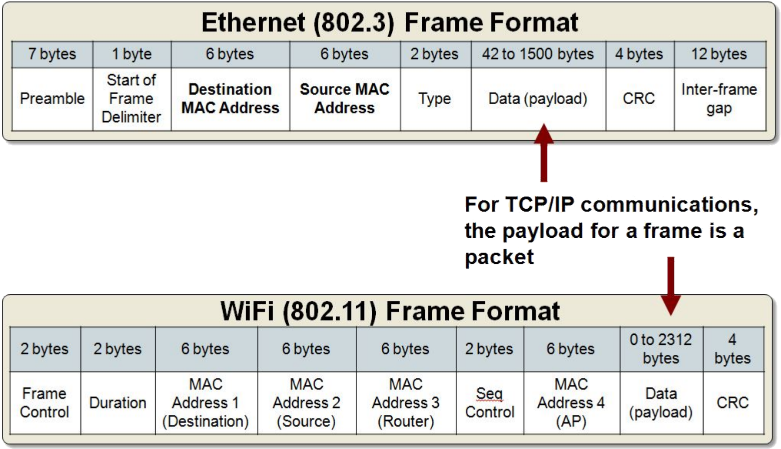
CS3200: Computer Networks

Lecture 10

IIT Palakkad

20 Aug, 2019

Link Layer Frame



Ethernet (802.3) Frame Format

7 bytes	1 byte	6 bytes	6 bytes	2 bytes	42 to 1500 bytes	4 bytes	12 bytes
Preamble	Start of Frame Delimiter	**Destination MAC Address**	**Source MAC Address**	Type	Data (payload)	CRC	Inter-frame gap

For TCP/IP communications, the payload for a frame is a packet

WiFi (802.11) Frame Format

2 bytes	2 bytes	6 bytes	6 bytes	6 bytes	2 bytes	6 bytes	0 to 2312 bytes	4 bytes
Frame Control	Duration	MAC Address 1 (Destination)	MAC Address 2 (Source)	MAC Address 3 (Router)	Seq Control	MAC Address 4 (AP)	Data (payload)	CRC

Figure: Link layer frame structure of Ethernet and WiFi.

Some Definitions

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

Some Definitions

```
typedef struct {
 frame_kind kind;
 seq_nr seq;
 seq_nr ack;
 packet pkt;
} frame;

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

/* Fetch a packet from the network layer for transmission on t
void from_network_layer(packet *p);

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

Some Definitions

```
/* Go get an inbound frame from the physical layer and copy it
void from_physical_layer(frame *f);

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

/* Start the clock running and enable the timeout event. */
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);
```

Some Definitions

```
/* 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
```

A Utopian Simplex Protocol

Assumptions

- Data are transmitted in one direction only.
- Both the transmitting and receiving network layers are always ready.
- Channel is error-free
- Zero processing times
- Infinite buffer

A Utopian Simplex Protocol

```
typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
 frame fs;
 packet pkt;

 while (true) {
  from_network_layer(&pkt);
  fs.pkt = pkt;
  to_physical_layer(&fs);
 }
}
```

A Utopian Simplex Protocol

```
void receiver1(void)
{
 frame fr;
 event_type event;

 while (true) {
  wait_for_event(&event);
  from_physical_layer(&fr);
  to_network_layer(&fr.pkt);
 }
}
```

A Simplex Stop-and-Wait Protocol

Assumptions

- Data are transmitted in one direction only.
- Both the transmitting and receiving network layers are always ready.
- Channel is error-free

A Simplex Stop-and-Wait Protocol

```
typedef enum {frame arrival} event type;
#include "protocol.h"

void sender2(void)
{
 frame fs;
 packet pkt;
 event_type event;

 while (true) {
  from_network_layer(&pkt);
  fs.pkt = pkt;
  to_physical_layer(&fs);
  wait_for_event(&event);
 }
}
```

A Simplex Stop-and-Wait Protocol

```
void receiver2(void)
{
 frame fr, fs;
 event_type event;

 while (true) {
  wait_for_event(&event);
  from_physical_layer(&fr);
  to_network_layer(&fr.pkt);
  to_physical_layer(&fs);
 }
}
```

A Simplex Stop-and-Wait Protocol for Noisy Channels

Assumptions

- Data are transmitted in one direction only.
- Both the transmitting and receiving network layers are always ready.

An ARQ/PAR Protocol

ARQ (Automatic Repeat reQuest) or PAR (PositiveAcknowledgement w

```
#define MAX SEQ 1
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void sender3(void)
{
 seq_nr next_frame_to_send;
 frame fs, fr;
 packet pkt;
 event_type event;
 next_frame_to_send = 0;
 from_network_layer(&pkt);

 while (true) {
  fs.pkt = pkt;
  fs.seq = next_frame_to_send;
  to_physical_layer(&fs);
```

An ARQ/PAR Protocol

```
  start_timer(s.seq);
  wait_for_event(&event);

  if (event == frame_arrival) {
   from_physical_layer(&fs);
    if (fs.ack == next_frame_to_send) {
     stop_timer(fs.ack);
     from_network_layer(&pkt);
     inc(next_frame_to_send);
    }
   }
 }
}
```

An ARQ/PAR Protocol

```
void receiver3(void)
{
 seq_nr frame_expected;
 frame fr, fs;
 event_type event;
 frame_expected = 0;

 while (true) {
  wait_for_event(&event);
```

An ARQ/PAR Protocol

```
  if (event == frame_arrival) {
   from_physical_layer(&fr);
  if (r.seq == frame_expected) {
   to_network_layer(&fr.pkt);
   inc(frame_expected);
  }
   fs.ack = 1 - frame expected;
   to_physical_layer(&fs);
  }
 }
}
```

Sliding Window Protocols

Need for bi-directional data transmissions. Seperate link for each direction. Is this efficient?

The *kind* field in the header of an incoming frame, the receiver can tell whether the frame is data or an acknowledgment.

Acknowledgment is attached to the outgoing data frame (using the *ack* field in the frame header). This is known as **piggybacking**.

Sliding Window Protocols

- Each outbound frame contains a sequence number, ranging from 0 up to $2^n - 1$, for some $n \geq 1$

- The sender maintains a set of sequence numbers corresponding to frames it is permitted to send. These frames are said to fall within the **sending window**.

- The receiver also maintains a **receiving window** corresponding to the set of frames it is permitted to accept.