

Расчет контрольных сумм в Internet

Computing the Internet Checksum

Статус документа

В этом документе даётся обзор эффективности методов и алгоритмов расчета контрольных сумм Internet. Документ не является стандартом, но описывает ряд полезных методов реализации. Документ может распространяться свободно.

1. Введение

В этом документе обсуждаются методы эффективного расчета контрольных сумм, используемых стандартными протоколами Internet - IP, UDP и TCP.

Эффективность расчета контрольных сумм очень важна с точки зрения производительности. В результате эффективной реализации остальных компонент протокольной обработки расчет контрольных сумм стал, например, одним из факторов, ограничивающих производительность TCP. Обычно для процедур расчета контрольных сумм используется ручная оптимизация с учетом особенностей работы конкретного процессора - экономия доли микросекунды в расчете на один байт данных TCP может привести к существенному снижению суммарного расхода процессорного времени.

В общих чертах алгоритм расчета контрольной суммы очень прост:

- (1) Соседние октеты информации, для которой создается контрольная сумма, объединяются в 16-битовые целые числа и для этих чисел формируется 16-битовое поразрядное дополнение до 1.
- (2) При расчете контрольной суммы значение самого поля контрольной суммы принимается нулевым. Для 16-битовых поразрядных дополнений вычисляется сумма. Для полученного в результате 16-битового целого числа создается 16-битовое поразрядное дополнение до 1 и помещается в поле контрольной суммы.
- (3) Для проверки контрольной суммы вычисляется сумма поразрядных дополнений до 1 для того же набора октетов, включая поле контрольной суммы. Если в результате получается значение, все биты которого равны 1 (-0 в арифметике дополнений до 1), это говорит о корректности контрольной суммы.

Предположим, что контрольная сумма определяется для последовательности октетов A, B, C, D, ... , Y, Z. Будем использовать обозначение [a,b] для 16-битовых целых чисел $a*256+b$, где a и b являются байтами. Тогда сумма 16-битовых дополнений до 1 для этих байтов будет задаваться одним из выражений:

$$\begin{array}{l} [A,B] + ' [C,D] + ' \dots + ' [Y,Z] \qquad [1] \\ [A,B] + ' [C,D] + ' \dots + ' [Z,0] \qquad [2] \end{array}$$

где '+' указывает сложение поразрядных дополнений до 1. Приведенные выше выражения относятся к последовательностям с четным и нечетным количеством байтов, соответственно.

В двоичных машинах сумма поразрядных дополнений до единицы должна вычисляться с использованием «кругового переноса»¹, т. е. при переполнении старшего бита значение переносится в младший, как показано в примерах ниже.

В параграфе 2 рассматриваются свойства контрольной суммы, которые могут использоваться для ускорения расчетов. Параграф 3 содержит несколько числовых примеров для наиболее важных методов реализации. В параграфе 4 приведены несколько конкретных алгоритмов для использования с распространенными типами процессоров. Авторы признательны Van Jacobson и Charley Kline за их вклад в алгоритмы, опубликованные в этом параграфе.

Свойства контрольных сумм Internet изначально были рассмотрены Биллом Пламмером (Bill Plummer) в документе IEN-45, названном "Checksum Function Design". Поскольку документ IEN-45 не получил широкого распространения, он включен в качестве приложения к данному RFC.

2. Расчет контрольной суммы

Эта простая контрольная сумма имеет множество чудесных математических свойств, которые могут быть использованы для вычисления расчетов. Эти свойства обсуждаются ниже.

(A) Коммутативность и ассоциативность

После того, как байты были распределены на четные и нечетные², суммирование может проводиться в любом порядке с возможностью разбиения на произвольные группы.

Например, сумму [1] можно представить в форме:

¹End around carry.

²По порядку их следования, а не по значениям байтов. *Прим. перев.*

(B) Независимость от порядка байтов

Сумма 16-битовых целых чисел может вычисляться для любого порядка байтов. Т. е., если мы рассчитаем сумму, сменив порядок байтов:

$$[B,A] + [D,C] + \dots + [Z,Y] \quad [4]$$

результат будет отличаться от значения [1] только сменой порядка байтов! Для того, чтобы это стало более понятным, отметим, что в обоих случаях перенос происходит из бита 15 в бит 0 и из бита 7 в бит 8. Иными словами, смена порядка суммируемых байтов лишь приводит к смене порядка байтов результата, но сохраняет порядок битов в каждом байте результата.

Следовательно, сумма может рассчитываться одинаково, независимо от используемого оборудованием порядка байтов (big-endian или little-endian¹). В качестве примера предположим, что машина little-endian вычисляет контрольную сумму данных, хранящихся в памяти с использованием сетевого (big-endian) порядка байтов. Выборка каждого 16-битового слова будет приводить к смене мест байт в словах, что приведет к суммированию [4]; однако при сохранении результата в памяти снова произойдет смена мест байтов и будет восстановлен сетевой порядок.

Смена мест байтов может явно использоваться для решения проблем, связанных с выравниванием по границе. Например, вторая группа в [3] может быть рассчитана, как:

$$[K,L] + \dots + [Z,0]$$

если байты результата поменять местами до того, как они будут добавлены к сумме первой группы (см. пример ниже).

(C) Параллельное суммирование

На машинах с размером слова, кратным 16 битам, можно использовать дополнительное увеличение скорости расчетов. Поскольку сложение ассоциативно, мы не обязаны складывать целые числа в порядке их следования в сообщении. Вместо этого мы можем складывать их «параллельно» используя более длинные машинные слова.

Для параллельного расчета контрольной суммы просто выполняется операция поразрядного дополнения до 1 для стандартного размера машинного слова. Например, на 32-разрядных машинах мы можем складывать одновременно по 4 байта : [A,B,C,D]+... После завершения расчета результат более длинное слово «вталкивается» в 16 битов путем сложения 16-битовых сегментов. При каждом сложении могут происходить переносы битов, которые следует учитывать.

Поскольку порядок байтов не имеет значения, мы можем посчитать 32-битовых слов [D,C,B,A]+... или [B,A,D,C]+... и потом поменять местами байты окончательной 16-битовой суммы (см. примеры ниже). Допускаются любые перестановки, которые будут обеспечивать сложение всех четных байтов в один байт суммы, а всех нечетных – в другой байт.

Ниже описано еще несколько методов ускорения расчета контрольных сумм.

(1) Отложенные переносы

Для некоторых типов процессоров эффективность расчета контрольных сумм может быть повышена за счет того, что добавление битов переноса осуществляется после завершения цикла суммирования.

Можно складывать 16-битовые слова в 32-битовую переменную, чтобы все биты переноса складывались в старших 16 битах. Этот вариант позволяет отказаться от использования команд, понимающих перенос битов, но требует удвоенного числа операций сложения, поскольку складываются 32-битовые сегменты. Который из вариантов будет быстрее, зависит от системной архитектуры.

(2) Использование циклов

Для повышения эффективности часто бывает полезно создание внутреннего цикла суммирования, выполняющего серию команд сложения за один проход. Этот метод часто дает существенное повышение эффективности, хотя может значительно усложнить логику программы.

(3) Объединение суммирования и копирования данных

Кроме вычисления суммы время расходуется также на копирование данных из одного места памяти в другое. В обоих случаях узким местом является скорость шины памяти (например, скорость выборки данных из памяти). На некоторых машинах (особенно на сравнительно медленных и простых микрокомпьютерах) производительность можно существенно повысить за счет объединения операций суммирования и копирования при котором происходит только одна выборка из памяти для обеих операций.

(4) Нарастающие обновления

В некоторых случаях можно избежать повторного вычисления всей контрольной суммы, если сменился только заголовок. Наиболее ярким примером может служить изменение поля TTL в заголовке IP при пересылке пакетов шлюзом, но есть и другие примеры (скажем, обновление source route). В таких случаях можно обновить контрольную сумму даже без просмотра сообщения или дейтаграммы.

Для обновления контрольной суммы достаточно просто добавить к ней разность между новым и прежним значениями изменившегося 16-битового слова. Чтобы понять, как это работает, напомним, что каждое целое число имеет аддитивную инверсию², а операция сложения ассоциативна. Если исходное значение слова обозначить m , новое значение - m' , а исходную контрольную сумму - C , тогда новая сумма C' будет равна:

$$C' = C + (-m) + m' = C + (m' - m)$$

¹«Сначала старший» или «сначала младший». Для обозначения этих вариантов используются также термины «сетевой» (network byte order) и «хостовый» (host byte order). *Прим. перев.*

²Число, совпадающее по значению и противоположное по знаку. *Прим. перев.*

3. Численные примеры

Ниже рассматриваются явные примеры расчета сумм дополнений до 1 на машине с дополнением до 2. Примеры показывают, как одну и ту же сумму можно рассчитать путем сложения байтов, 16-битовых слов с нормальным и измененным порядком байтов, а также 32-битовых слов с 3 вариантами порядка байтов. Все числа задаются в шестнадцатеричном формате.

	<i>Побайтовое</i>	<i>Нормальный порядок</i>	<i>Обратный порядок</i>
<i>Байты 0-1</i>	00 01	0001	0100
<i>Байты 2-3</i>	f2 03	f203	03f2
<i>Байты 4-5</i>	f4 f5	f4f5	f5f4
<i>Байты 6-7</i>	f6 f7	f6f7	f7f6
<i>Сумма 1</i>	2dc 1f0	2ddf0	1f2dc
	dc f0	ddf0	f2dc
<i>Переносы</i>	1 2	2	1
<i>Сумма 2</i>	dd f2	ddf2	f2dd
<i>Итог</i>	dd f2	ddf2	ddf2

<i>Байты 0-3</i>	0001f203	010003f2	03f20100
<i>Байты 4-7</i>	f4f5f6f7	f5f4f7f6	f7f6f5f4
<i>Сумма 1</i>	0f4f7e8fa	0f6f4fbe8	0fbe8f6f4
<i>Переносы</i>	0	0	0
<i>Верхняя половина</i>	f4f7	f6f4	fbe8
<i>Нижняя половина</i>	e8fa	fbe8	f6f4
<i>Сумма 2</i>	1ddf1	1f2dc	1f2dc
	ddf1	f2dc	f2dc
<i>Переносы</i>	1	1	1
<i>Сумма 3</i>	ddf2	f2dd	f2dd
<i>Итог</i>	ddf2	ddf2	ddf2

В заключение приводится пример суммирования в виде двух групп, из которых вторая начинается на нечетной границе.

	<i>Побайтовое</i>	<i>Нормальный порядок</i>
<i>Байты 0-1</i>	00 01	0001
<i>Байт 2</i>	f2 (00)	f200
<i>Сумма 1</i>	f2 01	f201
<i>Байты 4-5</i>	03 f4	03f4
<i>Байты 6-7</i>	f5 f6	f5f6
<i>Байт 8</i>	f7 (00)	f700
<i>Сумма 2</i>		1f0ea
<i>Сумма 2</i>		f0ea
<i>Перенос</i>		1
<i>Сумма 3</i>		f0eb
<i>Сумма 1</i>		f201
<i>Сумма 3 после смены порядка</i>		f0eb
<i>Сумма 4</i>		1ddf1
<i>Сумма 4</i>		ddf1
<i>Перенос</i>		1
<i>Сумма 5</i>		ddf2

4. Примеры реализации

Ниже приводятся примеры реализации алгоритма подсчета контрольных сумм Internet, которые доказали свою эффективность для разных типов CPU. В каждом случае приводится ядро алгоритма без включения дополнительного кода (например, связывания подпрограмм).

4.1 Код на языке C

Приведенный ниже пример на языке C показывает расчет контрольной суммы с использованием внутреннего цикла сложения 16-битовых значений в 32-битовый «аккумулятор».

```

in 6
{
    /* Расчет контрольной суммы Internet для count байтов, начиная с addr. */
    register long sum = 0;
    while( count > 1 ) { /* Внутренний цикл */
        sum += * (unsigned short) addr++;
        count -= 2;
    }
    /* Прибавляем байт переноса, если он есть */
    if( count > 0 )
        sum += * (unsigned char *) addr;
    /* поместим 32-битовую сумму в 16 битов */
    while (sum >> 16)
        sum = (sum & 0xffff) + (sum >> 16);
    checksum = ~sum;
}

```

4.2 Motorola 68020

Ниже приведен пример ассемблерной реализации алгоритма для процессора Motorola 68020. Этот вариант использует суммирование 32-битовых значений в один прием и использует внутренний цикл сложения с 16 операциями. Для простоты была опущена логика дополнения последнего слова для случаев, когда число суммируемых байтов не кратно 4. Результат сохраняется в регистре d0.

При тактовой частоте процессора 20 МГц время расчета контрольной суммы составляет 134 мксек/кбайт. Разработал этот алгоритм Van Jacobson.

```

movl    d1,d2
lsrl    #6,d1      | count/64 = # число проходов цикла
andl    #0x3c,d2   | Нахождение частей блока
negl    d2
andb    #0xf,cc    | Сброс X (расширенный флаг переноса)

jmp     pc@(2$-. -2:b,d2) | Переход в цикл

1$:     | Начало внутреннего цикла...

movl    a0@+,d2    | Выборка 32-битового слова
addxl   d2,d0      | Сложение слова и предыдущего переноса
movl    a0@+,d2    | Выборка 32-битового слова
addxl   d2,d0      | Сложение слова и предыдущего переноса
| ... еще 14 повторов

2$:     dbra    d1,1$ | (Отметим, что dbra не воздействует на X)

movl    d0,d1     | Вталкивание 32 битов суммы в 16 битов
swap   d1        | (Отметим, что swap не воздействует на X)
addxw  d1,d0
jcc    3$
addw   #1,d0

3$:     andl    #0xffff,d0

```

4.3 Cray

Ниже приводится ассемблерная реализация алгоритма для процессора Cray, которую предложил Charley Kline. Расчет контрольной суммы производится как векторная операция, обеспечивающая одновременное сложение до 512 байтов с базовым блоком суммирования 32 бита. Для простоты из примера исключены фрагменты, обеспечивающие возможность работы с короткими блоками.

Регистр A1 содержит адрес 512-байтового блока памяти для контрольной суммы. Две первых копии данных загружаются в два векторных регистра. Один вектор сдвигается вправо на 32 бита, а для второго используется операция AND с 32-битовой маской. После этого векторы складываются. Поскольку все эти операции связаны в цепочку, они дают один результат на каждый цикл процессора. Далее производится сжатие (collaps) результирующего вектора в цикле, который прибавляет каждый элемент к скалярному регистру. В заключение выполняется перенос и результат помещается в 16 битов.

```

EVM
A0      A1
VL      64      используются полные векторы
S1      <32     формируется 32-битовая маска из правой части.
A2      32
V1      ,A0,1   загрузка пакета в V1
V2      S1&V1   формирование "правых" 32 битов в V2.
V3      V1>A2   формирование "левых" 32 битов в V3.
V1      V2+V3   Сложение.
A2      63      Подготовка к сжатию в скаляр.
S1      0
S4      <16     формирование 16-битовой маски справа.
A4      16
СК$LOOP S2     V1,A2
A2      A2-1
A0      A2
S1      S1+S2
JAN     СК$LOOP
S2      S1&S4   формирование "правых" 16 битов в S2
S1      S1>A4   формирование "левых" 16 битов в S1
S1      S1+S2
S2      S1&S4   формирование "правых" 16 битов в S2
S1      S1>A4   формирование "левых" 16 битов в S1
S1      S1+S2
S1      #S1     Получение дополнения до 1
CMR     В этой точке S1 будет содержать контрольную сумму.

```

4.4 IBM 370

Следующий пример на ассемблере для процессора IBM 370 суммирует по 4 байта одновременно. Для простоты опущен код дополнения, используемых для выравнивания данных по 4-байтовой границе и обращения порядка байтов, когда это требуется. Результат сохраняется в регистре RCARRY.

Этот код на процессоре IBM 3090 давал время расчета 27 мксек/кбайт при расчете контрольной суммы байтов, содержащих только единицы. Время расчета снижается до 24.3 мксек/кбайт, если применить средства выравнивания слов (специальная обработка в начале и в конце, а при необходимости замена местами байтов при расчете с нечетной позицией).

```

*   Регистры RADDR и RCOUNT содержат адрес и размер суммируемого блока.
*
*   (RCARRY, RSUM) должны быть парой регистров (четный/нечетный).
*   (RCOUNT, RMOD) должны быть парой регистров (четный/нечетный).
*
CHECKSUM  SR   RSUM,RSUM      Сброс рабочих регистров.
          SR   RCARRY,RCARRY
          LA   RONE,1         Установка значения 1.
*
          SRDA RCOUNT,6      Count/64 в RCOUNT.
          AR   RCOUNT,RONE   +1 = # число циклов.
          SRL  RMOD,26        Размер частичного блока в RMOD.
          AR   RADDR,R3       Выравнивание для компенсации перехода
          S    RADDR,=F(64)   в цикл.
          SRL  RMOD,1         (RMOD/4)*2 - индекс "полуслов".
          LH   RMOD,DOPEVEC9(RMOD) используется специальный вектор для смещения
          B    LOOP(RMOD)     и перехода в цикл...
*
*   Внутренний цикл:
*
LOOP      AL   RSUM,0(,RADDR)  Сложить логические слова
          BC   12,**+6        Переход, если нет переноса
          AR   RCARRY,RONE     Добавить 1 переноса
          AL   RSUM,4(,RADDR)  Сложить логические слова
          BC   12,**+6        ветвление, если нет переноса
          AR   RCARRY,RONE     Добавить 1 перенос
*
*   ... еще 14 повторов ...
*
          A    RADDR,=F'64'    Увеличить адресный указатель
          BCT  RCOUNT,LOOP    Перейти к Count
*
*   Прибавить переносы к сумме и "затолкнуть" в 16 битов
*
          ALR  RCARRY,RSUM     Сложить слова SUM и CARRY
          BC   12,**+6        и учесть возможный перенос
          AR   RCARRY,RONE
          SRDL RCARRY,16       Поместить 32-битовую сумму
          SRL  RSUM,16         в 16 битов
          ALR  RCARRY,RSUM
          C    RCARRY,=X'0000FFFF' Прибавить оставшийся перенос
          BNH  DONE
          S    RCARRY,=X'0000FFFF'
DONE      X    RCARRY,=X'0000FFFF' Дополнить до 1

```

IEN 45¹

Секция 2.4.4.5

TCP Checksum Function Design

William W. Plummer

Bolt Beranek and Newman, Inc.
50 Moulton Street
Cambridge MA 02138

5 June 1978

1. Введение

Контрольные суммы включаются в пакеты для того, чтобы можно было детектировать ошибки, возникшие в процессе передачи. Для протоколов Internet, таких, как TCP [1,9], это особенно важно, поскольку пакеты могут передаваться через беспроводные сети типа Packet Radio Network [2] и Atlantic Satellite Network [3], в которых пакеты могут повреждаться. Протоколы Internet (например, используемые для передачи голоса в реальном масштабе времени) могут быть устойчивы к некоторому уровню ошибок при передаче и использование методов упреждающей коррекции ошибок или возможность отказа от использования контрольных сумм может давать хорошие результаты. Эта статья посвящена функциям подсчета контрольных сумм для таких протоколов, как TCP, где гарантии доставки обеспечиваются за счет повторной передачи.

Если принятое сообщение содержит корректную контрольную сумму, это еще не гарантирует отсутствия в сообщении незамеченных ошибок. Вероятность наличия таких ошибок составляет 2^{-C}, где C – число битов контрольной суммы. Ошибки могут возникать в результате некорректной работы оборудования (и программ), а также при передаче. Проявления аппаратных ошибок обычно хорошо известны и такие ошибки желательно принимать во внимание при создании функций подсчета контрольных сумм. В идеальном случае все аппаратные ошибки типа "отказ в оборудовании" должны быть обнаружены.

¹Этот документ доступен на сайте <http://www.rfc-editor.org/rfc/iен/iен45.txt>. Прим. перев.

Примером ошибок, обнаруживаемых при проверке контрольных сумм, являются битовые ошибки в сетевом интерфейсе (шине ввода-вывода, канале памяти и т. п.). Такие ошибки всегда будут приводить к некорректности контрольной суммы. В качестве примера неадекватности современной функции подсчета контрольных сумм предположим, что на сетевом интерфейсе прекратили работать сигналы управления и интерфейс сохраняет нули вместо реальных данных. Сообщения, включающие только нули, будут иметь корректную контрольную сумму. Шум в линии "There's Your Bit"¹ интерфейса ARPANET [4] может остаться незамеченным, поскольку дополнительные биты на входе могут приводить к тому, что контрольная сумма будет искажена (т. е., смещена) так же, как и данные.

Хотя сообщения с незамеченными ошибками будут иногда передаваться на вышележащие уровни протокола, они обычно не будут важны для этого уровня. В случае TCP большинство таких сообщений будет игнорироваться, но некоторые могут приводить к разрыву соединения. Искаженные данные могут вызывать проблемы для уровня протокола, лежащего выше TCP, который может использовать свою контрольную сумму.

Эта статья является первым этапом создания новой функции подсчета контрольной суммы для TCP и некоторых других протоколов Internet. Современная функция имеет несколько полезных свойств, которые возможно следует сохранить в новой функции. Исследовано множество схем подсчета контрольных сумм. Из числа этих схем лишь схема, основанная на умножении², представляется достаточно простой.

2. Современная функция подсчета контрольных сумм TCP

Современная функция ориентирована на 16-битовые машины типа PDP-11, но она может без труда использоваться и на иных компьютерах (например, PDP-10). Пакет представляет собой "строку" 16-битовых слов³ и контрольная сумма представляет собой сумму дополнений до 1 (сложение с переносом) этих слов. Это значение помещается в поле контрольной суммы пакета TCP. Перед вычислением контрольной суммы само значение поля контрольной суммы в заголовке устанавливается нулевым. Если контрольная сумма, рассчитанная для пакета его получателем, равна нулю, пакет предполагается не содержащим ошибок. Это является результатом учета на стороне получателя "отрицательного" значения поля контрольной суммы, которое по величине в точности совпадает с контрольной суммой остальной части пакета.

Если не принимать во внимание сложность самого расчета контрольной суммы для данного пакета, описанный механизм использования контрольных сумм достаточно прост. Механизм предполагает наличие некоторых свойств оператора вычисления контрольной суммы (дополнение до 1, обозначенное ниже как "+"):

- (P1) Операция + является коммутативной, т. е. результат не зависит от порядка сложения 16-битовых слов.
- (P2) Операция + имеет по крайней мере один тождественности⁴ (используемая в настоящее время функция имеет два таких элемента - +0 и -0). Это позволяет отправителю помещать нулевое значение в поле контрольной суммы перед расчетом последней.
- (P3) Контрольная сумма имеет противоположное значение. Таким образом, получатель может проверить значение контрольной суммы и ожидается получение нулевого результата при расчете.
- (P4) Операция + является ассоциативной, что позволяет размещать поле контрольной суммы в любом месте пакета и просматривать 16-битовые слова последовательно.

Математически эти свойства бинарной операции "+" над множеством 16-битовых целых чисел образуют абелеву группу⁵ [5]. Существуют и другие абелевы группы, но не все они подходят для использования в качестве операторов контрольной суммы. (другим подходящим оператором из набора команд PDP-11 является исключительное ИЛИ, но операция XOR не подходит по иным причинам).

Есть еще ряд свойств, которые должны быть сохранены в любой будущей схеме расчета контрольных сумм:

- (P5) Операция + выполняется достаточно быстро на различных машинах и не требует значительной памяти.

Современная версия достаточно хороша с этой точки зрения. На PDP-11 внутренний цикл имеет вид:

```

LOOP:  ADD (R1)+,R0    ; Прибавить следующее 16-битовое слово
        ADC R0        ; Добавить перенос
        SOB R2,LOOP   ; Цикл для всего пакета

```

(4 такта на каждое 16-битовое слово)

На PDP-10 возможно дополнительное использование свойств P1-4 и обработка двух 16-битовых слов в одном цикле:

```

LOOP:  ILDB THIS,PTR    ; Принять два 16-битовых слова
        ADD SUM,THIS    ; Прибавить к текущему значению суммы
        JUMPGE SUM,CHKSU2 ; Переход, если меньше 8 переносов
        LDB THIS,[POINT 20,SUM,19] ; Принять левые 16 битов и переносы
        ANDI SUM,177777 ; Сохранить младшие 16 битов
        ADD SUM,THIS    ; Прибавить переносы
CHKSU2: SOJG COUNT,LOOP ; Цикл для всего пакета

```

(3,1 такта на каждое 16-битовое слово)

Дополнительная команда в приведенном выше цикле требуется для преобразования команды прибавления дополнения до 2 в команду прибавления дополнения до 1 путем кругового переноса. Арифметика дополнений до 1 лучше, чем арифметика дополнений до 2, поскольку она чувствительна к ошибкам в любом бите. Если используется дополнение до 2, можно отбросить (или установить) четное число единиц в канале старших битов без изменения результирующей контрольной суммы. Это просто свойство, которое делает сложение более предпочтительным по сравнению с операцией «исключительное ИЛИ», которая используется достаточно часто, но позволяет сбросить (или

¹Это бит для вас.

²Product code – использование арифметического произведения в качестве кода контроля целостности.

³В исходном документе используется термин «байт». Но, поскольку этот термин в настоящее время используется преимущественно для обозначения 8-битовых слов, в переводе вместо термина «байт» использован термин «слово». *Прим. перев.*

⁴Identity element

⁵Abelian group

установить) четное число 1 в любом битовом канале. Формат бумажной ленты RIM10B в PDP-10s [10] заставляет использовать прибавление дополнения до 2 в силу ограничений программы загрузки.

Еще одно свойство современного оператора контрольной суммы рассмотрено ниже.

(P6) Добавление в пакет контрольной суммы не меняет информационных байтов. Peterson [6] назвал это «систематическим» кодом.

Это свойство позволяет промежуточным узлам (например, шлюзам) менять значения полей (например, Internet Destination Address) без полного декодирования пакета. Контрольные суммы CRC¹, используемые для коррекции ошибок, не являются симметричными. Однако большинство применений CRC предназначено скорее для детектирования, нежели для исправления ошибок и, следовательно, сообщения могут передаваться в неизменном виде с добавлением в конце значения CRC. 24-битовые суммы CRC, применяемые в интерфейсах ARPANET IMP и Very Distant Host [4], а также в стандарте ANSI для 800 и 6250 битов на дюйм магнитной ленты [11], используют именно такой режим.

Отметим, что операции, выполняемые шлюзами по отношению к некорректным пакетам, не должны оказывать влияния на работу протоколов вышележащих уровней. Шлюзы могут проверять корректность контрольной суммы во входящих пакетах, но в общем случае шлюз просто не знает, как это сделать, если контрольная сумма является свойством конкретного протокола.

И, наконец, еще одно свойство современной схемы расчета контрольных сумм является следствием P1 и P4.

(P7) Контрольная сумма допускает нарастающие обновления.

Mo		Ms		Mg
A	кодирование	7	декодирование	A
B	==>	1	==>	B
C		4		C
		...		
		2	контрольная	плюс "корректный" флаг
			информация	
Исходное		Переданное		Восстановленное

Это свойство позволяет промежуточным шлюзам добавлять в пакет информацию (например, временную метку) с соответствующим изменением поля контрольной суммы в пакете. Отметим, что обновленная контрольная сумма сохраняет возможность сквозного контроля и при обновлении не нужно заново выполнять суммирование для неизменной части пакета.

3. Произведения

Некоторые произведения² могут быть полезны для расчета контрольных сумм. Ниже приводится краткое рассмотрение использования произведений в контексте TCP. Более общее описание можно найти в работе Avizienis [7] и возможно в более поздних работах.

Базовая концепция такого кодирования заключается с тем, что сообщение (пакет) для передачи формируется путем преобразования исходного сообщения с добавлением некоторых битов контроля. Читая сообщение и преобразуя (возможно, по иному) его, получатель может восстановить исходное сообщение и определить факт его повреждения при передаче.

Использование произведения будет давать $M_s = K * M_o$. То есть, передаваемое сообщение M_s является просто произведением исходного сообщения M_o и хорошо известной константы K . Для декодирования принятое сообщение M_s делится на K , что будет давать в результате сообщение M_g и 0 в остатке, если M_g считается совпадающим с M_o .

Первой проблемой является выбор "хорошего" значения для "контрольного множителя" K . Число K должно быть простым по отношению к базе, выбранной для представления сообщения. Например, двоичное сообщение с некорректно выбранным значением $K=8$. Это приведет к тому, что сообщение M_s будет отличаться от M_o лишь добавлением трех нулей в конце. Единственным случаем, когда получатель может обнаружить ошибку, деля полученное сообщение на 8, - это наличие ошибки в одном из этих трех битов.

Для протокола TCP в качестве базы R выбрано значение 216. Таким образом, каждое 16-битовое слово (машинное слово PDP-11) будет рассматриваться как «цифра» большого «числа», представляющего собой сообщение. То есть,

$$M_o = \text{SIGMA} [B_i * (R^i)]$$

где B_i – i -ое слово (i лежит в диапазоне от 0 до N)

$$M_s = K * M_o$$

Повреждение одной цифры M_s приведет к тому, что M_s' будет на $C^*(R^j)$ больше или меньше исходного произведения. Получатель будет получать значение M_s'/K . Поскольку значения R и K являются относительно простыми, $C^*(R^j)$ не может быть кратно K . Следовательно, деление приведет к наличию ненулевого остатка, который будет показывать, что M_s' является поврежденным вариантом M_s . Легко видеть, что хорошим выбором для K является $(R_b - 1)$, для некоторого значения b , задающего "длину проверки", которая управляет уровнем контроля за детектированием «пакетных» ошибок³, воздействующих на строку цифр (т. е., 16-битовых слов) в сообщении. Фактически было выбрано значение $b = 1$, поэтому K будет равно $216 - 1$, что упрощает выполнение арифметических операций. Это означает, что все «пакеты» ошибок, затрагивающие не более 15 битов будут детектироваться. Согласно работе [7] такой выбор значения b приводит к следующей оценке доли недетектируемых ошибок с весом 2:

$$f = 16(k-1) / [32(16k-3) + (6/k)]$$

где k – длина сообщения.

Для больших пакетов f будет стремиться к 0,03125 (если k устремить к бесконечности).

¹Cyclic Redundancy Check

²product codes

³burst error

Многочисленные операции умножения и деления с высокой точностью обычно достаточно сложны, особенно на маленьких машинах, для которых даже деление и умножение с одинарной точностью представляют собой непростую задачу. Исключением являются ситуации, к которой относится и описываемая здесь, когда используется коэффициент $216 - 1$ на машине с размером слова 16 битов. Причину этого объясняет следующее выражение:

$$Q * (R^{**j}) = Q, \text{ mod } (R-1) \quad 0 \leq Q < R$$

То есть, любая «цифра» Q из заданного диапазона $(0, 1, \dots, R-1)$, умноженная на любую степень основания системы счисления (radix), будет давать остаток Q при делении на $\text{radix} - 1$.

Возьмем для примера десятичную систему счисления ($R = 10$) и $Q = 6$.

$$\begin{aligned} 6 &= 0 * 9 + 6 = 6, \text{ mod } 9 \\ 60 &= 6 * 9 + 6 = 6, \text{ mod } 9 \\ 600 &= 66 * 9 + 6 = 6, \text{ mod } 9 \quad \text{и т. д.} \end{aligned}$$

Более того,

$$\begin{aligned} \text{rem}(31415/9) &= \text{rem}((30000+1000+400+10+5)/9) \\ &= (3 \text{ mod } 9) + (1 \text{ mod } 9) + (4 \text{ mod } 9) + (1 \text{ mod } 9) + (5 \text{ mod } 9) \\ &= (3+1+4+1+5) \text{ mod } 9 \\ &= 14 \text{ mod } 9 \\ &= 5 \end{aligned}$$

Итак, остаток от деления числа на $\text{radix}-1$ может быть найден путем сложения всех цифр этого числа. Поскольку значение radix для случая TCP было выбрано 216 а «контрольный множитель» равен $216 - 1$, контрольную сумму легко посчитать, суммируя все 16-битовые слова (на PDP-11) с переносом битов переполнения в младший бит суммы. Если в результате получается 0, сообщение будет рассматриваться как корректное. Таким образом, проверка произведения имеет такую же степень сложности, как и современный алгоритм подсчета контрольных сумм TCP!

Для того, чтобы сформировать M_s , отправитель должен умножить с использованием высокой точности «число» M_o на $216 - 1$. Это еще можно выразить, как $M_s = (216)M_o - M_o$. Такая операция выполняется путем сдвига M_o влево на размер слова и вычитанием значения M_o . Поскольку требуется передача битов переноса между цифрами, но интересует только текущая цифра, используется дополнение до 1.

$$\begin{array}{r} (2^{**16})M_o = M_o0 + M_o1 + M_o2 + \dots + M_oX + 0 \\ - M_o = - (M_o0 + M_o1 + \dots + M_oX) \\ \hline M_s = M_s0 + M_s1 + \dots - M_oX \end{array}$$

Для реализации этой функции на PDP-11 может использоваться цикл вида:

```
LOOP:  MOV -2(R2),R0    ; Следующее слово (2**16)Mo
        SBC R0         ; Передача переносов из последнего вычитания (SUB)
        SUB (R2)+,R0   ; Вычитание слова Mo
        MOV R0,(R3)+   ; Сохранение в Ms
        SOB R1,LOOP    ; Цикл для всего сообщения
                          ; 8 тактов на 16-битовое слово
```

Отметим, что процедура кодирования выполняется отдельно, поскольку она не является систематичной. В общем случае исходное сообщение M_o будет сохраняться отправителем на случай повторной передачи и, следовательно, должно оставаться доступным. Поэтому требуется команда $\text{MOV } R0,(R3)+$, занимающая 2 из восьми тактов цикла.

Процедура кодирования будет добавлять к сообщению ровно одно 16-битовое слово, поскольку $M_s < (216)M_o$. Дополнительные 16 битов будут добавляться в хвост сообщения, но могут быть перемещены в заданное место заголовка TCP непосредственно перед передачей. Получатель будет возвращать M_s в стандартный формат перед декодированием сообщения.

Код на стороне получателя для полного декодирования сообщения основан на том факте, что любое слово в M_s содержит разность между двумя последовательными словами M_o за вычетом переносов из предыдущего слова, а младшее слово содержит отрицательное значение младшего слова из M_o . Таким образом, младшее (самое правое) слово M_r представляет собой просто отрицательное значение младшего слова M_s . Следующее слово M_r будет равно сумме следующего слова M_s , слова M_r и переноса из предыдущей операции.

Для защиты от передачи адресату сообщений, содержащих только нули требуется незначительное усовершенствование этой процедуры. Такие сообщения будут представляться корректными, поскольку $M_s/K = 0/K = 0$ с нулевым остатком. Усовершенствование заключается в использовании кодирования

$$M_s = K * M_o + C$$

где C – некая произвольная общеизвестная константа. Добавление этой константы требует второго прохода, но он обычно занимает очень мало времени, поскольку завершается как только прекратится передача переносов. Выбор $C = 1$ подходит для большинства случаев.

Контрольная сумма, созданная путем умножения, должна быть оценена с точки зрения наличия свойств P1 - P7. Показано, что требуется удвоенное число машинных тактов на расчет или проверку контрольной суммы методом умножения (P5 выполняется?).

Хотя код не является систематическим, контрольную сумму можно быстро проверить без декодирования сообщения. Если адрес получателя расположен на менее значимом конце пакета (где начинается расчет произведения), шлюзы могут декодировать только начало сообщения, не разбирая его целиком. Таким образом условие P6 выполняется по крайней мере частично. Алгебраические условия P1 - P4 не выполняются, но лишь незначительные вычисления потребуются для их выполнения – сообщение нужно реформатировать, как было показано выше.

Условие P7 выполняется, поскольку для полученной путем умножения контрольной суммы может осуществляться нарастающее обновление с учетом добавленного слова, хотя такое обновление включает часть процедуры. Предположим, что исходное сообщение имеет две половины - H_1 и H_2 . Таким образом, $M_o = H_1 * (R_j) + H_2$. Слово, содержащее временную метку, вставляется между этими половинками, формируя измененное сообщение

$$M_o' = H_1 * (R^{**j+1}) + T * (R^{**j}) + H_2$$

Поскольку для K было выбрано значение $R-1$, передаваемое сообщение будет иметь форму $M_s' = M_o'(R-1)$. Тогда

$$Ms' = Ms * R + T(R-1)(R^{**j}) + P2((R-1)**2)$$

$$= Ms * R + T * (R^{**j}) + T * (R^{**j}) + P2 * (R^{**2}) - 2 * P2 * R - P2$$

Памятью о том, что $R = 216$ (размер слова PDP-11), умножение на R означает копирование вниз на одно слово в памяти. Таким образом, первый член Ms' является просто исходным сообщением, скопированным на одно слово вниз. Следующий член представляет собой новые данные T , добавляемые в Ms' , начиная с $(j+1)$ -го слова. Добавление здесь достаточно просто, поскольку после добавления в T остается лишь «распространение» переноса и процесс завершается, когда переноса не возникает. Остальные члены могут обрабатываться аналогично.

4. Более сложные коды

Существует целая теория кодов детектирования и коррекции ошибок. Книга Петерсона [6] является прекрасным справочником по таким кодам. Большинство из этих схем CRC создавались для реализации с использованием регистра сдвига с обратной связью на основе логики «исключающее ИЛИ».

Программная реализация такого логического устройства будет слишком медленной и бесполезной, если не использовать некоторые хитрости при программировании.

Одну из таких хитростей предложил Kirstein [8]. Небольшое число (4 или 8) битов текущего состояния регистра сдвига комбинируются с битами входного потока (M_0) и результат используется как индекс таблицы, которая дает новое состояние регистра сдвига и, если код не является систематическим, биты для выходного потока (M_s). Пробное кодирование заведомо «хорошей» функции CRC с использованием 4-битовых слов показало, что этот метод работает примерно в 4 раза медленнее современной функции расчета контрольной суммы. Это наблюдалось как на машинах PDP-10, так и на PDP-11. Из перечисленных выше желательных свойств схемы CRC обладают лишь P3 (инверсный) и P6 (систематический). Местоположение поля контрольной суммы в пакете имеет важное значение и CRC не допускает нарастающего обновления.

Хотя основная часть теории кодирования имеет дело с двоичным кодом, значительная часть этой теории работает для алфавитов, содержащих q -символы, представляющие собой степень простого числа. Например, $q = 2^{16}$ хорошо подходит для работы на уровне слов.

5. Внешняя обработка

Когда вовлеченные в расчет контрольной суммы функции требуют большой вычислительной мощности, решением может послужить перенос вычислений на внешний процессор. В этом случае кодирование и декодирование сообщения становятся одиночными командами, которые не загружают основной процессор хоста. Компьютер Digital Equipment Corporation VAX/780 имеет специальное оборудование для генерации и проверки значений CRC [13]. В общем случае это не является хорошим решением, поскольку такие процессоры потребуются создавать для каждой хост-машины, использующей сообщения TCP.

Вполне возможно, что функции шлюза для большого хоста могут выполняться целиком на специальной машине (Internet Frontend Machine). Эта машина будет отвечать за пересылку пакетов, полученных из сетей или от протокольных модулей Internet в подключенном хосте, а также за сборку фрагментов и передачу их хосту. Другая возможность этой машины заключается в проверке контрольных сумм, чтобы для передаваемых хосту сегментов обеспечивалась корректность на момент выхода из frontend-машины. Поскольку вычислительные ресурсы frontend предполагаются недорогими и доступными, такое решение представляется разумным.

Проблема проверки контрольных сумм на frontend-машине нарушает принцип сквозного использования контрольных сумм. Во всяком случае здесь имеется наиболее мощное средство в виде контрольных сумм TCP! Это позволяет покрыть сквозной контрольной суммой соединение между хостом и машиной frontend. Для такого покрытия требуется разработать отдельный, компактный протокол. После проверки пакета приходящего из сети frontend-машина будет передавать пакет хосту, говоря ему: «Этот сегмент из Internet для вас. Назовите его #123». Хост будет сохранять этот сегмент и передавать копию обратно на frontend, говоря: «Этот сегмент был прислан мне, как #123. Это верно?». Машина frontend будет пословно сравнивать полученное с переданным ранее и скажет хосту: «Да, это #123» или «Вы корректно приняли #123. Передайте сегмент подходящему модулю для обработки.»

Заголовки сообщений, проходящих через соединение между хостом и машиной frontend очевидно будут учитываться достаточно сильным алгоритмом расчета контрольных сумм, поэтому информация типа выполненных функций и нумерации сообщений будет доставляться с гарантией. Эти заголовки достаточно коротки (примерно 15 битов), поэтому алгоритм может быть достаточно сильным. Основная часть сообщения при расчете суммы во внимание приниматься не будет.

Причина снижения вычислительной нагрузки на хост заключается в том, что от него не будет требоваться ничего, кроме валидации сообщения с использованием сквозной передачи контрольных сумм и уведомления frontend-машины о результате. Для хоста PDP-10 это потребует 0,5 цикла памяти на 16-битовое слово сообщения Internet и лишь несколько процессорных тактов для организации требуемых передач.

6. Заключение

Существует несколько вариантов функций для работы с контрольными суммами. Первый, самый простой вариант не делает ничего, что обеспечивало бы детектирование или корректировку ошибок. Второй вариант заключается в передаче постоянного значения, которое проверяется на другой стороне, он также достаточно слаб. Третий вариант заключается в использовании операции Исключающее-ИЛИ (XOR) по отношению к передаваемым данным. Операция XOR требует минимальных вычислительных ресурсов для генерации и проверки, но не обеспечивает качественных контрольных сумм. Суммирование с дополнением до 2 дает несколько лучшие результаты, не требуя значительных ресурсов процессорного времени. Пятым вариантом является суммирование с дополнением до 1, используемое в протоколе TCP. Этот метод требует чуть больше процессорного времени. Следующий вариант (product code). Основан на суммировании дополнений до 1, занимает еще больше процессорного времени, но обеспечивает защиту от распространенных аппаратных ошибок, хотя ему и присущи некоторые недостатки. Далее следуют полиномиальные контрольные суммы CRC, которые применяются только для контроля. Этот метод достаточно дорог в реализации. И, наконец, может использоваться полная схема CRC для детектирования и корректировки ошибок.

Для приложений TCP и Internet подходит схема product code. Основным ее недостатком является необходимость декодирования (зачастую многократного) на промежуточных шлюзах в цепочке пересылки. Если схема product code не может быть выбрана для повышения эффективности контрольных сумм, можно слегка изменить существующую схему. Например функция add and rotate (добавить и повернуть), используемая для бумажных лент в группе PDP-6/10 лаборатории Artificial Intelligence Laboratory в рамках проекта M.I.T. Project MAC [12], может оказаться полезной, если удастся доказать, что она лучше текущей схемы и может эффективно использоваться на разных машинах.

Литература

- [1] Cerf, V.G. and Kahn, Robert E., "A Protocol for Packet Network Communications," IEEE Transactions on Communications, vol. COM-22, No. 5, May 1974.
- [2] Kahn, Robert E., "The Organization of Computer Resources into a Packet Radio Network", IEEE Transactions on Communications, vol. COM-25, no. 1, pp. 169-178, January 1977.
- [3] Jacobs, Irwin, et al., "CPODA - A Demand Assignment Protocol for SatNet", Fifth Data Communications Symposium, September 27-9, 1977, Snowbird, Utah
- [4] Bolt Beranek and Newman, Inc. "Specifications for the Interconnection of a Host and an IMP", Report 1822, January 1976 edition.
- [5] Dean, Richard A., "Elements of Abstract Algebra", John Wiley and Sons, Inc., 1966
- [6] Peterson, W. Wesley, "Error Correcting Codes", M.I.T. Press Cambridge MA, 4th edition, 1968.
- [7] Avizienis, Algirdas, "A Study of the Effectiveness of Fault-Detecting Codes for Binary Arithmetic", Jet Propulsion Laboratory Technical Report No. 32-711, September 1, 1965.
- [8] Kirstein, Peter, private communication
- [9] Cerf, V. G. and Postel, Jonathan B., "Specification of Internetwork Transmission Control Program Version 3", University of Southern California Information Sciences Institute, January 1978.
- [10] Digital Equipment Corporation, "PDP-10 Reference Handbook", 1970, pp. 114-5.
- [11] Swanson, Robert, "Understanding Cyclic Redundancy Codes", Computer Design, November, 1975, pp. 93-99.
- [12] Clements, Robert C., частное сообщение.
- [13] Conklin, Peter F., and Rodgers, David P., "Advanced Minicomputer Designed by Team Evaluation of Hardware/Software Tradeoffs", Computer Design, April 1978, pp. 136-7.

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru