nsQUIC: Uma Extensão para Simulação do Protocolo QUIC no NS-3

Yan Soares Couto[1], **Diego Camarinha**[1], **Daniel Macêdo Batista**[1]

[1]Instituto de Matemática e Estatística – Universidade de São Paulo (USP)
São Paulo – SP

{yancouto,diegoamc,batista}@ime.usp.br

Abstract. *Although TCP provides many guarantees in relation to UDP, adapting it to new applications can take many years. In this context, Google has developed the QUIC protocol in order to implement evolutions more quickly. Google is already using QUIC under HTTP and conducting measurement experiments, but these experiments are difficult to replicate for several reasons. In this paper, we were able to work around these difficulties through a new, non-trivial extension developed to add QUIC to NS-3. More specifically, this extension integrates the QUIC code as a new NS-3 application, switching its communication with system sockets to simulator sockets and changing its behavior to work with discrete, rather than continuous, time line.*

Resumo. *Apesar do TCP fornecer muitas garantias em relação ao UDP, adaptá-lo para novas aplicações pode levar muitos anos. Nesse contexto, o Google desenvolveu o protocolo QUIC, a fim de implementar evoluções mais rapidamente. O Google já vem usando o QUIC sob o HTTP e realizando experimentos de medição, mas estes experimentos são difíceis de serem reproduzidos por diversas razões. Neste artigo, mostramos como contornar essas dificuldades por meio de uma nova extensão, não trivial de ser desenvolvida, que adiciona o QUIC ao NS-3. A extensão integra o QUIC como uma nova aplicação do NS-3, substituindo sockets do sistema pelos do simulador e alterando seu comportamento para lidar com passagem de tempo discreta ao invés de contínua.*

1. Introdução

O uso da Internet mudou consideravelmente nos últimos anos mas, apesar das mudanças, o protocolo de transporte utilizado pela maioria das aplicações continua sendo o TCP (*Transmission Control Protocol*). Muitas propostas de melhorias no TCP já foram apresentadas [Casetti et al. 2002] [Ha et al. 2008] [Radhakrishnan et al. 2011], porém colocá-las em prática não é simples: como o TCP é implementado no kernel dos sistemas operacionais, todos os nós envolvidos no transporte precisam ser atualizados para que uma mudança tenha efeito. Esses nós incluem clientes, servidores, *firewalls* e *proxies* sobre os quais desenvolvedores podem não ter controle. Consequentemente, pode demorar anos até que uma atualização no TCP seja realmente implantada e usada. Por isso, o TCP ainda não se adaptou a alguns cenários de rede, e deixa a desejar em múltiplos casos.

Neste contexto, o Google criou o QUIC (*Quick UDP Internet Connections*) [Google 2017b], um novo protocolo de transporte cujo objetivo é ter desempenho melhor que o TCP mantendo as mesmas garantias, como entrega em ordem, e melhorando

algumas técnicas, como estabelecimento de conexão. Justamente para evitar que todos os nós da rede tenham que ser atualizados para que novas técnicas e algoritmos sejam testados, o QUIC é implementado em espaço de usuário, adicionando funcionalidades ao protocolo UDP (*User Datagram Protocol*). Como todos os nós da Internet já sabem lidar com UDP (ele é usado com o QUIC sem necessidade de modificações), só o cliente e o servidor precisam conhecer o protocolo QUIC.

Lançar novas versões do QUIC é, portanto, mais rápido e ele tem sido desenvolvido ativamente nos últimos anos. Como o Google desenvolve o navegador Google Chrome e possui muitos servidores no mundo inteiro, a empresa usa essa infraestrutura para testar o QUIC. Dois estudos [Wilk et al. 2015][Langley et al. 2017] divulgados pelo Google indicam que o QUIC tem desempenho melhor que o TCP em muitas situações. Os testes realizados para embasar essa conclusão, entretanto, são difíceis de serem reproduzidos já que o Google não disponibiliza os cenários exatos nos quais os testes foram executados e é implausível ter a mesma infraestrutura de servidores, clientes e usuários que o Google tem.

Neste artigo é apresentado o `nsQUIC`, uma extensão que permite a simulação do protocolo QUIC no simulador de redes NS-3 [Riley and Henderson 2010]. Pelo que sabemos, nenhum outro trabalho até agora propôs a simulação do protocolo QUIC conforme feito no `nsQUIC`. As vantagens em ter uma ferramenta que permita simulações, em relação às medições, são que não há dependências nem risco de interferências externas, não há custo com outros equipamentos como roteadores e servidores e o pesquisador tem controle sobre as configurações de todos os nós da rede de forma mais simples do que realizado nos trabalhos anteriores, além da facilidade para escalar os cenários dos experimentos. Com o objetivo de permitir a reprodução dos experimentos, todo o código implementado está publicamente disponível [1][2][3] sob a licença GNU *General Public License* v2.0.

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta trabalhos relacionados. A Seção 3 apresenta a arquitetura do `nsQUIC` e a descrição das suas principais funcionalidades. A Seção 4 descreve dois cenários de simulação simulados com o `nsQUIC`, bem como discute os resultados encontrados, servindo assim como exemplos de casos de uso. A Seção 5 apresenta as conclusões e algumas sugestões para trabalhos futuros. Por fim, a Seção 6 descreve a demonstração planejada para o SBRC, informando equipamentos necessários para tal.

2. Trabalhos relacionados

O próprio Google já divulgou resultados de experimentos que foram realizados com o QUIC rodando em produção na Internet [Wilk et al. 2015]. Esses experimentos apontaram que o QUIC apresenta vantagens significativas em relação ao TCP: desempenho 3% melhor para carregar uma página do Google Search e 30% menos *rebuffers* em vídeos do YouTube. Em outro estudo mais recente [Langley et al. 2017], o Google confirma os ganhos de desempenho do QUIC bem como dimensiona seu espaço na Internet atual-

[1]URL da ferramenta: https://gitlab.com/diegoamc/ns-3-quic-module

[2]URL dos manuais e documentação: https://gitlab.com/diegoamc/ns-3-quic-module/blob/master/README.md

[3]URL de um vídeo explicativo: https://youtu.be/2lKJyD2ai4Y

mente: o QUIC já é responsável por 7% de todo o tráfego na Internet. Com o nsQUIC é possível conduzir experimentos reprodutíveis com o QUIC, em diversos cenários, sem a necessidade de possuir a infraestrutura do Google para obter resultados relevantes.

Outros trabalhos já tentaram avaliar o desempenho do QUIC usando emulação para criar cenários de rede de modo que os experimentos fossem reprodutíveis. No mais recente deles [Kakhki et al. 2017], os autores tinham controle sobre o roteador que conectava o cliente à Internet. Com isso, puderam emular diversos cenários de rede, controlando a quantidade de banda disponível, atraso, perda de pacotes e *jitter*. Eles comprovaram que o QUIC tem desempenho melhor que o TCP na maioria das situações e mostraram algumas em que o TCP é melhor, como no caso de muitas transferências de poucos dados. Com o nsQUIC, está sendo proposta uma maneira alternativa para criar experimentos reprodutíveis com o QUIC, sem custos com equipamentos adicionais como roteadores e servidores e com maior controle sobre os cenários de teste.

O nsQUIC avança o estado da arte na realização de experimentos com o protocolo QUIC pelo fato de permitir que esses experimentos sejam realizados por meio de simulação. Além disso, o nsQUIC é baseado no simulador NS-3, um simulador bastante difundido e usado pela comunidade que realiza pesquisa em redes de computadores, facilitando assim a sua utilização.

3. Arquitetura e principais funcionalidades do nsQUIC

Esta seção descreve as principais características do QUIC e as dificuldades para integrar seu código no NS-3, explicando as decisões que foram tomadas para superá-las até, finalmente, chegar na primeira versão do nsQUIC. Para fazer a integração, foi usada a versão 41 do QUIC, disponível em [Google 2017a], e a versão 3.27 do NS-3. Ao fim da seção são destacadas as principais funcionalidades do nsQUIC.

3.1. O protocolo QUIC

O QUIC se apoia em estudos e técnicas previamente propostas para implementar suas funcionalidades. As principais delas são o estabelecimento de conexão em 0-RTT (*Round-trip Time*), multiplexação e controle de congestionamento otimizado.

A primeira delas permite que um cliente que tenha previamente se comunicado com um servidor possa iniciar uma nova conexão sem a necessidade do *handshake* inicial implementado pelo TCP. Multiplexação permite que uma transferência seja dividida em diferentes fluxos lógicos de dados, mitigando o impacto de *head of line blocking*. Por fim, o controle de congestionamento do QUIC é uma implementação do TCP Cubic com melhor sinalização de ACKs e estimativas de RTT e largura de banda mais precisas.

3.2. Arquitetura do nsQUIC

O NS-3 é um simulador de redes discreto composto de módulos. Cada módulo tem uma funcionalidade e eles têm o mínimo de acoplamento entre si. O nsQUIC foi implementado como um módulo para o NS-3, e nele foi encapsulada toda a funcionalidade de clientes e servidores QUIC, adaptando o código disponibilizado pelo Google. Ambos os projetos estão escritos em C++. Um resumo do tempo gasto na implementação do nsQUIC está ilustrado na Figura 1, na qual o diâmetro de cada círculo cinza está diretamente proporcional ao tempo gasto nas tarefas descritas nestes círculos.

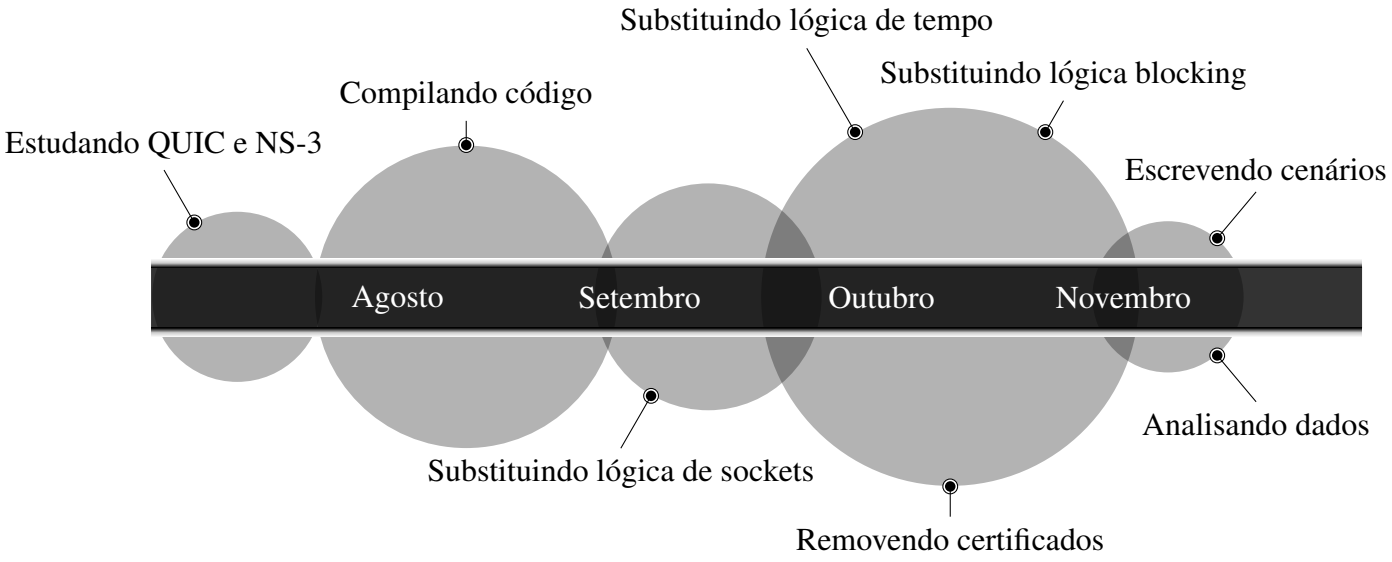

Figura 1. Linha do tempo aproximada do tempo gasto nas tarefas relatadas.

Ao escrever o código do cliente e do servidor, usando funções do código do QUIC, foi necessário passar o código do QUIC para o novo módulo do NS-3. A base de código é grande, tendo mais de quatro mil arquivos de implementação e vários outros projetos somados a esta base. Além disso, enquanto o NS-3 usa o `Waf` [Thomas Nagy 2017] como ferramenta de compilação do código, o QUIC usa `Gn` [Google 2017c] e `Ninja` [Ninja 2017], ferramentas criadas para manutenção do Google Chrome. Assim, não é direta a forma de acoplar os dois projetos.

Uma solução considerada foi copiar todo o código-fonte do QUIC para o módulo e tentar compilá-lo usando o `Waf`. Essa opção foi descartada pois algumas funções ou variáveis eram implementadas em mais de um arquivo, causando erros de ligação. Assim, foram escolhidos os arquivos (`.h`) que declaram as funções utilizadas, e as implementações (arquivos `.cc`) relacionados a eles. Entretanto estes arquivos dependiam de outros, e estes de outros, e assim por diante.

Usando opções de compilação especiais do compilador `g++` (`-M` e `-MM`) e estudando as ações realizadas pelas ferramentas de compilação foi possível copiar apenas os arquivos necessários e deduzir as opções necessárias para compilá-los. O código do QUIC, porém, continuava a fazer as chamadas de função usuais para lidar com *sockets* UDP do sistema, ou seja, usava as funções da biblioteca `sys/socket.h`.

3.3. Adaptação do código do QUIC

Como próximo passo, era necessário modificar o código do QUIC para usar os *sockets* UDP do NS-3 ao invés dos *sockets* UDP do sistema, definidos em `sys/socket.h`. Para isso, foi feita uma cópia dessa biblioteca e todas as suas funções foram reimplementadas para usar os *sockets* definidos no NS-3.

Entretanto, o código de cliente e servidor do QUIC assumem que o tempo é contínuo. O código de servidor, por exemplo, fica em um laço infinito esperando por pacotes vindos do cliente. Além disso, esses códigos nunca devolvem o controle para o código que os chamou, o que conflita com o modo que o código de uma simulação no

NS-3 funciona. Nele, uma simulação é baseada em eventos e, ao invés de o servidor esperar em um laço infinito pelo pacote vindo do cliente, ele deve registrar uma função que será chamada quando um pacote chega. Depois dessa função fazer o que precisa, ela deve retornar controle ao NS-3 para então chamar o próximo evento.

Para isso funcionar da maneira correta, foi necessário modificar o código do cliente e servidor QUIC, trocando chamadas de funções que bloqueiam até receber algum pacote por funções de *callback*. O sistema de alarme do QUIC também teve que ser modificado para usar o esquema de tempo do NS-3.

3.4. Principais funcionalidades do nsQUIC

Como o `nsQUIC` é uma aplicação do NS-3, todas as principais funcionalidades de uma aplicação NS-3, como a disponibilização de um "*helper*" de criação, configuração do endereço e configuração da porta, podem ser utilizadas. Além disso, todas as funcionalidades do NS-3 continuam sendo possíveis de serem utilizadas, como definição de cenários de redes cabeadas e sem fio, configuração de atributos dos enlaces, geração de arquivos de *trace*, etc... Especificamente com relação ao protocolo QUIC, o `nsQUIC` permite a definição de clientes e servidores, que recebem parâmetros relacionados com sockets UDP do NS-3 e a quantidade de dados que os servidores enviarão para os clientes.

Diversos exemplos estão disponíveis juntamente com o código-fonte do `nsQUIC` a fim de facilitar a utilização do mesmo pela comunidade.

4. Experimentos de simulação

O `nsQUIC` já foi utilizado para simular diversas topologias de rede, mas por restrições de espaço serão relatados alguns resultados obtidos apenas com as simulações das topologias ilustradas na Figura 2 e na Figura 3, variando a quantidade de dados enviados e o atraso a fim de avaliar o desempenho do QUIC em relação àquele de duas versões do TCP (o tempo para transmissão dos dados foi a métrica comparada). Foram utilizadas duas versões do TCP para comparação com o QUIC: o TCP Hybla e o TCP NewReno. O primeiro foi escolhido porque, em experimentos preliminares, ele se mostrou ser o TCP com os melhores resultados em relação ao QUIC. O segundo foi escolhido porque é a primeira versão que apresenta as quatro fases de controle de congestionamento (partida lenta, prevenção de congestionamento, recuperação de perda e retransmissão rápida) que outras variações do TCP também usam. O ideal seria comparar o QUIC com o TCP Cubic, pois essa é a versão padrão do Linux e utiliza o mesmo controle de congestionamento padrão do QUIC. Porém, ela não está implementada na versão 3.27 do NS-3.

O TCP Hybla [Caini and Firrincieli 2004] propõe uma forma para resolver o desempenho ruim do TCP em conexões com alto atraso, modificando as regras de alteração na janela de congestionamento. O TCP NewReno [Floyd et al. 2004] melhora o algoritmo de recuperação de perda de seu antecessor, o TCP Reno, mantendo a janela de congestionamento cheia mesmo quando há ACKs duplicados.

4.1. Cenário simples

A topologia exibida na Figura 2 serve para avaliar o desempenho do QUIC em situações "ótimas", ou seja, sem levar em conta perdas e nós intermediários, simulando uma conexão física direta entre dois computadores. Quando não especificado nas escalas dos

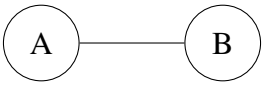

Figura 2. Topologia simples com 2 nós.

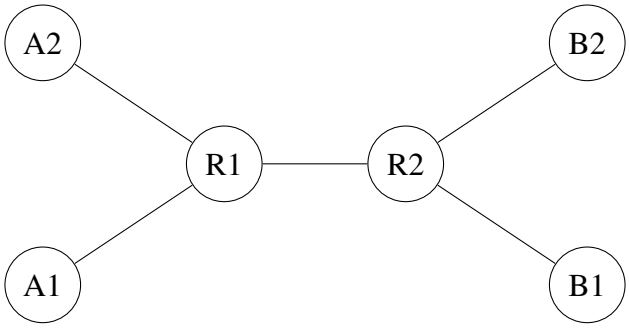

Figura 3. Topologia Dumbbell com 4 nós e 2 roteadores.

gráficos ou nos rótulos das figuras, todos os experimentos nessa topologia consideraram uma transferência de 100MB de B (servidor) para A (cliente), atraso no enlace de 30ms, capacidade de transmissão de 20Mbps e sem perda de pacotes. Todos os resultados foram obtidos com 1 execução de cada configuração de cenário, o que é suficiente dado o caráter determinístico das simulações. Além disso, o tempo de execução apresentado é aquele retornado pela simulação no NS-3 e não o tempo de relógio real.

O gráfico na Figura 4 mostra como o tempo de transmissão varia em relação à quantidade de dados enviados, sem perda de pacotes. Nota-se que para uma transferência de poucos dados (1KB), o QUIC tem desempenho um pouco pior do que o do TCP. Isso se deve ao mecanismo de partida lenta empregado pelos protocolos. O algoritmo usado pelo QUIC, chamado *Hybrid Slow Start*, sai rápido do estado de partida lenta devido a um aumento no RTT mínimo registrado pelo servidor. Nesse caso, o algoritmo entende que esse aumento é indicativo de congestionamento, reduzindo a taxa de transmissão. Já para transferências de 10KB até 1MB, o QUIC tem desempenho melhor porque seu algoritmo de controle de congestionamento é mais agressivo, aumentando mais e com maior frequência a janela de congestionamento. Para transferências entre 10MB e 1GB quase não há diferença nos tempos de transmissão porque nesses casos os algoritmos de controle de congestionamento das duas versões de TCP tem tempo suficiente para alcançar a máxima utilização da largura de banda disponível.

4.2. Dumbbell

A topologia exibida na Figura 3 serve para avaliar os protocolos em um cenário onde há tráfego de interferência. Cada nó Ai (cliente) recebe dados do nó Bi (servidor). Conectando os nós há apenas dois roteadores em linha (Ri), o que pode causar perda de pacotes a depender do tamanho do *buffer* dos roteadores. Quando não especificado nas escalas dos gráficos ou nos rótulos das figuras, cada servidor envia 100MB de dados para o respectivo cliente, a conexão entre os nós e os roteadores é de 10Mbps com atraso de 50ms, e a conexão entre os roteadores é de 10Mbps com 75ms de atraso.

O gráfico na Figura 5 mostra como o tempo de transmissão varia em função do atraso no enlace entre os roteadores. É possível observar que o QUIC tem desempenho

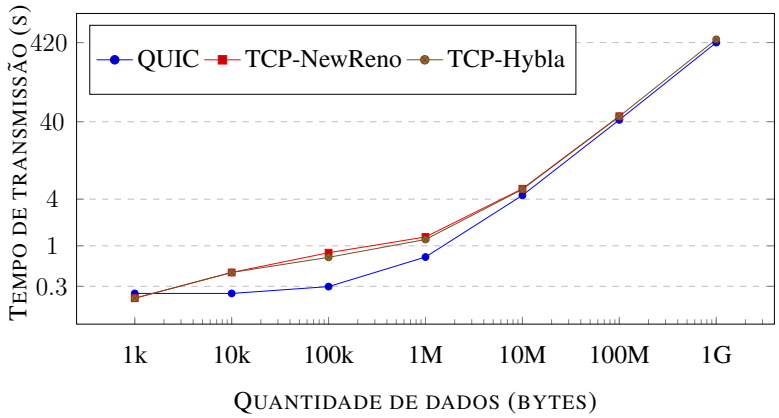

Figura 4. Variação da quantidade de dados enviados no cenário simples (sem perda de pacotes).

significativamente melhor que ambas as versões de TCP. Por conta da escala do gráfico, tem-se a impressão de que o QUIC não é afetado com esse tipo de variação, mas na realidade seu desempenho é pouco afetado com pequenas variações no atraso. Esses resultados evidenciam como o QUIC estima melhor o RTT e a largura de banda.

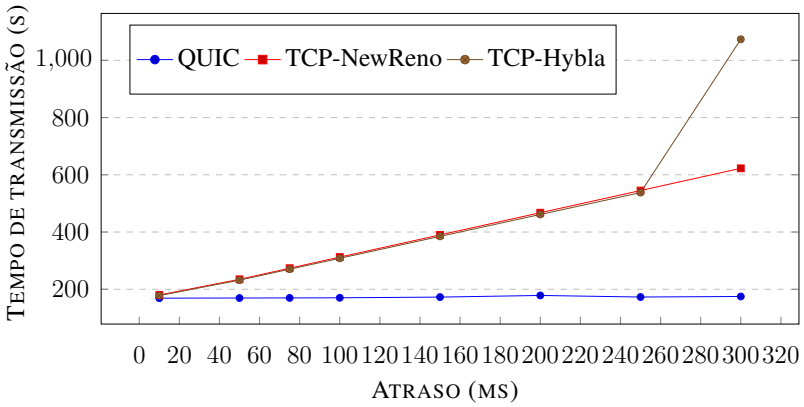

Figura 5. Variação do atraso no cenário Dumbbell (sem perda de pacotes).

5. Conclusões e trabalhos futuros

Neste artigo foi apresentado o `nsQUIC`, uma extensão ao simulador NS-3 que permite a realização de simulações com o protocolo QUIC de forma reprodutível. O artigo relatou a arquitetura e as principais funcionalidades do `nsQUIC` destacando as dificuldades e decisões tomadas na integração para que futuros trabalhos como esse sejam facilitados. Experimentos de simulação foram descritos e atestaram o funcionamento do `nsQUIC`. Como trabalho futuro, pretende-se recriar no simulador, com o máximo de semelhança, os mesmos cenários de trabalhos de medição a fim de se comparar os resultados obtidos.

6. Demonstração planejada para o SBRC

A demonstração que será realizada para mostrar as funcionalidades do `nsQUIC` será baseada na topologia Dumbbell apresentada na Subseção 4.2. Um código de simulação base

será apresentado e modificações serão feitas, de forma didática pelo apresentador, para mostrar a facilidade em simular mais nós na rede, bem como em incluir outras versões de TCP a fim de gerar, ao término da demonstração, gráficos que comparem os protocolos analisados. Como o apresentador levará o seu próprio computador, só será necessário um monitor de, no mínimo, 40 polegadas.

Referências

Caini, C. and Firrincieli, R. (2004). TCP Hybla: a TCP Enhancement for Heterogeneous Networks. *IJSCN*, 22(5):547–566.

Casetti, C., Gerla, M., Mascolo, S., Sanadidi, M. Y., and Wang, R. (2002). TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks. *Wireless Networks*, 8(5):467–479.

Floyd, S., Henderson, T., and Gurtov, A. (2004). The NewReno Modification to TCP's Fast Recovery Algorithm. https://tools.ietf.org/html/rfc3782. Acessado em 18 de Março de 2018.

Google (2017a). GitHub - google/proto-quic. https://github.com/google/proto-quic/. Acessado em 15 de Setembro de 2017.

Google (2017b). QUIC Transport Protocol. https://chromium.org/quic. Acessado em 15 de Setembro de 2017.

Google (2017c). The gn Meta-Build System. https://chromium.googlesource.com/chromium/src/tools/gn/. Acessado em 15 de Setembro de 2017.

Ha, S., Rhee, I., and Xu, L. (2008). CUBIC: a New TCP-friendly High-Speed TCP Variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74.

Kakhki, A. M., Jero, S., Choffnes, D., Nita-Rotaru, C., and Mislove, A. (2017). Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols. In *Proceedings of the IMC'17*, pages 290–303.

Langley, A., Riddoch, A., Wilk, A., Vicente, A., Krasic, C., Zhang, D., Yang, F., Kouranov, F., Swett, I., Iyengar, J., et al. (2017). The QUIC transport protocol: Design and Internet-scale deployment. In *Proceedings of the ACM SIGCOMM*, pages 183–196.

Ninja (2017). The Ninja Build System. https://ninja-build.org/. Acessado em 15 de Setembro de 2017.

Radhakrishnan, S., Cheng, Y., Chu, J., Jain, A., and Raghavan, B. (2011). TCP Fast Open. In *Proceedings of CoNEXT '11*, pages 21:1–21:12.

Riley, G. F. and Henderson, T. R. (2010). The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer.

Thomas Nagy (2017). Waf: the Meta Build System. https://waf.io/. Acessado em 15 de Setembro de 2017.

Wilk, A., Hamilton, R., and Swett, I. (2015). A QUIC Update on Google's Experimental Transport. https://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html. Acessado em 18 de Março de 2018.