ПАКЕТНАЯ ПЕРЕДАЧА ДАННЫХ

Исторически так сложилось, что компьютерные сети имеют последовательную природу. Объяснить это можно тем, что реализовать передачу данных на сравнительно больши'е расстояния в параллельном виде значительно сложнее, чем в последовательном. Между станциями данные передаются по последовательным каналам, а внутри станций обрабатываются параллельно.

Для именования порции информации, передаваемой по каналам компьютерных (и не только компьютерных) сетей, используется обобщенный термин *пакет* (packet). Пакет содержит последовательно сформированные станцией-передатчиком *поля* (fields), предназначенные для их интерпретации в станции-приемнике. В общем случае, пакеты могут быть самыми разнообразными (как по структуре, так и по длине), но подавляющее большинство пакетов подпадают под типовую структуру.

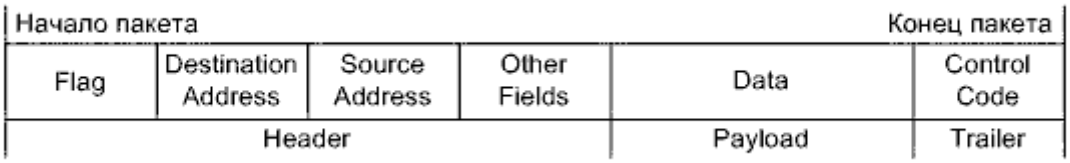

Рисунок -- Структура типового пакета КС

Назначение полей:
1. Flag -- флаг, точнее, флаг начала пакета -- позволяет определить начало пакета.
2. Destination Address -- адрес назначения -- позволяет указать станцию, для которой предназначен пакет.
3. Source Address -- адрес источника -- позволяет указать станцию, сгенерировавшую пакет.
4. Other Fields -- прочие поля -- специфические поля (в том числе и специфические флаги) определенной реализации.
5. Data -- данные -- «полезное» наполнение пакета.
6. FCS (Frame Check Sequence) -- контрольная сумма -- позволяет проверить целостность пакета.

Часть пакета, включающую поля, расположенные до начала данных, принято называть *заголовком* (header) пакета, после данных -- *хвостовиком* (trailer).

Обычно в байт-ориентированных реализациях длина пакета кратна восьми битам, то есть пакет состоит из так называемых *октетов* (octets).

Все поля в составе любого пакета можно условно разделить на полезные и служебные. *Полезная нагрузка* (payload) заключается в собственно данных. Но следует понимать, что вкладываемая в качестве данных информация может носить служебный характер. В некоторых пакетах поле данных не предусмотрено вообще. Сколько дополнительного трафика порождается в связи с наличием служебных полей оценивается как overhead.

В соответствии с концепцией модели OSI, соседние уровни абстрагированы друг от

друга. Поэтому вполне закономерно, что на каждом уровне работают со своими структурами данных. При продвижении информации между уровнями возникает необходимость в преобразованиях структур данных. Преобразования выражаются в инкапсуляции и декапсуляции. Под *инкапсуляцией* (encapsulation) в КС понимается вкладывание пакета определенного вышестоящего уровня в поле данных пакета смежного нижестоящего уровня в процессе подготовки к передаче, то есть при продвижении сверху вниз. Под *декапсуляцией* (decapsulation) понимается обратное действие после приема, то есть при продвижении снизу вверх.

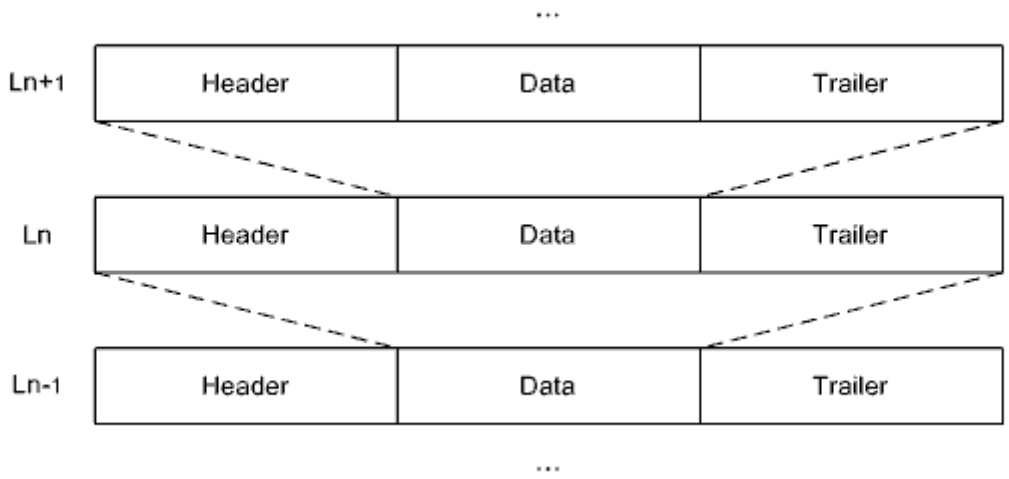

Рисунок -- Инкапсуляция пакетов

Функционал любого из вышестоящих уровней «знает», какие нижестоящие ресурсы ему требуется и чем он «располагает». Поэтому процесс инкапсуляции не доставляет трудностей. А вот функционал нижестоящего уровня при разборе полученных пакетов заранее не знает, какой из вышестоящих подсистем передавать эти пакеты. Проблема решается введением в структуру пакета служебного поля, в котором записывается код протокола вышестоящего уровня.

Важной особенностью инкапсуляции является то, что в большинство реализаций заложена возможность передавать пакеты, относящиеся к некоторому протоколу некоторого уровня (например, сетевого), вкладывая их в пакеты другого протокола того же уровня, то есть организовывать *туннелирование* (tunneling).

Инкапсуляция имеет еще ряд проявлений.

Если при выполнении инкапсуляции данные некоторого уровня не помещаются в поле отведенной длины, то можно прибегнуть к *фрагментации* (fragmentation) -- разбить данные на фрагменты и передать цепочку пакетов. Принимающая сторона будет вынуждена выполнить *дефрагментацию* (defragmentation).

Перемежение (interleaving) позволяет «распараллелить» пересылку пакетов или их фрагментов и заключается в одновременном задействовании нескольких каналов. Особенно это применимо в низкоскоростных СрПД.

Фрагментация (при наличии альтернативных путей в СПД) и перемежение могут привести к «перемешиванию» пакетов и, как следствие, разрушению сообщения. Контроль за порядком фрагментов может быть возложен как на протокол подверженного фрагментации уровня, так и протокол вышестоящего уровня.

Несмотря на целостность уровней, вышестоящие уровни зависят от нижестоящих. Но степень зависимости различается. Иногда требуется просто наличие поддержки одного из нижестоящих протоколов, иногда требуется поддержка конкретного нижестоящего протокола.

Названия структурных единиц передаваемой информации в привязке к уровням модели OSI:
L1 -- сигналы (signals).
L2 -- *кадры* (frames).
L3 -- собственно *пакеты* (packets).
L4 + L5 -- *сегменты* (segments).
L6 + L7 -- *сообщения* (messages).

Фундаментальная задача СПД заключается в том, чтобы правильно передать сообщение.

Пакеты «возникают» начиная со второго уровня, хотя собственно пакетами традиционно называются пакеты, относящиеся к третьему уровню.

Первый и второй уровни часто совмещаются в рамках аппаратных технологий.

Четвертый и пятый уровни, равно как шестой и седьмой уровни, обычно реализуются «неразрывно» в рамках программных технологий.

Для обобщенной ссылки на порцию данных, над которой оперируют на некотором уровне, иногда используют термин PDU (Protocol Data Unit).

Понятно, что для правильной интерпретации пакета нужно его считать из канала полностью, причем с соблюдением последовательности. Если бы взаимодействующие станции работали бесконечно и находились в соответствующей степени готовности, то это не составляло бы особого труда. Но, поскольку станция-приемник может подключиться к каналу (да и вообще начать работать) в произвольный момент времени, возникает проблема, связанная с распознаванием флага начала пакета. Флаг начала пакета представляет собой зарезервированную цифровую последовательность, которая собственно позволяет станции-приемнику определить начало пакета. Проблема заключается в том, что такая же последовательность вполне может встретиться в пакете и после флага начала. Следовательно, возникает задача обеспечения уникальности флага начала пакета, то есть исключения этой последовательности из оставшейся части пакета.

Это достигается за счет действия, заключающегося в модификации следующей за флагом цифровой последовательности, которое в бит-ориентированных системах называется *бит-стаффингом* (bit stuffing), а в байт-ориентированных -- *байт-стаффингом* (byte stuffing).

При бит-стаффинге совпадающая с флагом последовательность разбивается с помощью вставки дополнительно бита с соответствующим значением. Применение бит-стаффинга приводит к увеличению длины пакета. Теоретически, с целью уменьшения связанных с бит-стаффингом «издержек», следует стремиться к минимизации количества вставок: разбивающий бит нужно вставлять после наиболее длинной уникальной подпоследовательности в флаговой последовательности.

Классическим флагом начала пакета является байт со значением 01111110b (7Eh). Оптимальная реализация бит-стаффинга при использовании классического флага

проиллюстрирована на рисунке.

Рисунок -- Пример реализации бит-стаффинга

На передающей стороне после нуля и шести единиц всегда вставляется седьмая единица, а на принимающей стороне единица после нуля и шести единиц всегда удаляется.

Цель байт-стаффинга полностью совпадает с целью бит-стаффинга. В сравнении с алгоритмами бит-стаффинга, алгоритмы байт-стаффинга манипулируют байтами, являются более сложными и более «затратными», но при программировании они позволяют избежать битовых операций (бит-стаффинг, в отличие от байт-стаффинга, обычно реализуется аппаратно).

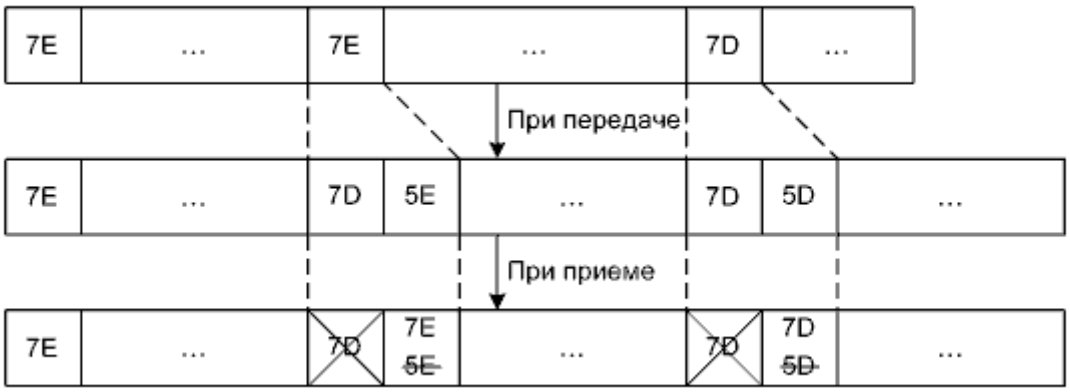

Рисунок -- Пример реализации байт-стаффинга

Единственным способом обеспечения уникальности флагового байта является замена совпадающего с ним байта на некий выбранный другой. Но возникает вопрос, как принимающая сторона отличит замененный байт от такого же незамененного. Решением является применение так называемого ESC-символа. Наличие ESC-символа говорит станции-приемнику о факте замены, а следующий за ESC-символом символ -- код замены позволяет определить какая замена была осуществлена. Байт-стаффингу можно подвергать целые группы символов.

Бит-стаффинг обычно применяется при задействовании синхронных СрПД, а байт-стаффинг -- асинхронных. Примерами технологий могут служить SDLC, HDLC, ISDN и другие (многие поддерживают как синхронные так и асинхронные СрПД). Примером протокола может служить PPP.

Следует отметить, что на практике (например, применительно к HDLC) бит-стаффинг выполняется вставкой нуля после пяти единиц.