Computer Webs

Кратенькое содержание

Лектор: Пашков В.Н.

Кратенькое содержание набросали студенты специализации РС

НИУ ВШЭ, 2019-2020

GitHub: github.com/distsys-hse/computer-networks

Содержание

1 Адресация в сети

Как давно вы переводили из десятичной системы в двоичную?

Сейчас мы немного напомним, как происходит адресация в сетях.

Во-первых у любого ip адреса (в идеальном мире) есть адрес сети и маска. Сам по себе ip-адрес - это полный путь до компьютера. Маска помогает определить адрес подсети и номер компьютера в этой сети.

Маска - это ip из n последовательных единиц.

Обычно маска записывается, как суффикс к ip. Например 172.16.139.46/20 говорит о том, что в маске 20 единиц. (255.255.240.0)

Из этого можно получить несколько

1. **Адрес сети** получается побитовым and ip и маски
2. **Первый адрес в сети** - это адрес сети + 1
3. **Широковещательный адрес** - это *адрес сети* **or** ($\sim mask$)
4. **Последний адрес в сети** - это *Широковещательный адрес* - 1
5. **Следующая сеть** - это *Широковещательный адрес* + 1

Скрипт *adress.py*, который умеет это вычислять можно найти на Github

2 Скользящее окно

Задача: хост A отправляет хосту B сообщение, состоящее из n пакетов. Порядок отправки пакетов должен соответствовать порядку получения пакетов. Необходимо доставить все n пакетов при условии, что они могут теряться.

Идея stop and wait: A отправляет пакет и ждёт от B подтверждение получения. Если время ожидания больше определённого $timeout$, то A считает, что пакет потерялся и отправляет его повторно. Если же A дождался подтверждения, то он отправляет следующий пакет.

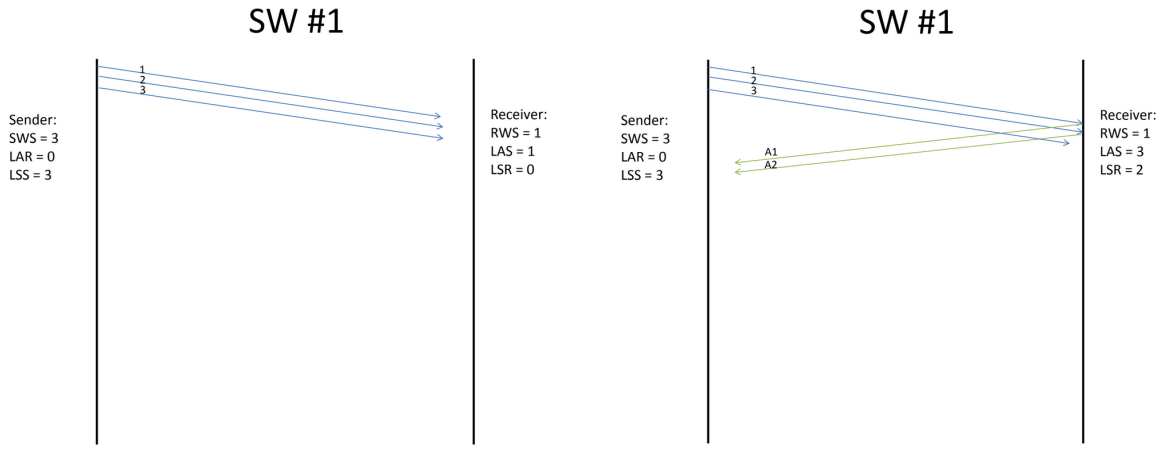
Идея скользящего окна: A отправляет сразу несколько (SWS) пакетов, то есть наше окно на первом шаге выглядит вот так $[1, 2, \ldots, SWS]$.

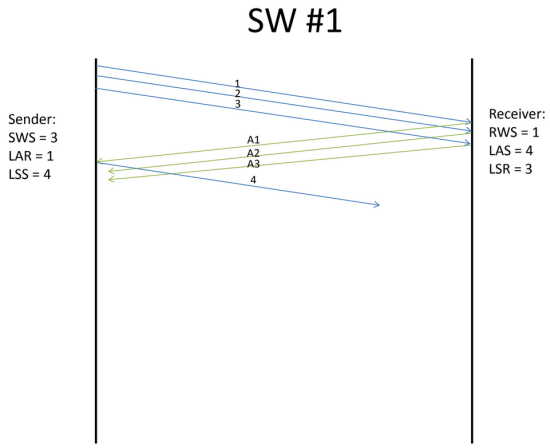
Хост B действует следующим образом: если приходит пакет с номером k и пакет с номером $k-1$ уже был принят, то пакет k принимается, иначе не принимается. Далее B отправляет номер **последнего принятого пакета**.

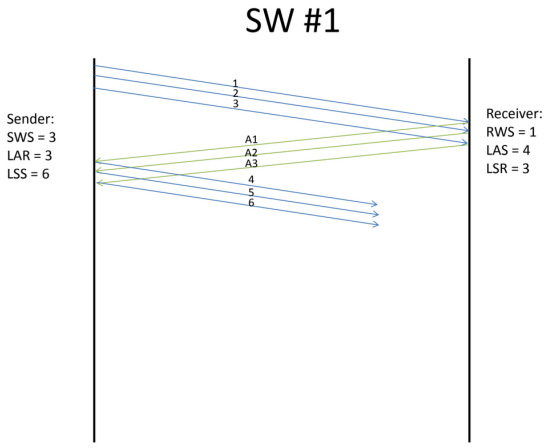
Хост A хранит предыдущее значение последнего принятого хостом B пакета (LAR, Last Acknowledgement Received) и номер последнего посланного пакета (LSS, Last Segment Sent). A ждёт подтверждения о получении пакетов. Если подтверждение не приходит через определённое время ($timeout$), то A отправляет этот пакет повторно.

Если A получает от B подтверждение с номером последнего пакета $N > LAR$, то окно сдвигается на $LAR - N$, то есть мы досылаем пакеты с номерами от $LSS + 1$ до $LSS + LAR - N$, в итоге $LSS = LSS + LAR - N$. Если же $N = LAR$, мы начинаем отправку пакетов повторно (отправляем по очереди все пакеты текущего окна).

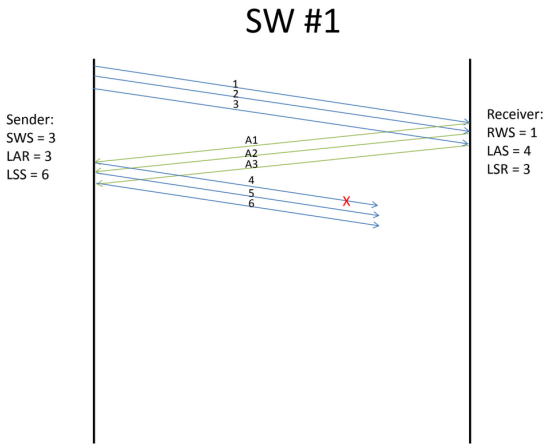
Поведение при отсутствии потерь:

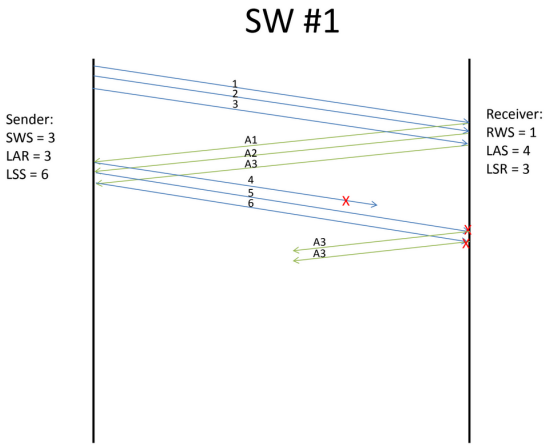


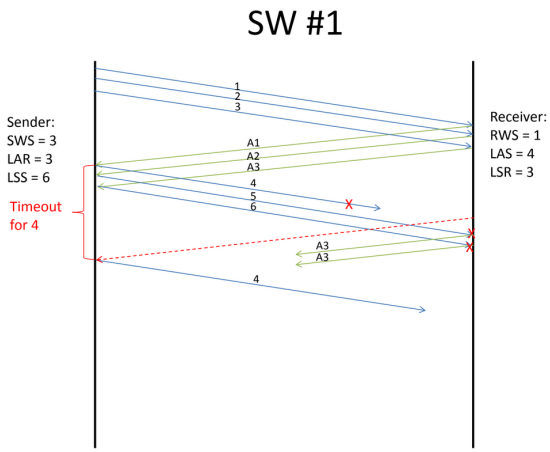


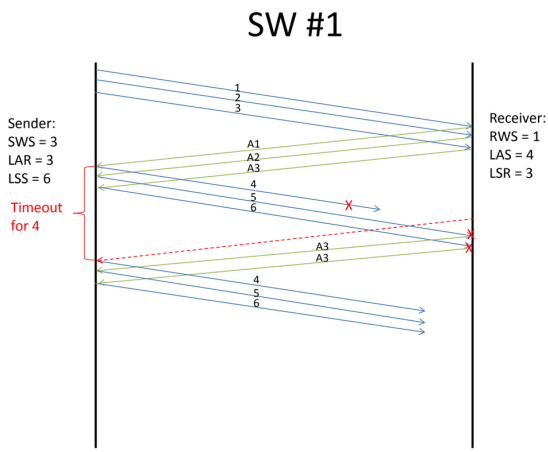


Поведение в случае потери пакета:









Для лучшего понимания предлагаю посмотреть видео. Предложенный там алгоритм несколько отличается от того, что нам рассказывали, но в целом его идея такая же. Там сигналом к тому, что нужно заново посылать пакеты, является только время ожидания, без акцепта со старым значением прянятого пакета. То есть, ещё раз, если прилетает аццепт на уже аццептнутый пакет, окно начинает отсылаться заново.

2.1 Пример 1

Пример есть в чатике. Искать по хештегу #ОляШарит

3 CRC коды.

CRC – Cyclic Redundancy Check С помощью CRC мы можем вычислять контрольные суммы. Основная идея - представление сообщения в виде многочлена с коэффициентами 1 и с такими степенями, что в этой позиции стоит единица.

Например, сообщение 1100010 будет записано как $x^6 + x^5 + x$

Сообщение мы обозначаем как $M(x)$

Вместе с этим существует **порождающий полином**, который записывается как $G(x)$

В итоге мы хотим найти $T(x)$ - **остаток** от ксорного деления одного многочлена на другой

Что есть ксорное деление? Возьмём многочлен $M(x)$ и домножим его на степень многочлена $G(x)$. Получится $\hat{M}(x)$ Возьмём $\hat{M}(x)$ и $G(x)$ в бинарном виде. Первое - делимое, второе - делитель. Начнём стандартное деление. Но вместо деления мы будем делать *xor*

В итоге получится какой-то остаток. То, что получится в результате деления нам всё равно, важет только остаток. Этот остаток и называется **checksum**.

Многочлены $G(x)$ разделены на разные классы в зависимости от наибольшей степени. Например, многочлен $x^6 + x^5 + x$ принадлежит классу CRC-6, а многочлен $x^{13} + x^8 + x^6 + x^2 + 1$ - к классу CRC-13.

Порождающий полином должен быть дан в условии задачи.

Скрипт, который умеет считать это называется *checksum.py*. Его тоже можно найти на GitHub.

4 Min-Max-распределение

Когда у нас есть сеть и хочется как-то распределить количество информации, которое передаётся в сети, справедливо, то одним из очевиднейших решений является *MinMaxDistribution*.

Распределение max-min справедливо если нельзя увеличить скорость какого-нибудь потока, не понизив скорости другого, меньшего потока

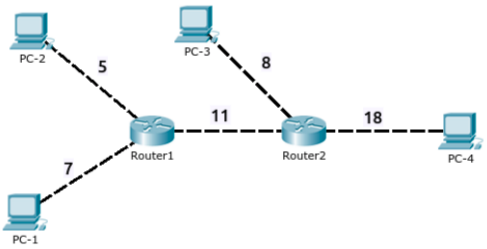
Давайте запустим наш итерационный алгоритм - будем каждому активному потоку добавлять одно и то же значение, пока какой-либо канал не переполнится. Смотрим, какие потоки были через переполненный канал и переводим их в пассивные - увеличить мы их уже не сможем. Продолжаем до тех пор, пока все потоки не станут пассивными.

Разберём на примерах.

4.1 Пример 1.

Max-min-справедливое распределение #1

Дана топология:



Для линков цифрами указаны пропускные способности в МБит/с. Потоки данных идут от PC-1, PC-2 и PC-3 к PC-4. Построить max-min-справедливое распределение потоков данных.

	Flows			Links				
δ	PC1 → PC4	PC2 → PC4	PC3 → PC4	PC1-R1	PC2-R1	R1-R2	PC3-R2	R2-PC4
5	5	5*	5	5/7	**5/5**	10/11	5/8	15/18
1	6*		6	6/7		**11/11**	6/8	17/18
1			7*	—||—			7/8	**18/18**

Оформлять нужно именно такой табличкой. Разберём, что произошло.

На первой итерации мы пихнули поток 5. Забилось ребро PC2 → R1. Поток 1 объявляется пассивным со значением 5. На второй итерации пихнули поток 1. Забилось ребро R1 → R2. Поток 2 объявляется пассивным со значением 6. Увеличиваем ещё на один оставшиеся и забивается ребро R2 → PC4. Последний поток становится пассивным со значением 7.

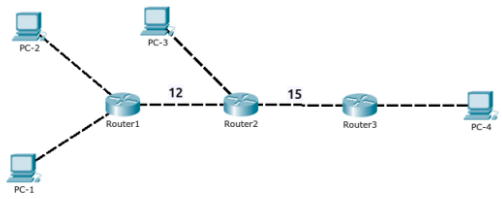
Ответ: Распределение потоков: 6, 5, 7.

Для удобства потоки и линки можно перенумеровывать, но можно оставить и так. Для ясности.

4.2 Пример 2.

Max-min-справедливое распределение #3

Дана топология:



Для линков цифрами указаны пропускные способности в МБит/с. PC-1 передаёт данные PC-4, PC-2 –> PC-3, PC-3 –> PC-4. Считаем, что линки между PC и роутерами имеют достаточную пропускную способность. Построить max-min-справедливое распределение потоков данных.

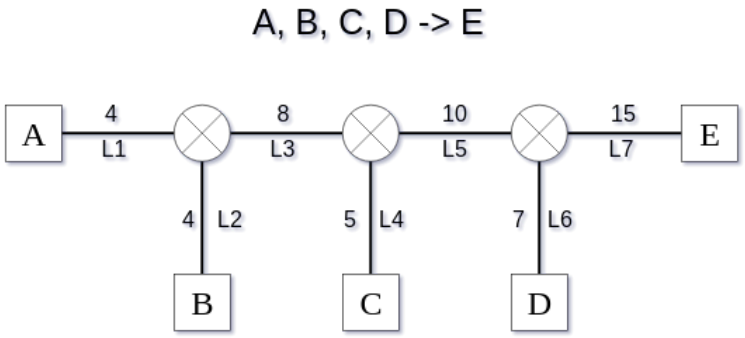
- Здесь линки между PC и роутерами можно не нумеровать

	Flows			Links	
δ	PC1 → PC4	PC2 → PC3	PC3 → PC4	R1-R2	R2-R3
6	6*	6*	6	**12/12**	12/15
3			9*		**15/15**

Ответ: 6, 6, 9

4.3 Пример 3.

Потоки могут быть дробными, так как мы работаем с относительно большими пропускными способностями (Мб)



	Flows				Links						
δ	AE	BE	CE	DE	L1	L2	L3	L4	L5	L6	L7
$3\frac{1}{3}$	$3\frac{1}{3}$ *	$3\frac{1}{3}$*	$3\frac{1}{3}$*	$3\frac{1}{3}$	$3\frac{1}{3}/4$	$3\frac{1}{3}/4$	$6\frac{2}{3}/8$	$3\frac{1}{3}/5$	**10/10**	$3\frac{1}{3}/7$	$13\frac{1}{3}/15$
$1\frac{2}{3}$				5*	—||—	—||—	—||—	—||—		5/7	**15/15**

Ответ: AE: $3\frac{1}{3}$, BE: $3\frac{1}{3}$, CE: $3\frac{1}{3}$, DE: 5;

5 Алгоритм AIMD

Одним из алгоритмов регулирования перегрузок является алгоритм AIMD. Суть его очень похожа на Sliding Window. Выберем W_0 - изначальный размер окна (например, 1) и при получении Ack на все пакеты конкретного окна, окно увеличивается. Процесс прогона одного окна называется **round**. То есть когда все пакеты, которые были отправлены в рамках этого раунда, инициировали подтверждения и подтверждения дошли, то окно увеличивается на один пакет. Но постоянно увеличивать мы не можем, иначе рано или поздно канал забъётся. Ровно поэтому в тот момент, когда мы поняли, что какой-то пакет потерялся, мы уменьшаем размер окна *вдвое*.

Для понимания однопоточного алгоритма стоит посмотреть: AIMD Animation

Давайте посмотрим, как решаются задачки. Для начала посмотрим на основные меременные.

- RTT_k - Round Trip Time - время, за которое раунд (Все пакеты раунда k) пройдёт в сети.
- W_k - Количество пакетов на раунде k.
- R - скорость сети
- MSS - Maximum Segent Size - размер пакета.
- B - размер буфера.

Также существуют некоторые формулы.

1. $W_k = W_0 + k \cdot MSS$
2. $B_k = k \cdot MSS$
3. $RTT_k = RTT_0 + k \cdot \frac{MSS}{R}$

Давайте немного опустим эти формулы и начнём с азов.

Во-первых, как может вычисляться скорость?

$$R = \frac{W_0}{RTT}$$

W_0 - это число пакетов, которые пролетают сеть без задержки. Обычно RTT и R мы знаем и хотим узнать как раз-таки W_0. Методом не очень сложных подсчётов получим два результата:

$$W_0 = RTT \cdot R$$
$$\Delta RTT_k = \frac{B_k}{RTT} => RTT_k = RTT_0 + \frac{B_k}{R}$$

Во-вторых, как определить максимальный буфер, который может встретиться на нашем пути. Давайте посмотрим на наш путь и определим бутылочное горлышко. Вместимость буфера перед бутылочным горлышком и будет нашим значением B_{max}, то есть таким, после пробития которого роутер начнёт сбрасывать пакеты.

Теперь мы готовы начать решать задачи.

5.1 Пример 1

Некто качает видео на скорости 1 Mb/s с сервера. Все пакеты имеют размер 125 Байт, ping до сервера в отсутствие передачи данных 5 мс. Во время передачи максимальный и минимальный размеры окна в AIMD не изменяются.

Найти минимальный достаточный размер буфера маршрутизатора в узком месте.

Найти наибольший размер окна отправителя и RTT при его достижении.

$RTT = 5ms$	Во-первых, минимум пилы и максимум пилы неизменны.
$R = 1Mb/s$	Из чего мы делаем вывод, что ниже W_0 мы не можем быть.
$MSS = 125B = 1000b$	Но так как максимум тоже константный, то W_{max} равен $2 \cdot W_0$
	$W_0 = RTT \cdot R = 5ms \cdot 1Mb/s = 5000b = 5MSS = B_{max}$
K - ?	$W_{max} = 10MSS => W_k = 10MSS => K = 5$
RTT_k - ?	$RTT_5 = RTT_0 + \frac{B}{R} = 0.005 + \frac{5MSS}{1000MSS} = 0.01$
	Ответ: $K = 5$; $RTT_k = 0.01$

5.2 Пример 2

- По условию задачи
  - $RTT_0 = 5ms$
  - $R = 1Mb/s$
  - $MSS = 125B = 1000b$
- Через какое время после начала передачи данных буфер будет заполнен, если начальный размер окна 1 MSS?

$RTT = 5ms$	$W_{max} = RTT * R = 1000MSS/s \cdot 5ms = 5MSS$
$R = 1Mb/s$	$RTT_k = RTT_0 + k \cdot \frac{W_k}{RTT}$
$MSS = 125B = 1000b$	$TMO = \sum_1^5 RTT_i = 5RTT_0 + (1+2+3+4+5) \cdot \frac{MSS}{R} = 0.025 + 0.015 = 0.04$
Time-MSS-OF - ?	Ответ: Первый пакет умрёт через $0.04s$

5.3 Пример 3

$RTT = 50ms$ $R = 1Mb/s$ $MSS = 125B = 1000b$ $B_{max} = 1Mb$	1) Пакет сбросится, когда буфер переполнится. Это произойдёт на $K = \frac{B_{max}}{R} = \frac{1000MSS}{MSS} = 1000$ шаге 2) $W_0 = R \cdot RTT = 1000MSS \cdot 50 \cdot 0.001 = 50MSS$
W_{max} - ?	$W_{max} = W_0 + k \cdot MSS = W_0 + B_{max} = 1050MSS$ Ответ: $W_{max} = 1050MSS$

6 Алгоритм TCP Tahoe

7 Алгоритм TCP Reno