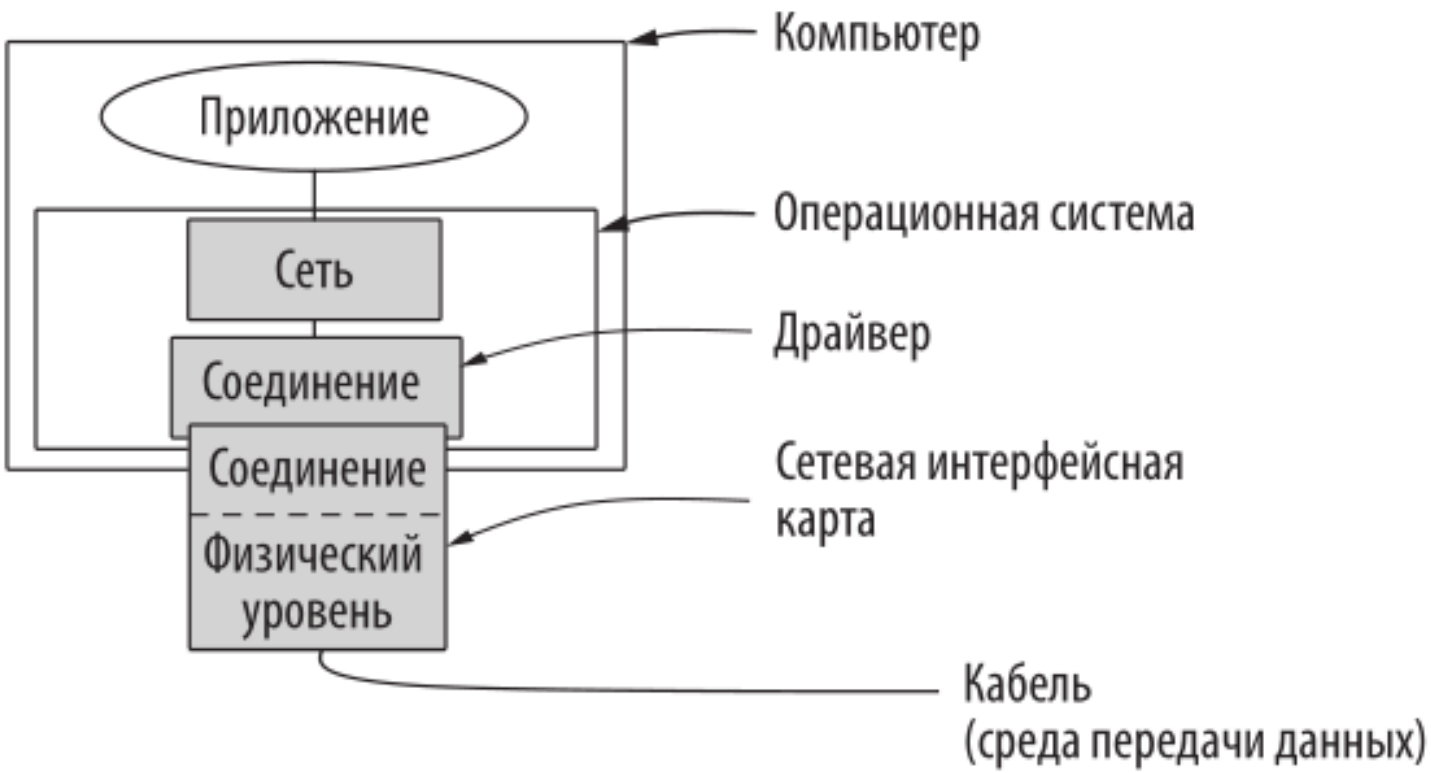
Канальный уровень 3

7 мая 2026 г.

Исходные упрощающие допущения

- Независимые процессы
- Однонаправленная передача данных
- Надежные устройства и процессы

Независимые процессы



Физический, канальный и сетевой уровни являются независимыми процессами, которые взаимодействуют между собой путем передачи сообщений.

Однонаправленная передача данных

Следующее ключевое допущение состоит в том, что устройство A хочет отправить на устройство B большой поток данных, используя надежную, ориентированную на установление соединений службу. Позднее мы рассмотрим случай, когда одновременно с этим B также хочет передать данные на A. Предполагается, что устройство A имеет бесконечный источник данных, готовых к отправке, и что ему никогда не нужно ждать их генерации. Когда канальный уровень A запрашивает данные, сетевой уровень их сразу предоставляет.

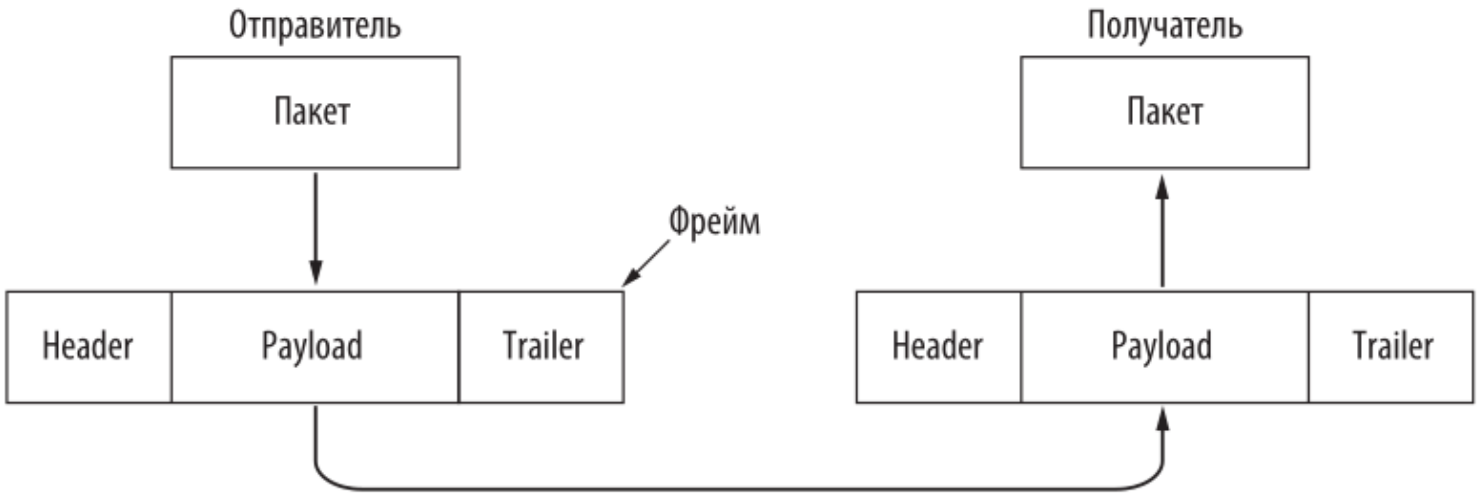
Надежные устройства и процессы

Также предполагается, что компьютеры не выходят из строя. При передаче могут возникать ошибки, но не проблемы, связанные с поломкой оборудования или случайной перезагрузкой.

Базовая схема передачи и приема данных

Получив пакет от сетевого уровня отправителя, канальный уровень формирует из него фрейм, добавляя заголовок и трейлер. Таким образом, фрейм состоит из встроенного пакета, некоторой служебной информации (в заголовке) и контрольной суммы (в трейлере). Затем фрейм передается канальному уровню целевого устройства.

Базовая схема передачи и приема данных



Базовая схема передачи и приема данных

Мы будем исходить из наличия подходящих библиотечных процедур, например to_physical_layer для отправки фрейма и from_physical_layer для его получения. Они вычисляют и добавляют или проверяют контрольную сумму (обычно это делается аппаратно), так что при разработке протоколов, представленных в этом разделе, можно об этом не беспокоиться. К примеру, они могут использовать рассмотренный ранее алгоритм CRC (циклический избыточный код).

Изначально получатель ничего не должен делать, он просто ожидает. В протоколах, рассматриваемых в этой главе, ожидание событий канальным уровнем происходит путем вызова процедуры wait_for_event(&event). Эта процедура возвращает управление, только когда что-то происходит (например, получение фрейма). При этом переменная event сообщает, что именно случилось.

Следует отметить, что на практике канальный уровень не находится в холостом цикле ожидания событий (согласно нашему допущению), а получает прерывание, когда это событие происходит. При этом он приостанавливает текущие процессы и обрабатывает полученный фрейм. Но для простоты мы проигнорируем эти детали и будем исходить из того, что канальный уровень все свое время посвящает работе с одним каналом.

Когда принимающая сторона получает фрейм, контрольная сумма вычисляется заново. Если она неверна (то есть при передаче возникли ошибки), канальный уровень получает соответствующую информацию (event=cksum_err). Если фрейм приходит без ошибок, канальному уровню об этом также сообщается (event=frame_arrival), после чего он может получить этот фрейм у физического уровня с помощью процедуры from_physical_layer.

Получив неповрежденный фрейм, канальный уровень проверяет управляющую информацию, находящуюся в заголовке, и если все в порядке, часть фрейма передается сетевому уровню; заголовок фрейма не передается.

```
#define MAX_PKT 1024 //determines packet size in bytes
typedef enum {false, true} boolean;
typedef unsigned int seq_nr; //sequence or ack numbers
typedef struct {unsigned char data[MAX_PKT];} packet;
typedef enum {data, ack, nak} frame_kind;

typedef struct { //frames are transported
                 // in this layer
        frame_kind kind; // what kind of frame is it?
        seq_nr seq; // sequence number
        seq_nr ack; // acknowledgement number
        packet info; // the network layer packet
} frame;

//Wait for an event to happen; return its type in even
void wait_for_event(event_type *event);
```

```c
/* Fetch a packet from the network layer for
  transmission on the channel. */
void from_network_layer(packet *p);

/* Deliver information from an inbound frame
        to the network layer. */
void to_network_layer(packet *p);

/* Go get an inbound frame from the physical layer
   and copy it to r. */
void from_physical_layer(frame *r);

/* Pass the frame to the physical layer
   for transmission. */
void to_physical_layer(frame *s);
```

```
// Start the clock running and enable the timeout even
void start_timer(seq_nr k);

// Stop the clock and disable the timeout event.
void stop_timer(seq_nr k);

/* Start an auxiliary timer and enable the ack
        timeout event. */
void start_ack_timer(void);

/* Stop the auxiliary timer and disable the ack
        timeout event. */
void stop_ack_timer(void);
```

```
/* Allow the network layer to cause a network
        layer ready event. */
void enable network layer(void);

/* Forbid the network layer from causing a network
        layer ready event. */
void disable network layer(void);

/* Macro inc is expanded in-line:
        increment k circularly. */
#define inc(k) if (k < MAX SEQ) k = k + 1; else k = 0
```

Тип seq_nr является целым без знака, используемым для нумерации фреймов, благодаря которой их можно различать. Нумерация идет от 0 до числа MAX_SEQ включительно, определяемого для конкретного протокола.

Тип packet — это единица информации, используемая при обмене данными между сетевым и канальным уровнями одного компьютера или двумя сетевыми уровнями. В нашей модели пакет всегда состоит из MAX_PKT байт, хотя на практике он обычно имеет переменную длину.

Тип frame содержит четыре поля: kind, seq, ack и info; первые три содержат управляющую информацию, а последнее может включать данные, которые необходимо передать. Три управляющих поля вместе называются **заголовком фрейма** (frame header).

Поле kind сообщает о наличии данных внутри фрейма, так как некоторые протоколы отличают фреймы, содержащие только управляющую информацию, от фреймов, в которых также есть данные.

Поле seq используется для хранения последовательного номера фрейма, ack — для подтверждения.

Поле данных фрейма info содержит один пакет. В управляющем фрейме это поле не задействуется. На практике используется поле info переменной длины, в управляющих фреймах оно полностью отсутствует.

Важно понимать взаимоотношения между пакетом и фреймом. Сетевой уровень создает пакет, принимая сообщение от транспортного уровня и добавляя к нему свой заголовок. Пакет передается канальному уровню, который включает его в поле info исходящего фрейма. Когда целевое устройство получает фрейм, канальный уровень извлекает пакет из фрейма и передает его сетевому. Таким образом, сетевой уровень может действовать так, будто устройства обмениваются пакетами напрямую.

Процедура wait_for_event представляет собой холостой цикл ожидания какого-либо события.

Процедура to_network_layer используется канальным уровнем для отправки пакетов сетевому, from_network_layer — для получения пакетов от него.

Процедуры ifrom_physical_layer и to_physical_layer служат для обмена фреймами между канальным и физическим уровнями.

Процедуры to_network_layer и from_network_layer относятся к интерфейсу между уровнями 2 и 3, а процедуры from_physical_layer и to_physical_layer — к интерфейсу между уровнями 1 и 2.

В большинстве протоколов предполагается использование ненадежного канала, который может случайно потерять целый фрейм. Чтобы избежать неприятных последствий, при отправке фрейма передающий канальный уровень запускает таймер. Если за установленный интервал времени ответ не получен, срок ожидания истекает и канальный уровень получает сигнал прерывания.

В приведенных здесь протоколах этот сигнал реализован в виде значения event=timeout, возвращаемого процедурой wait_for_event. Для запуска и остановки таймера используются процедуры start_timer и stop_timer соответственно.

Событие timeout может произойти, только если был запущен таймер, но еще не была вызвана процедура stop_timer. Процедуру start_timer разрешается запускать во время работающего таймера. Такой вызов просто перезапускает таймер, и отсчет начинается заново (до нового перезапуска или выключения).

Процедуры start_ack_timer и stop_ack_timer запускают и останавливают вспомогательные таймеры, используемые для создания подтверждений в некоторых ситуациях.

Процедуры enable_network_layer и disable_network_layer применяются в более сложных протоколах, где уже не предполагается, что у сетевого уровня всегда есть пакеты для отправки. Когда канальный уровень разрешает работу сетевого, последний может посылать сигнал прерывания, когда ему нужно передать пакет. Такое событие обозначается как event = network_layer_ready. Когда сетевой уровень отключен, он не может инициировать такие события. Канальный уровень тщательно следит за включением и выключением сетевого и не допускает ситуации, когда тот заваливает его пакетами, для которых нет места в буфере

Последовательные номера фреймов всегда находятся в пределах от 0 до MAX_SEQ включительно. Число MAX_SEQ отличается в разных протоколах. Для увеличения последовательного номера фреймов на 1 циклически (то есть с обнулением при достижении числа MAX_SEQ) используется макрос inc.

Определение простых операций в виде макросов (а не процедур) не снижает удобочитаемости программы, увеличивая при этом ее быстродействие.

Объявления из листинга выше входят во все последующие протоколы. В целях экономии места и для наглядности они были извлечены и собраны вместе, но, по сути, они должны быть объединены с протоколами. В языке C такое объединение производится путем размещения определений в специальном файле заголовка (в данном случае protocol.h) и включения их в файлы протокола с помощью **#include** — директивы препроцессора C.

Симплексные протоколы канального уровня

В данном разделе мы рассмотрим три простых протокола, из них каждый следующий способен справиться с более реалистичной ситуацией.

Протокол «Утопия»: без управления потоком и без исправления ошибок

В качестве первого примера мы рассмотрим самый простой протокол. Данные передаются только в одном направлении, а опасений, что где-то может произойти ошибка, даже не возникает. Сетевые уровни передающего и целевого устройств находятся в состоянии постоянной готовности. Время обработки минимально, размер буфера неограничен. А главное, линия связи между канальными уровнями никогда не теряет и не искажает фреймы. Он всего лишь демонстрирует базовую структуру, необходимую для построения настоящего протокола.

Протокол состоит из двух процедур, sender1 (отправитель) и receiver1 (получатель). Процедура sender1 работает на канальном уровне отправляющего устройства, а процедура receiver1 — на канальном уровне целевого. Ни последовательные номера, ни подтверждения не используются, поэтому MAX_SEQ не требуется. Единственным возможным событием является frame_arrival (то есть получение неповрежденного фрейма).

```
typedef enum {frame_arrival} event_type;
#include "protocol.h"
void sender1(void)
{
frame s; /* buffer for an outbound frame */
packet buffer; /* buffer for an outbound packet */
while (true) {
        from_network_layer(&buffer); /* go get somethi
                                        to send */
        s.info = buffer;             /* copy it into s
                                        for transmission */
        to_physical_layer(&s);       /* send it on its way
        }
}
```

Процедура sender1 представляет собой бесконечный цикл while, отправляющий данные на линию с максимально возможной скоростью. Тело цикла состоит из трех действий: получение пакета от сетевого уровня (всегда исправно работающего), формирование исходящего пакета с помощью переменной s и передача пакета адресату. «Утопия» использует только поле info, поскольку другие поля фрейма относятся к обработке ошибок и управлению потоком, а они в данном протоколе не применяются.

```
void receiver1(void)
{
frame r;
event_type event; /* filled in by wait, but not used
                     here */
while (true) {
        wait_for_event(&event); /* only possibility
                                   is frame_arrival */
        from_physical_layer(&r); /* go get the inbound
                                    frame */
        to_network_layer(&r.info); /* pass the data to
                                      the network layer */
        }
}
```

Процедура receiver1 так же проста. Вначале она ожидает, пока что-нибудь произойдет (как уже упоминалось, единственным событием в данном протоколе может быть получение неповрежденного фрейма). Когда фрейм приходит, процедура wait_for_event возвращает управление, при этом переменной event присваивается значение frame_arrival (которое все равно игнорируется). Вызов процедуры from_physical_layer удаляет вновь прибывший фрейм из аппаратного буфера и помещает его в переменную r. Наконец, порция данных передается сетевому уровню, а канальный уровень переходит в режим ожидания следующего фрейма.

Добавляем управление потоком: протокол с остановкой и ожиданием

Усложним задачу: предположим, отправитель посылает данные слишком быстро и получатель не успевает их обработать. В реальности такая ситуация может возникнуть в любой момент, поэтому крайне важно научиться ее предотвращать. Допущение об отсутствии ошибок в канале связи сохраняется. Линия остается симплексной.

Возможное решение проблемы — обратная связь со стороны получателя. Передав пакет сетевому уровню, он посылает источнику небольшое служебное сообщение, разрешающее отправку следующего фрейма. Отправитель, отослав фрейм, должен ждать этого разрешения. Подобная задержка — простейший пример протокола с управлением потоком.

Протоколы, в которых отправитель посылает один фрейм, после чего ожидает подтверждения, называются **протоколами с остановкой и ожиданием** (stop-and-wait).

Хотя пересылка данных в этом примере осуществляется по симплексному принципу, по направлению от отправителя получателю, на практике фреймы идут и в обратную сторону. Следовательно, линия связи между двумя канальными уровнями должна поддерживать двунаправленную передачу. Однако данный протокол диктует жесткое чередование направлений пересылки: источник и получатель отправляют фреймы строго по очереди. Для такой реализации хватило бы полудуплексного физического канала.

```
typedef enum {frame_arrival} event_type;
#include "protocol.h"
void sender2(void)
{
frame s;                    /* buffer for an outbound fram
packet buffer;              /* buffer for an outbound pack
event_type event;           /* frame_arrival is the onply
                               possibilty */
while (true) {
        from_network_layer(&buffer); /* go get somethi
                                        to send */
        s.info = buffer;            /* copy it into s
                                       for transmission */
        to_physical_layer(&s);      /* bye-bye little fram
        wait_for_event(&event);
        }
}
```

Как и в протоколе 1, в начале цикла отправитель извлекает пакет с сетевого уровня, формирует из него фрейм и отправляет фрейм по линии связи. Отличие в том, что теперь он должен ждать получения фрейма с подтверждением, прежде чем запустить новую итерацию цикла и обратиться к сетевому уровню за следующим пакетом. В данной модели канальный уровень отправителя даже не просматривает входящий фрейм, поскольку он всегда означает только одно: подтверждение.

```
void receiver2(void)
{
frame r, s;              /*buffers for frames */
event_type event; /* frame_arrival is the only
                              possilbilty*/
while (true) {
        wait_for_event(&event); /* only possibility
                                   is frame_arrival */
        from_physical_layer(&r); /* go get the inbound
                                   frame */
        to_network_layer(&r.info); /* pass the data to
                                   the network layer */
        to_physical_layerf(&s);  /* send a dummy frame
                                   to awaken sender*/
        }
}
```

Единственное отличие между процедурами receiver2 и receiver1 состоит в том, что после передачи пакета сетевому уровню receiver2 посылает подтверждение обратно отправителю, после чего идет на следующую итерацию цикла. Поскольку для отправителя важно само прибытие ответного фрейма, а не его содержание, то получателю не нужно заполнять его специальной информацией.

Добавляем исправление ошибок: порядковые номера и протокол ARQ

Теперь рассмотрим реальную ситуацию: канал связи, в котором могут быть ошибки. Фреймы могут либо повреждаться, либо теряться. Однако мы будем считать, что если фрейм был изменен при передаче, то аппаратное обеспечение целевого устройства определит это, подсчитав контрольную сумму.

На первый взгляд может показаться, что можно улучшить протокол 2, добавив таймер. Получатель будет возвращать подтверждение только в случае получения правильных данных. Неверные пакеты будут просто игнорироваться. Через некоторое время у отправителя истечет интервал времени и он отправит фрейм еще раз. Этот процесс будет повторяться до тех пор, пока фрейм наконец не прибудет в целости.

В этой схеме есть недосток. Рассмотрим следующий сценарий.

1. Сетевой уровень устройства A передает пакет 1 своему канальному уровню. Пакет доставляется в целости на устройство B и передается его сетевому уровню. B посылает фрейм подтверждения обратно на A.
2. Фрейм подтверждения полностью теряется в канале связи. Он просто не попадает на устройство A. Все было бы намного проще, если бы терялись только информационные, но не управляющие фреймы, но, к сожалению, канал связи не способен их различать.
3. У канального уровня устройства A внезапно истекает отведенный интервал времени. Не получив подтверждения, оно предполагает, что отправленный им фрейм с данными был поврежден или потерян, и посылает его еще раз.
4. Дубликат фрейма в целости прибывает на канальный уровень B и передается на сетевой уровень. В итоге часть файла, переданного с A на B, дублируется. Копия файла на устройстве B будет неверной, и ошибка не будет обнаружена, другими словами, протокол даст сбой.

Разумеется, необходим некий механизм, с помощью которого получатель смог бы различать новые фреймы и переданные повторно.

Наиболее очевидное решение — нумерация фреймов. Отправитель указывает порядковый номер фрейма в его заголовке. Благодаря этому принимающее устройство отличает новый фрейм от дубликата, который необходимо проигнорировать.

Необходимо, чтобы протокол выполнялся без ошибок, а нумерация не занимала много места в заголовке фрейма, поскольку соединение должно использоваться эффективно. Возникает вопрос: каково минимальное количество битов, достаточное для порядкового номера фрейма?

Единственная неопределенность в данном протоколе может возникнуть между фреймом m и следующим за ним фреймом m + 1. Если m потерян или поврежден, получатель не подтвердит его и отправитель повторит передачу.

Когда он будет успешно принят, получатель отправит подтверждение. Именно здесь находится источник потенциальной проблемы. В зависимости от наличия подтверждения отправитель дублирует фрейм m или передает новый фрейм m + 1.

На стороне отправителя событием, запускающим передачу фрейма m + 1, является получение подтверждения доставки фрейма m. Но это означает, что фрейм m – 1 уже передан и подтверждение его доставки отправлено и получено. В противном случае протокол не стал бы посылать новый фрейм. Следовательно, неопределенность может возникнуть только между двумя соседними фреймами

Таким образом, для нумерации фрейма достаточно всего одного бита информации (со значением 0 или 1).

В каждый момент времени получатель ожидает прибытия фрейма с определенным порядковым номером. Фрейм с верным номером принимается, передается сетевому уровню, затем отправляется подтверждение его получения. Номер следующего ожидаемого фрейма увеличивается по модулю 2 (то есть 0 становится 1, а 1 — 0). Фрейм с неверным номером отбрасывается как дубликат. Однако последнее подтверждение повторяется, чтобы сообщить отправителю, что фрейм получен полностью.

```
#define MAX_SEQ 1
/* must be 1 for protocol 3 */
typedef enum {frame_arrival, cksum_err, timeout}
        event_type;
#include "protocol.h"
void sender3(void)
{
seq_nr next_frame_to_send;  /* seq number of next
                               outgoing frame */
frame s;                    /* scratch variable */
packet buffer;              /* buffer for an outbound
                               packet */
event_type event;
next_frame_to_send = 0;     /* initialize outbound
                               sequence numbers */
from_network_layer(&buffer);  /* fetch first packet */
```

```
while (true) {
s.info = buffer;    /* construct a frame for
                       transmission */
s.seq = next_frame_to_send;  /* insert sequence
                                number in frame */
to_physical_layer(&s);  /* send it on its way */
start_timer(s.seq);  /* if answer takes too long,
                        time out */
wait_for_event(&event);  /* frame arrival, cksum err,
                            timeout */
if (event == frame_arrival) {
        from_physical_layer(&s);  /* get the
                                     acknowledgement */
        if (s.ack == next_frame_to_send) {
                stop_timer(s.ack);  /* turn the timer
                                       off */
                from_network_layer(&buffer); /* get th
                                                next one to se
                inc(next_frame_to_send); /* invert nex
```

Протокол, в котором отправитель ожидает положительного подтверждения, прежде чем перейти к пересылке следующего фрейма, часто называется PAR (Positive Acknowledgement with Retransmission — **положительное подтверждение с повторной передачей**) или ARQ (Automatic Repeat reQuest — **автоматический запрос повторной передачи**).

Протокол 3 отличается от своих предшественников тем, что и отправитель, и получатель содержат переменную, значение которой хранится, пока канальный уровень находится в режиме ожидания.

Отправитель запоминает номер следующего фрейма в переменной next_frame_to_send, а получатель записывает номер следующего ожидаемого фрейма в переменной frame_expected.

После передачи фрейма отправитель запускает таймер и ждет какого-либо события. Возможны три ситуации: либо придет неповрежденный фрейм подтверждения, либо будет получен поврежденный фрейм подтверждения, либо просто истечет интервал времени.

В первом случае отправитель возьмет у сетевого уровня следующий пакет и разместит его в буфере поверх предыдущего, а также увеличит порядковый номер фрейма.

Если же прибудет поврежденный фрейм подтверждения или время истечет, то ни буфер, ни номер не будут изменены и будет отправлен дубликат фрейма. В любом случае затем посылается содержимое буфера (следующий пакет либо дубликат предыдущего).

Когда неповрежденный фрейм прибывает к получателю, проверяется его номер. Если это не дубликат, то фрейм принимается и передается сетевому уровню, после чего формируется подтверждение.

Дубликаты и поврежденные фреймы на сетевой уровень не передаются, но при их получении подтверждается прибытие последнего правильного фрейма, благодаря чему отправитель понимает, что нужно перейти к следующему фрейму или повторить передачу поврежденного.