Trabalho Prático 2: Controle de Congestionamento

Nelson G. Prates Junior

1 de novembro de 2018

1 Exercício A

Varie o tamanho da janela fixa editando controller.cc para ver o que acontece. Faça um gráfico 2D de taxa de transfência versus o atraso no sinal com percentil de 95 à medida que você altera esse valor. Qual é o melhor tamanho de janela única que você pode encontrar para maximizar a potência (taxa de transferência/atraso)? Quão repetíveis são as medições feitas com o mesmo tamanho de janela em várias execuções?

Para a realização dos testes utilizei um script que extrai os resultados de cada teste (pasta "tp2A/resultados"). O script realiza 33 testes onde os valores atribuídos para as janelas seguem um somatório de índice: 3, índice inicial: 3 e final: 99. Os resultados são mostrados na tabela 1. Nota-se que nos testes realizados, os melhores resultados estão entre os tamanhos de janela 12 e 15. Então realizei 25 novos testes com cada um dos valores as quais mantiveram uma média de 11,79 e 11,66 respectivamente. Os scripts e códigos modificados são encontrados na pasta *"Tp2A"*

Os gráficos referentes às Figuras 1 e 2, demonstram claramente que com o aumento na janela, a utilização da largura de banda também aumenta, no entanto, o atraso é maior. O que é esperado, pois um fluxo muito grande de pacotes é a principal causa de atrasos de fila. Outro ponto importante para se notar, é o aumento significativo da potência nos valores mais baixos da janela quando comparados com os valores mais altos. Acredito que como a principal motivação para o aumento do atraso é a fila de pacotes, quando a rede não ultrapassa a capacidade do receptor de tratar e confirmar o recebimento, o desempenho é uma consequência. Ao contrário de quando a janela está definida com valores elevados, onde o consumo da largura de banda é maior, porém não há capacidade de tratar e responder as requisições em tempo hábil (o que gera atraso de fila).

Window Size	Power Score	Throughput (Mb/s)	Delay(ms)
3	6,40	0,65	102
6	10,60	1,24	117
9	12,51	1,78	142
12	12,84	2,25	175
15	12,53	2,67	213
18	12,15	3,04	250
21	11,58	3,36	290
24	10,97	3,63	331
27	10,56	3,87	366
30	10,14	4,06	400
33	9,80	4,22	431
36	9,40	4,36	464
39	9,08	4,48	493
42	8,73	4,57	524
45	8,36	4,66	557
48	8,03	4,73	589
51	7,75	4,79	618
54	7,53	4,84	642
57	7,26	4,88	672
60	6,89	4,91	712
63	6,63	4,93	743
66	6,41	4,94	772
69	6,21	4,96	798
72	6,00	4,97	828
75	5,81	4,98	856
78	5,68	4,98	877
81	5,51	4,99	907
84	5,39	5	927
87	5,27	5	949
90	5,15	5,01	972
93	5,05	5,01	993
96	4,93	5,01	1017
99	4,80	5,02	1045

Tabela 1: Tabela referente a os resultados dos testes realizados com janela fixa

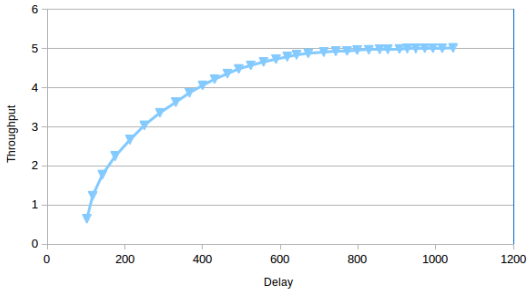

Figura 1: Throughput x Delay

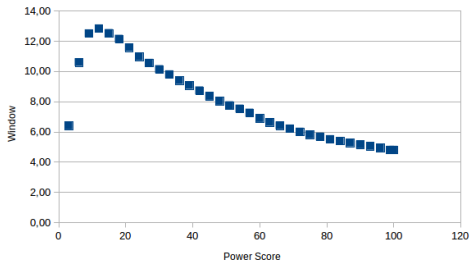

Figura 2: Window size x Power

2 Exercício B

Implemente um esquema AIMD simples, semelhante á fase de prevenção de congestionamento do TCP. Quão bem isso funciona? Quais constantes você escolheu?

O esquema AIMD (***A**dditive **I**ncrease **M**ultiplicative **D**ecrease*) tem sua funcionalidade baseada em dois parâmetros, α e β, as quais são responsáveis por controlar o tamanho da janela. Então, a janela aumenta conforme um somatório onde com $ndice = \alpha$ ($\sum_{k=\alpha}^{threshold} Window$). Caso haja a identificação de um congestionamento ou atinja um determinado limite de crescimento, ela diminui bruscamente através da divisão da janela por β ($Window = Window/\beta$). A identificação do congestionamento neste contexto é realizada através de um threshold predefinido, ele significa um limite, que em milissegundos, representa o tempo de espera de uma resposta a partir o envio de um datagrama.

Para a implementação do AIMD modifiquei o arquivo controller.cc. As modificações incluem as seguintes alterações:

- Inclusão de variáveis privadas:
  - AI = α;
  - MD = β;
  - boundary_window = limite de crescimento da janela;
  - time_out_ms = limite de tempo de resposta;
- Inclusão da função $voidset\_window_size(boolAIMD)$;
- Modificação nas funções *window_ size, datagram_ was_ send e ack_ received*;

A função *void set_ window_ size(bool AIMD)* recebe como parâmetro uma variável booleana onde se falso, a janela é incrementada; se não, a janela é decrementada. Ambos seguindo as regras mencionadas a cima. As modificações realizadas nas funções compreendem a remoção do valor fixo da janela, a inclusão de uma verificação de time out: se time out a janela é decrementada, por fim, a inclusão dos verificadores para incremento da janela dentro dos limites estipulados.

Os testes iniciais foram bem-sucedidos, podemos ver o gráfico (Figura 3) demonstrando o funcionamento da implementação do AIMD. Note que o tamanho da janela aumenta (neste caso $\alpha = 0.1$) até atingir o limite (*boundary_ window = 12*); em seguida, o tamanho da janela diminui bruscamente pela metade ($\beta = 2$), a variação acontece repetidamente. O gráfico representado na Figura 4 demonstra o momento em que o primeiro congestionamento foi identificado através do time out (neste caso $time\_out\_ms = 80$). Note a redução brusca no tamanho da janela, também, logo ao fim, a recuperação do crescimento da janela. Ambos os gráficos (Figuras 3 e 4) representam uma única execução da

simulação para a avaliação do AIMD. Para a criação dos gráficos incluí uma função que extrai logs em tempo de execução. Todos os scripts e os códigos modificados são encontrados na pasta *"tp2B"*.

Após a realização de diversos testes manuais, notei que os melhores resultados herdam características da janela fixa, com janela limitada entre 12 e 15. Também que os valores atribuídos a α mantém os melhores resultados conforme $0 < alpha < 1$. Então programei um script para rodar 20 testes com janela limitada em 12 ($boundary\_window = 12$); α entre 0.1 e 0.20; β entre 1 a 20. Os resultados são mostrados na Tabela 2. Note que os melhores resultados então entre os menores valores de β. Seguindo este parâmetro realizei novos testes com $\alpha = 0.1$ e $\beta = 2$ as quais atingiu $Power = 19.18$, sendo o melhor resultado obtido até o momento.

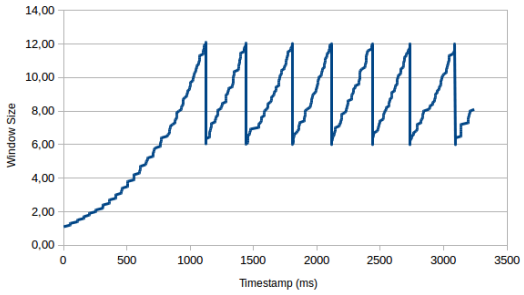

Figura 3: AIMD em funcionamento padrão

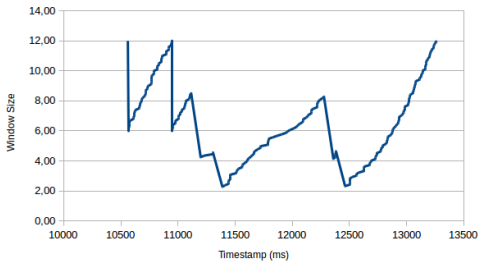

Figura 4: AIMD sob congestionamento

3 Exercício C

Tente diferentes abordagens e trabalhe para maximizar a potência (compare com os resultados dos demais colegas da sala e tente melhorar). Em seu relatório, explique sua abordagem, incluindo as decisões importantes que você teve que tomar e como você as fez. Inclua gráficos ilustrativos com os resultados.

Para encontrar uma abordagem nova para melhorar os resultados, primeiramente recorri a literatura sugerida pela professora. No entanto, apesar de ajudarem muito conceitualmente, os trabalhos vão muito além do que simplesmente o ajuste da janela de congestionamento, como o Janus Zhou et al. [2018] que interage com as aplicações. Então devido ao curto espaço de tempo que eu tinha para aprender como fazer e reproduzir optei por procurar soluções mais simples. Ao pesquisar nas referências dos trabalhos notei que o Nimbus Goyal et al. implementa um controle de Delay (um dos principais fatores da avaliação) inspirado no trabalho TIMELY Mittal et al. [2015]. As quais ao ler, identifiquei (graças as leituras anteriores) um modelo mais simples de controle da janela de congestionamento e semelhante ao AIMD. Outro ponto positivo é que os autores apresentaram um pseudo-algoritmo bastante esclarecedor sobre as variáveis e parâmetros, o que diante das minhas habilidades simplificaria o desenvolvimento e consequentemente levaria menos tempo para o desenvolvimento.

AI	MD	Throughput	Delay	Power
0.1	19	1182	82000	14,41
0.2	18	1374	96000	14,31
0.3	17	1480	103000	14,37
0.4	16	1604	114000	14,07
0.5	15	1548	110000	14,07
0.6	14	1748	115000	15,20
0.7	13	1702	119000	14,30
0.8	12	1811	126000	14,38
0.9	11	1919	125000	15,35
0.1	10	1190	80000	14,87
0.11	9	1223	82000	14,91
0.12	8	1254	82000	15,29
0.13	7	1320	86000	15,34
0.14	6	1380	89000	15,50
0.15	5	1448	93000	15,57
0.16	4	1508	90000	16,76
0.17	3	1680	98000	17,14
0.18	2	1864	113000	16,49
0.19	1	2386	160000	14,92

Tabela 2: Resultados dos testes realizados para definição dos valores α e β referentes ao AIMD

A principal proposta do TIMELY Mittal et al. [2015] é controlar o congestionamento causado pelos atrasos de fila através do RTT em *datacenters*. Para isso, primeiramente eles provam que o RTT é correlacionado com o enfileiramento. Em seguida, apresentam o **TIMELY** (**T**ransport **I**nformed by **ME**asurement of **L**atenc**Y**), um sistema de controle de congestionamento que além de usarem a taxa de variação do RTT, usam um cálculo de gradiente (grau de variação do atraso) para prever o início do congestionamento e, portanto, manter os atrasos baixos, ao mesmo tempo que fornecem alto rendimento. Por fim, realizam testes em um sistema de mensagens de desvio pertencente a um grande datacenter, as quais provam a eficiência do sistema. A Figura 5 de demonstra um pseudo-algoritmo que representa o a implementação do TIMELY.

Para implementar o TIMELY primeiramente tentei ajusta a classe de timestamp. Pois os autores enfatizam que a as capturas em microssegundos é uma das principais vantagens na efetividade do algoritmo. No entanto, todo o código disponibilizado para a realização da atividade é programado para funcionar com as unidades de tempo em milissegundos. Então, diante do não sucesso na tentativa de implementar as capturas em microssegundos, resolvi implementar o algoritmo utilizando a unidade de tempo em milissegundos (e funcionou). Nas primeiras execuções percebi através do *debug*, que as vezes a janela (*rate*) chegava ao valor 0, trazendo a necessidade de incluir um limite mínimo para a

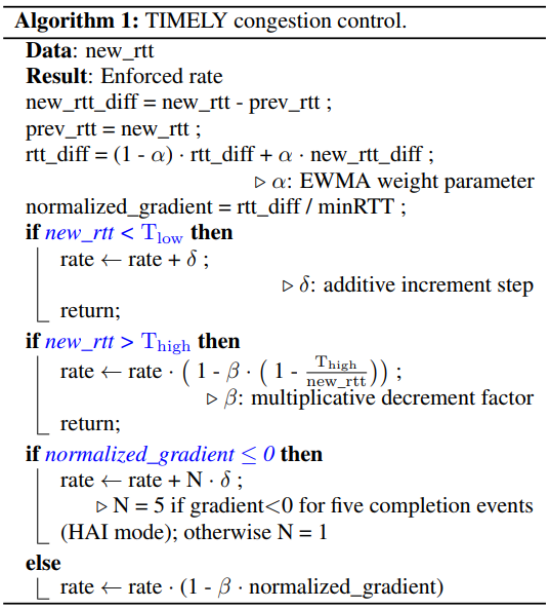

Figura 5: Algoritmo referente a o TIMELY

janela.

O algoritmo define um tamanho de janela para cada conexão e atualiza RTT a cada transmissão, ou seja, ao receber o ack. Os cálculos realizados permitem uma observação mais detalhada do rtt ao longo do tempo. O algoritmo é composto por quatro estados que definem como será a manipulação da janela. Os estados alternam conforme as constantes que determinam um limite máximo e mínimo para a janela, também são realizados cálculos para definir a taxa de variação do rtt (gradiente). O primeiro estado é iniciado enquanto o rtt é menor que o valor mínimo estipulado (T_LOW), então o tamanho da janela é aumentado de forma agressiva (*Additive Increase*). O segundo estado identifica que o rtt atingiu o seu limite máximo e reduz a janela de transmissão (*Multiplicative Decrease*). O terceiro estado é controlado pelo gradiente de atraso, caso o gradiente for negativo, significa que a fila está conseguindo ligar com o tráfego, então a janela pode aumentar (*Hyperative Increase*). Já o quarto e último estado reduz o tamanho da janela se o gradiente for positivo. Os dois primeiros são em casos extremos, quando o atraso é muito alto, ou muito baixo. Os dois últimos respeitam uma forma mais gradual de modificação da janela. A Figura 6 do artigo Mittal et al. [2015] demonstra as áreas de atuação de cada estado.

Conforme cada estado um modelo de acréscimo ou decréscimo da janela é definido. Eles respeitam os valores atribuídos a β e δ. Para calcular o gradiente de atraso, o TIMELY calcula a diferença entre duas amostras de RTT consecutivas. Normaliza essa diferença dividindo-a pelo RTT mínimo, obtendo uma quantidade sem dimensões. Usa também um valor fixo (α) que representa o atraso de

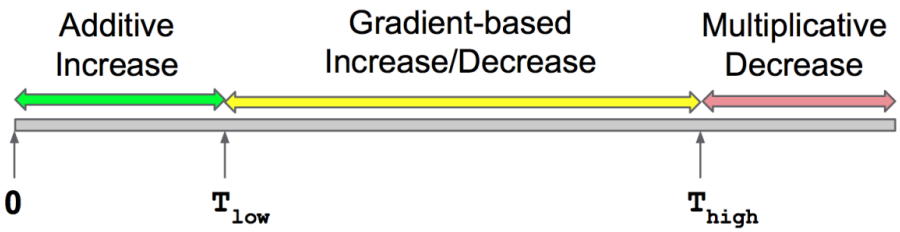

Figura 6: Zona do gradiente com limite de RTT baixo e alto

propagação. Por fim, passa o resultado através de um filtro EWMA. Esse filtro permite detectar a tendência geral de subida e descida na fila, enquanto ignora pequenas mudanças na fila que não indicavam congestionamento.

Para definir os valores das constantes (*"T_LOW", "T_HIGH","MIN_RTT",* "α, β *e* δ") realizei pesquisas sobre implementações do TIMELY e rodei diversos testes manuais. Para definir os tamanhos da janela máximos e mínimos me segui os valores sugeridos no artigo TIMELY Mittal et al. [2015] onde além de definir os valores, sugere que estes não são muito influentes nos resultados finais. Os autores definem $T\_LOW = 50$, $T\_HIGH = 500$ $MIN\_RTT = 20$. No entanto os valores atribuídos para "α e β" no artigo são calculados em Mbps. Então tentei calcular algum valor equivalente e realizai alguns testes infelizes. Como o autor fixa o valor de 0.8 para δ, testei o código com $\delta = 0.8$, $\alpha = 1$ e $\beta = 1$ e, com sorte consegui, o melhor resultado até então $Power = 25.13$. Acompanhando os resultados, percebi que o rtt na maior parte da simulação se mantém a cima de 50 e o cálculo da normalização foi definido pelo autor como 20. Ao testar com $MIN\_RTT = 50$ os resultados melhoraram muito, atingindo um o valor de $Power = 38.02$.

Com o objetivo de aprimorar os resultados, desenvolvi um script as quais testava os valores de α, β e δ. Os valores utilizados em cada teste realizado de dois dos parâmetros eram fixados em 1 enquanto o terceiro era definido seguindo um somatório de 0.1 até 0.9. As quais os resultados são representados pela Tabela 3. Seguindo os melhores valores representados na tabela realizei uma série de testes manuais (os logs estão na pasta "tp2/logs"). As quais cheguei a conclusões diferentes das iniciais. Dentre os trinta testes realizados o melhor resultado atingido foi 42,64.

Alpha	Beta	Delta	Power
0.1	1	1	17547
0.2	1	1	22218
0.3	1	1	24696
0.4	1	1	28474
0.5	1	1	29027
0.6	1	1	31422
0.7	1	1	31862
0.8	1	1	32589
0.9	1	1	32407
1	0.1	1	12617
1	0.2	1	16394
1	0.3	1	18926
1	0.4	1	22916
1	0.5	1	26254
1	0.6	1	29789
1	0.7	1	31370
1	0.8	1	33079
1	0.9	1	33107
1	1	0.1	9943
1	1	0.2	18642
1	1	0.3	26635
1	1	0.4	32043
1	1	0.5	35488
1	1	0.6	35422
1	1	0.7	37802
1	1	0.8	34281
1	1	0.9	35542

Tabela 3: Resultados dos testes realizados para a definição dos melhores parâmetros para o TIMELY

Alpha	Beta	Delta	T_LOW	T_HIGH	MIN_RTT	Throughput	Delay	Power
0.8	0.3	0.25	60,00	150,00	50,00	4121,53	98866,67	41,68

Tabela 4: Tabela referente a os parâmetros utilizados e média dos resultados de 30 repetições

Referências

Fan Zhou, David Choffnes, and Kaushik Roy Chowdhury. Janus: A multi-tcp framework for application-aware optimization in mobile networks. *IEEE Transactions on Mobile Computing*, 2018.

Prateesh Goyal, Akshay Narayan, Frank Cangialosi, Deepti Raghavan, Srinivas

Narayana, Mohammad Alizadeh, and Hari Balakrishnan. Elasticity detection: A building block for internet congestion control.

Radhika Mittal, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, David Zats, et al. Timely: Rtt-based congestion control for the datacenter. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 537–550. ACM, 2015.