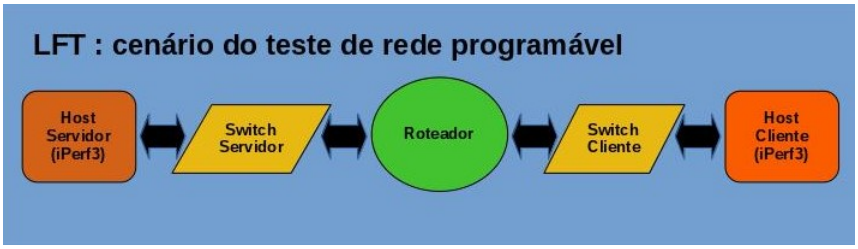

Figura 1. Cenário do Teste construído no *LFT*, envolvendo *GNU Radio*, de uma rede programável aplicando ProgNet.

Para este cenário foi utilizado o aplicativo *iPerf* na sua versão 3, para criarmos um *host* servidor e um *host* cliente, no qual tráfego de arquivos de dados foram usados para transferir do servidor para o cliente e vice-versa, realizados com sucesso, demonstrando que a estruturação permitiu identificar todos os componentes envolvidos na rede emulada, permitindo clareza e objetividade na realização dos testes. Assim, o ProgNet se demonstrou eficaz na gestão da configuração e no controle do cenário operacional, permitindo obter uma visão de gestão do ambiente emulado.

O cenário de testes foi criado usando o sistema operacional Linux Ubuntu versão 24.04.1 LTS, juntamente com o software *GNU Radio* e o software *LFT* (*Lightweight Fog Testbed*). O *GNU Radio* atua como interface gráfica para o *LFT*, servindo também como plano de controle, através de fluxogramas. O *LFT* lida com a simulação real da rede *LTE* usando contêineres *Docker* e o *srsRAN*. Os fluxogramas do *GNU Radio* utilizados com o *LFT*, são uma combinação de blocos *Out-Of-Tree* (*OOT*) integrados e personalizados, que enviam comandos de controle (iniciar, parar, configurar) por meio de passagem de mensagens para scripts *Python* que gerenciam os componentes *Dockerizados* do *LFT* (*eNodeB*, *EPC*, *UE*, etc.).

Além disso, os blocos de origem *OOT* personalizados no *GNU Radio* recebem dados (sinais simulados, estatísticas de rede) do *LFT*, normalmente por meio de soquetes TCP, para visualização usando blocos *sink* personalizados. O teste teve duração média de 40 segundos e foi feito entre dois *hosts*, o servidor e o cliente, medindo a performance do tráfego de rede TCP entre eles, além do *throughput*, *jitter* e perda de pacotes. Parte dessa integração é percebida através dos dados expostos na figura 2.

O gráfico apresenta a taxa de transferência de dados (throughput) entre dois *containers* com medições realizadas utilizando o LFT (*Lightweight Fog Testbed*). A linha do gráfico demonstra como a velocidade de transmissão de dados flutua ao longo do tempo, permitindo visualizar variações no desempenho da comunicação entre os *containers* testados. A análise dos resultados, obtidos com o LFT, é essencial para identificar padrões de

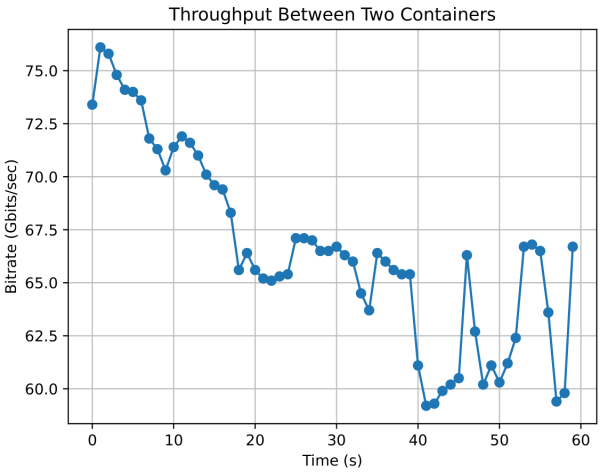

Figura 2. *Throughput* entre dois *containers*

estabilidade, picos ou quedas na taxa de transferência. Esses dados fornecem resultados sobre a eficiência e a consistência do fluxo de dados. Destacam-se eventuais limitações na infraestrutura ou possíveis áreas de otimização no sistema. Com o LFT, é possível ajustar cenários e parâmetros para melhorar o desempenho e atender a demandas específicas de aplicações *fog computing*.

5. Considerações Finais

O ProgNet está atualmente em suas fases iniciais de desenvolvimento, com avanços significativos já alcançados. Até o momento, já existe uma integração, mesmo que tímida, do *LFT* (*Lightweight Fog Testbed*) com o *GNU Radio*, estabelecendo uma conexão eficaz para gerar e processar tráfego de rede usando o *iPerf*. Essa integração representa um marco importante, demonstrando a viabilidade do sistema e seu potencial para simular e analisar redes em ambientes controlados.

À medida que avançamos, os próximos passos envolverão a definição e implementação dos blocos com base nos requisitos e etapas de design previamente estabelecidos. Esse processo incluirá a criação e integração de novos componentes de rede adaptados às necessidades educacionais e às especificações técnicas do sistema.

A próxima fase de desenvolvimento será crucial para o refinamento do ProgNet, garantindo que ele ofereça uma experiência de aprendizado robusta e eficiente. Um dos objetivos do ProgNet é servir como uma ferramenta educacional para instituições, proporcionando uma plataforma prática que facilita a compreensão dos alunos sobre redes. Permitindo que os estudantes experimentem e explorem conceitos complexos de forma interativa, o sistema visa tornar o aprendizado mais dinâmico e acessível.

A conclusão bem-sucedida dessas etapas permitirá que o ProgNet se torne uma ferramenta valiosa na educação em redes, fornecendo um ambiente de aprendizado dinâmico e adaptável, além de ter alto potencial promissor no emprego em âmbito corporativo.

Referências

Bourque, P. and Fairley, R. E., editors (2014). *SWEBOK: Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, Los Alamitos, CA, version 3.0 edition.

Brown, A. and Taylor, M. (2020). Lightweight self-organising distributed monitoring of fog infrastructures. *Journal of Fog Computing*, 7(3):123–135.

Doe, J. and Smith, J. (2022). Lightweight self-adaptive cloud-iot monitoring across fed4fire+. In *Proceedings of the 2022 IEEE International Conference on Cloud Computing*, pages 56–63.

Doriguzzi-Corin, R., Knob, L. A. D., Mendozzi, L., Siracusa, D., and Savi, M. (2024). Introducing packet-level analysis in programmable data planes to advance network intrusion detection. *Computer Networks*, 239:110162.

Elkhail, A. A. and Cerny, T. (2019). On relating code smells to security vulnerabilities. In *2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, pages 7–12.

Ferreira, P. V., Diniz, P., Veiga, A., and Carneiro, M. (2012). Frequency response acquisition of a digital radio transceiver using usrp module and gnu radio software. In *2012 Fourth International Conference on Computational Intelligence, Modelling and Simulation*, pages 243–248.

Kellerer, W., Kalmbach, P., Blenk, A., Basta, A., Reisslein, M., and Schmid, S. (2019). Adaptable and Data-Driven Softwarized Networks: Review, Opportunities, and Challenges. *Proceedings of the IEEE*, 107(4):711–731.

Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

Larsen, J., Guanciale, R., Haller, P., and Scalas, A. (2023). P4r-type: a verified api for p4 control plane programs (technical report).

Liatifis, A., Sarigiannidis, P., Argyriou, V., and Lagkas, T. (2022). Advancing sdn from openflow to p4: a survey. *ACM Computing Surveys*, 55.

Lima, A., Juca, S., Lemos, P., and Silva, S. (2021a). AplicaÇÃo dos conceitos sdn, devops e infraestrutura como cÓdigo no processo de ensino e aprendizagem de ciÊncias e engenharias. *Enciclopédia Biosfera*, 18.

Lima, A., Juca, S., Lemos, P., and Silva, S. (2021b). AplicaÇÃo dos conceitos sdn, devops e infraestrutura como cÓdigo no processo de ensino e aprendizagem de ciÊncias e engenharias. *Enciclopédia Biosfera*, 18.

Saraswat, S., Agarwal, V., Gupta, H. P., Mishra, R., Gupta, A., and Dutta, T. (2019). Challenges and solutions in Software Defined Networking: A survey. *Journal of Network and Computer Applications*, 141(March):23–58.

Tornhill, A. and Borg, M. (2022). Code red: The business impact of code quality – a quantitative study of 39 proprietary production codebases.

Yang, L., Wen, F., Cao, J., and Wang, Z. (2022). Edgetb: a hybrid testbed for distributed machine learning at the edge with high fidelity. *IEEE Transactions on Parallel and Distributed Systems*, 33(10):2540–2553.