Resumo - nsQUIC defesa

Renê Cardozo
rene.cardozo@usp.br

1 Introdução

Boa parte do tráfego da Internet é composto pelo streaming e download de vídeos, os quais em geral possuem baixa latência e alto consumo de banda. Experimentos científicos como LIGO (Laser Interferometer Gravitational-Wave Observatory) e LHC (Large Hadron Collider) geram petabytes de dados, que precisam ser distribuidos ao redor do mundo através de redes providas de enlaces de alta capacidade (da ordem de dezenas de Gbps), porém com alto atraso, uma vez que os dados percorrem distância continentais. Por fim, dispositivos IoT possuem pouca capacidade de processamento, sofrem alta taxa de perda de pacotes, uma vez que em geral estão conectados a redes móveis ou sem fio, tendo pouco atraso e taxa de transmissão limitada.

O protocolo mais utilizado para a camada de transporte, TCP, possui limitações quanto a seu desempenho nos instantes inciais da conexão. Também possui limitações referentes a conexões sem fio, dado que foi projetado em um momento histórico onde praticamente todas as conexões eram cabeadas, e, por isso, está otimizado para assumir que praticamente a única causa para a perda de pacotes é o congestionamento de redes, o que não é verdade para conexões sem fio. Este comportamento leva a uma redução na capacidade de transmissão sem necessidade, provocando uma subutilização da rede e oscilação na conexão.

O TCP slow-start pode demorar diversos RTT para atingir a capacidade máxima da banda, uma vez que baseia-se no incremento gradual da janela de congestionamento e não em uma estimativa prévia da capacidade da rede.

Especificações quanto ao tamanho máximo dos pacotes (64KB) pode interferir na utilização ótima de recursos da rede e no desempenho de transferências. O Window Scaling apenas afeta a RWIN, aumentando-a para 1GB e não o tamanho máximo de cada pacote. [1]

O intervalo de número de sequência de 32 bits pode apresentar limitações em redes de alta performance, uma vez que este limite de 4GB pode ser esgotado antes do primeiro ACK ser retornado.

Há também a limitação imposta por características da implementação da pilha do protocolo, a qual é atrelada ao sistema operacional, o que dificulta muito uma ampla e rápida alteração dos algoritmos que a compõem.

Iniciado em 2012, o desenvolvimento do protocolo QUIC (Quick UDP Internet Connections) alcançou ganhos significativos em relação ao TCP. Sua

implementação em espaço do usuário, acima do protocolo UDP, garante uma capacidade de atualização muito maior do que o TCP.

Contudo, os resultados divulgados pela Google, empresa que criou e desenvolve o protocolo, não possuem especificações a respeito das condições e configuração sob a qual estes experimentos foram realizados, eliminando a capacidade de reprodutibilidade destes. Sendo assim, cria-se a necessidade da criação de uma ferramenta capaz de analisar o desempenho do protocolo de maneira reprodutível e transparente.

2 Conceitos

3 Técnica de Avaliação

Três técnicas podem ser utilizadas para analisar o desempenho de sistemas computacionais:

- Modelagem Analítica: através da descrição matemática do objeto estudado, provendo uma solução precisa, porém com alta complexidade a medida que elementos interativos são adicionados ao sistema.
- Simulação: replicação dos eventos reais através de software, realizando suposições a cerca das configurações do sistema e pela criação de um modelo de descrição deste.
- Medição: obtenção de dados reais do sistema, considerando diversas variáveis aleatórias e sistemáticas não previstas pelos outros modelos. Contudo, este tipo de simulação pode ser extremamente custoso.

A simulação pode ser dividida em quatro categorias, para sistemas computacionais:

- Emulação: utiliza hardware ou firmware para imitar o comportamento do sistema real.
- Monte Carlo: através da geração de grandes amostras aleatórias podem ser obtidos resultados a cerca de sistemas complexos.
- Orientada a rastros: utiliza registros de eventos cronologicamente ordenados como parâmetros de entrada da simulação.
- Eventos discretos: Captura estados estáticos de uma sistema, em que cada mudança de estado é causada por um evento.

Será utilizado o método de simulação por eventos discretos para estudar o protocolo, bem como será realizada a sua validação comparando seu resultado com o de pesquisas anteriores.

4 Problemas com o TCP

4.1 Estabelecimento de conexão

A implementação padrão do protocolo TCP proíbe o envio de dados do servidor para o cliente antes que o three-way handshake esteja completo, buscando evitar ataques DoS. Alternativas como o T/TCP [2] visaram permitir o envio de requisições a partir do segmento SYN através do TCP Accelerated Open (TAO), o qual contava o número de conexões realizadas com o cliente e mantinha-as como parâmetro no servidor. Caso um segmento SYN possuísse o parâmetro de contagem maior do que aquele mantido pelo servidor, a requisição seria aprovada diretamente, burlando o handshake inicial. Os riscos desta implementação são enormes, uma vez que apenas induzindo um valor alto o suficiente para superar a contagem do servidor, um atacante poderia estabelecer uma false conexão.

Uma alternativa moderna para o handshake, proposta pela Google em 2014, é o TCP Fast Open (TFO) [3], o qual permite ao cliente solicitar o envio de um cookie pelo servidor em seu primeiro handshake, no qual constará o IP do cliente criptografado pelo servidor. Em conexões futuras, este cookie poderá ser enviado para o servidor junto as requisições iniciais pelo segmento SYN, permitindo que o servidor confirme a autenticidade do cliente através da comparação entre o cookie e o IP de envio criptografado ou desencriptação do cookie e comparação com o IP do requerente. Este mecanismo baseia o estabelecimento da conexão do protocolo QUIC.

A camada de segurança em conexões TCP é realizada pelo protocolo TLS, o qual exige duas RTT adicionais para que seu handshake próprio seja efetuado, o que adiciona uma alta latência a conexão. O uso de session identifiers a partir do TLS 1.2 [4] e session tickets [5], permitem que o cliente guarde informações já estabelecidas em um handshake anterior e envie-as ao servidor no primeiro ClientHello, reduzindo a latência para o estabelecimento da conexão em 1 RTT.

O TLS False Start [6], proposto pela Google em 2016, permite ainda que o handshake TLS seja realizado simultaneamente ao handshake TCP, reduzindo a necessidade de trocas para 1-RTT para conexões em que já exista contato prévio entre servidor e cliente.

4.2 Congestionamento

O algoritmo inicial do TCP para determinar a capacidade de transmissão é o slow-start, o qual baseia-se no incremento da janela de congestionamento de maneira exponencial. Esse aumento terminará quando uma perda for identificada por timeout do temportizador de um segmento, ou pelo recebimento de 3 ACKs duplicados.

No caso de ultrapassar o limite, o TCP altera o valor limite da janela (ssthresh) para metade do valor em que a perda ocorreu, passando então para um incremento linear de 1 MSS (Maximum Segment Size) para cada RTT, o qual é chamado de prevenção de congestionamento. Esse estado também realizará o reconhecimento da perda de pacotes através de timeout ou do recebimento de 3

ACKs duplicados.

Caso 3 ACKs duplicados sejam recebido o TCP interrompe seu estado de partida lenta/prevenção de congestionamento, e realiza a retransmissão do segmento que assume-se perdido, a qual pode ser efetuada antes do timeout do temporizador do segmento, sendo, portanto, uma retransmissão rápida. A diminuição da janela de congestionamento é realizada pela sua redução pela metade e adição de 3 MSS e o limite da janela é também reduzido pela metade do valor de quando a perda foi detectada. Há um incremento de 1 MSS para cada ACK duplicado recebido e, uma vez que o segmento antes perdido seja reconhecido em ACK, o valor da janela é atualizado para o seu limite. Este mecanismo é conhecido como recuperação rápida e após ele a conexão será passada/retornada para o estado de prevenção de congestionamento.

Em qualquer caso, um timeout retorna a conexão para o estado inicial de partida lenta.

O comportamento do TCP é dito AIMD, "Additive-Increase Multiplicative-Decrease".

O CUBIC, versão atual da pilha de protocolo TCP no Linux, utiliza uma função cúbica para ajustar a janela de congestionamento em termos do tempo passado desde a última perda de pacote. Na implementação, o valor máximo da janela de congestionamento é determinada pelo valor da janela durante a última perda de pacote. Em uma perda a janela é reduzida e o valor de janela máximo atualizado. Inicia-se então um aumento da janela de acordo com a parte côncava da função cúbica, e seu platô é definido como o valor máximo da janela. Quando uma nova perda de pacote ocorrer, repete-se o processo anterior, mas agora o incremento será realizado com base na parte convexa da função. Caso o valor da janela que seria ajustado pelo TCP for maior do que aquele previsto pelo CUBIC, o valor do TCP é utilizado (modo TCP). O CUBIC também possui a função de convergência rápida, a qual auxilia uma nova conexão a inserir-se na rede, uma vez que força uma redução maior do que a normal nos fluxos já existentes para que o novo tenha chance de se estabelecer.

O algoritmo BBR (Bottleneck Bandwidth and Round-Trip propagation time), publicado em 2016 e já implementado no Linux, não realiza seu controle de congestionamento por perda de pacotes, mas um histórico de RTTs recentes e da quantidade de dados enviada reconhecida, estimando o tempo de transmissão e o gargalo na largura de banda do caminho de transmissão, as quais são atualizadas a cada ACK recebido. O BBR também dá intervalos entre seus pacotes baseado nas estimativas do enlace de gargalo, para que não haja congestionamento. Para detectar aumentos na capacidade de transmissão, periodicamente o algoritmo realiza um RTT com um tempo menor do que o estimado. Caso tenha aumentado, a estimativa é atualizada e, caso não tenha, o intervalo de pacotes é aumentado.

Inicialmente, o BBR dobra a taxa de envio enquanto a taxa de pacotes cresce (Startup) até que a largura de banda do enlace de gargalo seja encontrada, após isso, a taxa de aumento da fase inicial é agora utilizada para drenar a fila formada no gargalo (Drain). Após atingir sua configuração estável, a conexão passará para o estado de sondagem, onde aumentará periodicamente o envio de

pacotes para testar um aumento na capacidade da conexão (ProbeBW). Quando a atualização da estimativa não ocorrer por um determinado período de tempo, a taxa de envio é diminuída em, pelo menos, 1 RTT, visando aliviar a fila de pacotes no gargalo. Esta última fase (probeRTT) é útil para conexões novas estabelecidas na rede.

O desempenho do BBR é melhor do que o CUBIC, portanto há tentativas de incorporá-lo ao QUIC.

5 QUIC

O protocolo QUIC foi projetado para atender às novas funcionalidades do HTTP/2, sendo que "na superfície, o QUIC é muito similar a TCP+TLS+HTTP/2 implementado usando UDP"[7]. Mescla portanto as camadas de transporte, de segurança e parte da camada de aplicação. O HTTP/2 permite multiplexação, isto é, a divisão da transferência de dados em diversos fluxos lógicos dentro de uma mesma conexão. Como o TCP possui um único fluxo de dados, o HTTP/2 necessita realizar o acesso de dados da camada de transporte (buffers do TCP) para conseguir coordenar a multiplexação. Com o QUIC, a multiplexação já é realizada pelo protocolo, sendo apenas necessária uma API de comunicação para o HTTP/2.

A ideia de permitir a paralelização do processamento de requisições pelo servidor iniciou-se com o HTTP/1.1, o qual introduziu o conceito de pipelining. Nesta versão do protocolo, permitiu-se que requisições a diferentes recursos fossem enviadas em paralelo para o servidor, garantindo um processamento mais rápido destas. Contudo, a incapacidade de identificação da correspondência entre os pacote e requisições do HTTP/1.1, faz com que as requisições apenas possam ser respondidas em ordem, portanto uma requisição anterior ainda não processada travará uma requisição nova que já foi terminada e teoricamente está pronta para ser enviada. Este problema é conhecido como head-of-line blocking.

O HTTP/2 propõe um novo mecanismo para a paralelização de requisições, no qual diferentes stream (fluxos de dados) são criadas e identificadas individualmente. Caso requisições sejam feitas em múltiplas streams, as respostas poderão ser identificadas como correspondentes a uma requisição de acordo com o ID da stream a que pertence. Portanto, uma vez que uma resposta esteja pronta em um servidor, este poderá despachá-la sem a necessidade de manter uma ordenação. Este mecanismo é chamado multiplexing.

Apesar de eficiente, o modelo implementado pelo HTTP/2 esbarra nas limitações do protocolo de uma camada abaixo. O TCP exige a entrega dos protocolos de maneira ordenada e possui apenas um fluxo para todos os dados e, em caso de perda de pacotes, travará todas os pacotes enviados anteriormente até que o pacote perdido seja recebido novamente. Portanto, o head-of-line blocking entre requisições retirado da camada de aplicação passa para a camada de transporte.

A característica unitária do fluxo de dados no TCP também apresenta problemas relacionados ao seu controle de congestionamento, o qual será aplicado

para todos os fluxos do HTTP/2 e, em caso de colisões, causará a diminuição da janela de congestionamento que afetará toda a transmissão.

O QUIC ainda implementa os mesmos mecanismos de segurança do TLS 1.3, os quais garantem uma segurança equivalente com menos round-trips do que seu sucessor TLS 1.2 e sem a necessidade de descriptografar pacotes em ordem.

Visando burlar os problemas de estabelecimento de conexão do UDP gerados por NATs, o QUIC não define uma conexão de acordo com IP e porta, mas através de um número pseudo aleatório e único para as conexões com o servidor, conhecido como Connection Identifier. Este número é enviado no primeiro pacote para o servidor e está presente em todos os outros, sendo válido por toda a conexão. Por fim, para superar a falta de notificação a respeito do encerramento de conexões por um NAT, o QUIC propõe que o cliente mande periodicamente mensagens de keep-alive, as quais em geral são enviadas pouco antes do tempo de expiração da tabela NAT ou a um passo constante, mas que deve ser maximizado, uma vez que pacotes pequenos enviados periodicamente podem representar grande parte do consumo de energia em dispositivos móveis. [8]

5.1 Estabelecimento de Conexão

Para estabelecer uma conexão segura rapidamente, o QUIC utiliza uma série de preferências enumeradas e estáticas no servidor, chamadas "server config". Estas preferências estão assinadas com a chave privada do servidor, portanto é possível garantir sua autenticidade.

Caso o cliente não tenha realizado contato com o servidor ainda, será enviada uma mensagem incompleta CHLO (Client Hello), a qual será rejeitada pelo servidor, que enviará o pacote REJ (Rejection), neste pacote estará disponível as provas de autenticidade e o server config do servidor. Uma vez que o cliente possua estas informações, é possível enviar uma mensagem diretamente criptografada com a chave pública do servidor. Após a primeira mensagem, o servidor poderá criar um valor efêmero e utilizá-lo para gerar chaves novas no cliente e servidor, o que permitirá uma chave diferente para cada conexão estabelecida (perfect forward secrecy). Sendo assim, se a chave de uma conexão for comprometida, não será possível interceptar outras conexões e descriptografá-las.

6 Controle de Conexão

O ACK no QUIC aponta o maior número de sequência recebido como no TCP e também são seguidos por uma lista de intervalos de números de sequência dos pacotes pendentes. A utilização de fluxos separados (antes presentes apenas na camada de aplicação) na camada de transporte, eliminam o problema do head-of-line blocking entre fluxos. O QUIC separa o controle de fluxo para fluxos individuais e para a conexão como um todo. Nos fluxos individuais, o destinatário é responsável por indicar uma janela máxima de dados que quer receber em cada fluxo e aumentá-lo ou diminuí-lo conforme necessário. Na

conexão como um todo existe a limitação do buffer agregado que um destino pode alocar, sendo também indicada uma janela máxima de dados, porém esta será o limite da soma que todos os fluxos podem possuir. No QUIC até mesmo as partes do cabeçalho que não estão encriptadas são autenticadas pelo servidor e todo o restante da conexão é criptografada, evitando a influência de agentes externos.

Semelhante a um mecanismo de RAID, o QUIC possuía um pacote que indica a paridade dos pacotes enviados de um determinado grupo, chamado FEC (Forward Error Correction). Este pacote é igual à aplicação da operação de XOR nos pacotes enviados anteriormente no grupo, podendo recuperar um pacote perdido do lado do cliente, sem a necessidade de executar uma retransmissão. Este mecanismo foi descontinuado, uma vez que só era possível recuperar um dos pacotes e, caso mais pacotes fossem perdidos, seria necessária a retransmissão de todos os pacotes já enviados. Ainda, o pacote FEC utilizava parte da banda dos pacotes contendo dados para serem transmitidos.

Procurando evitar a queda de desempenho abrupta do TCP causada pela diminuição da janela de congestionamento, o QUIC implementa o packet pacing, o qual realiza uma estimativa da largura de banda do caminho em que a conexão está estabelecida e busca reduzir ao máximo o número de retransmissões de pacotes. Ainda, o controle de congestionamento do QUIC pode ser realizado por uma reimplementação do TCP CUBIC ou do BBR. Cada pacote possuirá um número de sequência diferente, independentemente de ter sido fruto de retransmissão ou não, o que evitará a ambiguidade dos números presentes em retransmissões no TCP.

7 Análise de Desempenho do TCP

Quando comparou-se o CUBIC com seu antecessor (BIC) e TCP, utilizou-se os critérios de justiça entre fluxo de um mesmo protocolo com relação à utilização de banda, e o impacto em um fluxo TCP gerado pela variante analisada. Determinou-se que o CUIC apresenta uma maior capacidade de colaboração com os fluxos TCP já existentes em uma topologia simples. Contudo, quando utilizada a topologia de Dumbbell, o protocolo demonstrou-se injusto com relação as conexões TCP tradicionais.

8 nsQUIC

O NS-3 segue a arquitetura do TCP/IP, contendo funcionalidades básicas para simular redes de computadores, sendo que recursos adicionais podem ser adicionados por meio de módulos. Como o QUIC atua na camada de transporte, seria possível colocá-lo junto a outros protocolos semelhantes no NS-3, contudo, dadas as suas características de implementação em produção, foi escolhido implementá-lo como uma aplicação no simulador, sendo uma classe que controla quando enviar bytes e o que fazer quando receber estes bytes da camada

de transporte.

A reimplementação do protocolo dentro do NS-3 mostrou-se bastante complicada, uma vez que não há definição exata das especificações do protocolo. Foi então utilizado um fork de uma versão standalone do protocolo disponibilizada pela Google, a qual equivale à versão 41 do protocolo. Este fork possui uma grande base de código (mais de quatro mil arquivos) e utiliza ferramentas de compilação utilizadas pelo Google Chrome, Gn e Ninja. A utilização destas ferramentas apresenta um problema de integração com o NS-3, o qual utiliza a ferramenta Waf para auxiliar sua compilação.

A grande cadeia de dependências presente no código do QUIC gerou problemas para filtrar apenas o código necessário para ser integrado ao NS-3. Utilizando flags do g++ (-M e -MM) foi possível listar as dependências dos arquivos e realizar a importação automática de todos os headers e arquivos .cc de mesmo nome. Utilizando esta abordagem foi possível pegar apenas os arquivos referentes a implementação para Linux do protocolo. Em alguns casos, os headers não condiziam com o nome do arquivo .cc e tiveram que ser buscados manualmente. Ainda, o Gn gerava alguns arquivos automaticamente e outros arquivos não eram compilados da mesma maneira pelo Waf do que pelo Ninja, devido a opções de compilação únicas da ferramenta. Sendo assim, foram identificadas as opções necessárias para adaptar o código e gerados os arquivos que antes tinham build automático, bem como a ligação de todos os objetos corrigida.

A biblioteca standalone incorporada referia-se aos sockets da biblioteca sys/socket.h do sistema operacional, o que contrasta com o NS-3 que possui sockets próprios do simulador. As funções foram adaptadas para que somente utilizassem os sockets presentes no simulador.

O paradigma de tempo real presente na versão standalone do QUIC foi adaptada para o sistema baseado em eventos que ativam funções de callback no NS-3. Por fim, a parte de autenticação de certificados foi removida do módulo por conta da sua complexidade de implementação com diversas bibliotecas externas.

9 Experimentos

Foram utilizadas duas topologias: uma topologia simples, em que dois computadores são ligados diretamente e linearmente, supondo uma conexão ótima e sem perdas; e a topologia de Dumbbell, onde existem dois computadores como clientes (A_1 *e* A_2), os quais estão ligados a outros dois servidores (B_1 *e* B_2) por meio de dois roteadores (R_1 *e* R_2). Esta segunda topologia permite avaliar um cenário com congestionamento, uma vez que a inserção de roteadores implica na adição de buffers e um tempo de processamento para o direcionamento de pacotes, o que pode gerar a perda destes pacotes caso este buffer sofra overflow.

Na topologia de Dumbbell, a largura de banda e o atraso dos enlaces entre nós e roteadores são fixos em 1Gbps e 10ms, sendo então variados os parâmetros entre os dois roteadores como fator limitante do sistema. A simulação de perda de pacotes foi realizada de acordo com uma distribuição de probabilidade

uniforme que introduzia erros propositalmente com o objetivo de simular um cenário real, através da ferramenta RateErrorModel do NS-3.

Os parâmetros variados nos roteadores foram:

- Largura de banda: 5Mbps, 10Mbps, 50Mbps, 100Mbps, 500Mbps, 1Gbps
- Atraso: 10ms, 50ms, 100ms, 150ms, 200ms
- Taxa de erros: 0 (0%), 0.0001 (0.01%), 0.001 (0.1%), 0.01 (1%), 0.1 (10%)
- Bytes transmitidos: 10KB, 100KB, 200KB, 500KB, 1000KB, 10000KB e 210000KB.

As variações do protocolo TCP às quais o módulo QUIC foi comparada foram simuladas utilizando as aplicações BulkSend e PacketSink do NS-3, buscando igualar a um comportamento de cliente servidor. As medições de desempenho foram feitas através do atributo timeLastRxPacket do módulo flow monitor do NS-3, o qual indica o tempo em que o último pacote da transferência foi recebido.

Apesar da versão standalone da Google ser considerada não adequada para a utilização em produção devido a uma baixa janela de congestionamento e a não capacidade para processamento de requisições simultâneas. O fork utilizado corrigiu os problemas relacionados a esta janela, e o modelo de simulação utilizado com apenas um cliente, portanto não há requisições simultâneas.

10 Resultados

Em uma topologia simples, o QUIC apresenta um desempenho superior a todas as diferentes variações do TCP, sendo essa diferença de desempenho aumentada conforme aumentou-se o tamanho dos dados transmitidos. O QUIC apresenta um comportamento de curva similar o CUBIC, uma vez que implementa o mesmo algoritmo para controlar sua janela de congestionamento, contudo sua implementação é mais agressiva com o amento da janela, bem como opera mais próximo ao limite do enlace, portanto apresenta um desempenho melhor do que o CUBIC no TCP.

Sua característica agressiva de aumento da janela de congestionamento propicia que o QUIC realize um melhor aproveitamento da banda em transmissões mais curtas, alcançando patamares mais altos em menos tempo do que algoritmos tradicionais do TCP. A reação a erros do QUIC mostra-se muito mais rápida do que o TCP quando tirada a média de 30 simulações, seu tempo de transmissão para taxas de erro em torno de 1% são 10 vezes menores. Isso pode ser explicado pela melhor capacidade de recuperação do QUIC e melhor estimativa e aproveitamento da banda em casos de erro.

As características dos resultados obtidos na topologia simples para uma transmissão com parâmetros constantes, variando a largura de banda, atraso e taxa de perda de pacotes (mantendo todos os outros parâmetros fixos) é preservada na topologia de Dumbbell, tendo comportamento similar às curvas do QUIC e TCP CUBIC na topologia simples.

Quando variados o número de nós da topologia de Dumbbell, o QUIC ainda apresentou uma maior performance na transmissão, independente do número de nós, os quais foram variados de 4 a 100.

Para protocolos iguais compartilhando um meio de conexão física, o índice de justiça é próximo de um, o que significa que os fluxos não prejudicam uns aos outros. Contudo, este cenário se altera quando TCP CUBIC e QUIC competem pelo meio, a característica agressiva de incremento da janela de congestionamento pelo QUIC faz com que este alcance o limite da largura de banda muito mais rápido do que o TCP, tendo uma taxa média de recebimento de dados 3,6 vezes maior do que o TCP para 100KB.

Para realizar a validação do experimento, foi utilizado um trabalho desenvolvido em 2017, "Taking a Long Look at QUIC", o qual avalia o desempenho do protocolo através de medições de situações reais. Neste trabalho é afirmado que para ambientes desktop o QUIC é melhor que o TCP CUBIC em quase todos os cenários, o QUIC é justo com fluxos do mesmo protocolo, contudo é injusto com fluxos TCP CUBIC, consumindo mais do dobro da largura de banda apropriada.

Portanto, os resultados obtidos pelo nsQUIC são consistentes com aqueles encontrados em medições reais.

11 Limitações

A falta da implementação da conexão em 0-RTT pelo módulo pode subestimar a superioridade dos resultados obtidos pelo QUIC com relação as variações de TCP. As atualizações recentes da Google com relação ao algoritmo CUBIC utilizado pelo QUIC poderiam também aumentar o desempenho do protocolo em determinadas situações.

12 Referências

Artigo Original: CAMARINHA, Diego de Araujo Martinez. Análise de desempenho do nsQUIC: um módulo para simulação do protocolo QUIC. 2018. Dissertação (Mestrado em Ciência da Computação) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2018. doi:10.11606/D.45.2018.tde-16102018-181616. Acesso em: 2020-06-10.

1 - RFC 1323 (TCP Extensions for High Performance) https://www.ietf.org/rfc/rfc1323.txt

2 - RFC 1644 (TCP Extensions for Transactions Functional Specification) https://tools.ietf.org/html/rfc1644

3 - RFC 7413 (TCP Fast Open) https://tools.ietf.org/html/rfc7413

4 - RFC 5246 (The Transport Layer Security (TLS) Protocol Version 1.2) https://tools.ietf.org/html/rfc5246

5 - RFC 5077 (Transport Layer Security (TLS) Session Resumption without Server-Side State) https://www.ietf.org/rfc/rfc5077.txt

6 - RFC 7918 (Transport Layer Security (TLS) False Start) https://tools.ietf.org/html/rfc7918

7 - QUIC, a multiplexed stream transport over UDP https://www.chromium.org/quic

8 - Qian, Feng & Wang, Zhaoguang & Gerber, Alexandre & Mao, Zhuoqing & Sen, Subhabrata & Spatscheck, Oliver. (2011). Profiling Resource Usage for Mobile Applications: a Cross-layer Approach. 321-334. 10.1145/1999995.2000026.