Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Nom et prénom					
1	2	3			Total
NOTE					

DeSEm DeSEm

Examen DeSEm 2017-2018

11 juin 2018

Durée : 120 minutes

1 (20)

Ingénierie du logiciel embarqué, design patterns

Soit le petit système suivant :

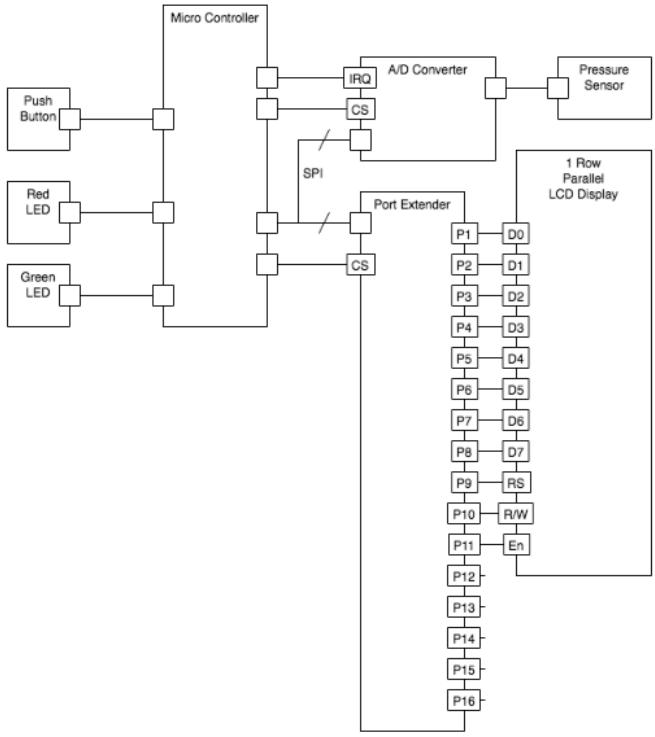


Figure 1 : Un système simple

Dans ce système qui a pour but le monitoring d'une pression, un microcontrôleur est connecté à quelques périphériques dont un display LCD parallèle 16x1 et un convertisseur A/D. Comme ces deux périphériques sont pilotés par la même interface SPI, ils ont chacun besoin d'une ligne « chip select ». Le convertisseur A/D dispose de plus d'une ligne d'interruption pour indiquer au contrôleur que la mesure est accomplie et qu'il peut lire la valeur digitalisée. Le bouton déclenche la mesure. La LED rouge est alors allumée et reste dans cet état jusqu'à la fin de la mesure. A ce moment, la LED rouge est éteinte et la LED verte clignote trois fois pour indiquer la fin de la mesure.

Questions :

1. Identifiez les classes pour les paquets Board et HAL pour le système illustré en Figure 1. Etablissez une liste avec les noms des classes pour chaque paquet.

Object name	Class Name

2. Représentez un diagramme de classes avec paquets, classes et relations pour l'entier du système de la Figure 1. Utilisez entre autres le pattern HAL-Board. Appliquez, si nécessaire, un pattern de découplage entre les classes de la couche Board et la classe représentant le contrôleur. Il n'est pas nécessaire d'indiquer les méthodes. Utilisez aussi le pattern Singleton lorsque c'est approprié.
3. Créez le diagramme de classe de la Factory pour le système de la Figure 1.

2 (25)

Ingénierie du logiciel embarqué, développement de protocole

Un jeu TRON basé sur une communication sans fil doit être implémenté sur une carte avec processeur M4F et une radio 2.4 GHz. Un nœud peut se connecter au serveur TRON. Dès que deux nœuds sont connectés au serveur, le serveur envoie une trame BEACON contenant des positions aléatoires pour les deux joueurs ainsi qu'un octet qui indique le démarrage de la partie. Trois secondes après cette trame, le jeu commence. La vitesse est constante et identique pour les deux joueurs. Elle reste la même pendant une partie. Ces trois secondes passées, le serveur envoie une trame BEACON toutes les 30 [ms]. Cette trame contient la position des deux adversaires. A la réception de cette trame, les écrans des deux nœuds sont mis à jour et les nœuds répondent avec une trame DATA N*10 [ms] après la réception du BEACON. N est un nombre entre 0 et 1 et il est attribué aux joueurs dans la réponse à leur demande de connexion au serveur. La trame DATA contient la position actuelle du joueur. Cette position sera distribuée par le BEACON suivant. Un joueur peut en tout moment abandonner le jeu en se déconnectant du serveur. Si le serveur détecte une collision, il termine le jeu en mettant un octet spécifique dans le BEACON. La figure suivante montre une copie d'écran pendant une partie dans laquelle le joueur bleu a perdu :

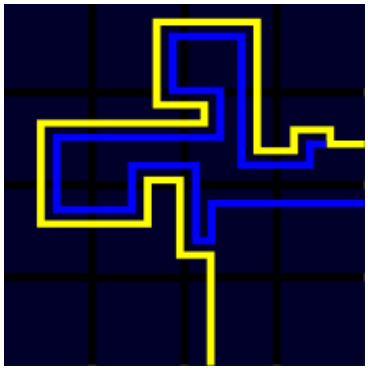


Figure 2 : Copie d'écran du jeu TRON

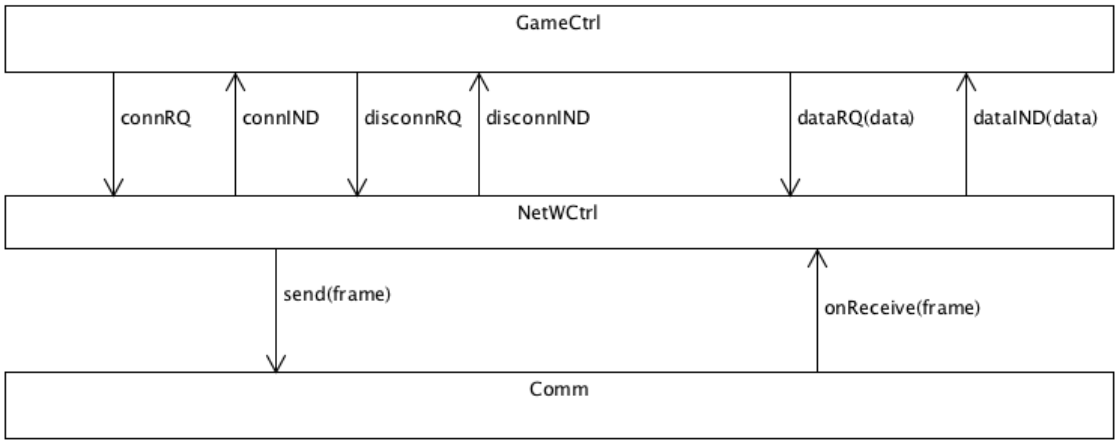


Figure 3 : Architecture de communication d'un nœud TRON.

Le diagramme de séquence ci-dessous présente un exemple de messages échangés.

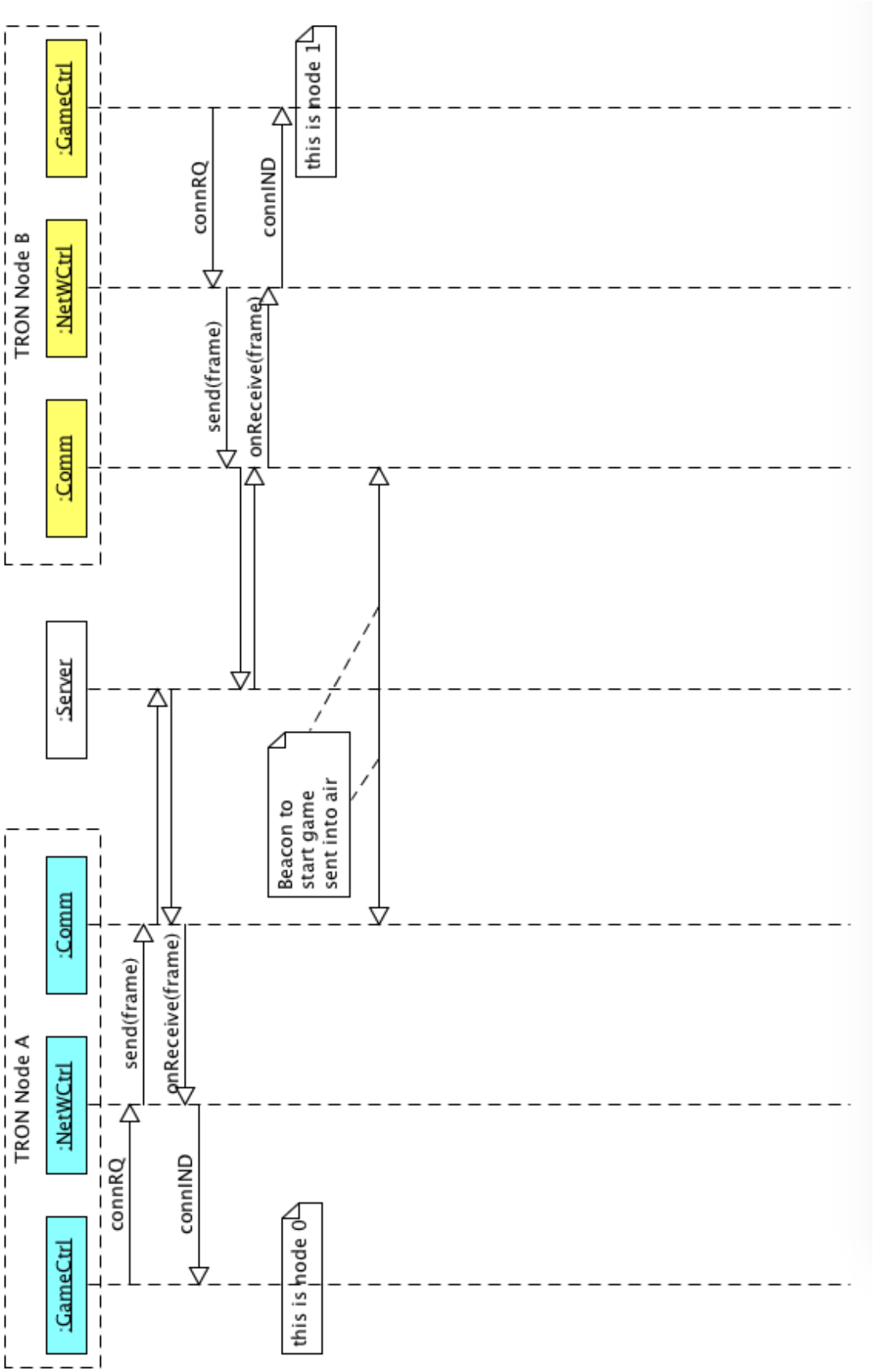


Figure 4 : Les messages de connexion et l'envoi de la trame qui démarre le jeu

Pour découpler la couche applicative de la couche de communication, on prescrit l'utilisation du pattern **`SAP`** décrit et explicité dans le cours.

Questions :

1 1. Proposez un nom de paquet pour la partie application et un nom de paquet pour la partie communication. Des noms possibles ont été proposés dans le cadre du cours DeSEm.

2 2. A côté de la classe `GameCtrl`, proposez deux autres classes importantes pour faire abstraction du jeu TRON.

2 3. A côté de la classe `NetWCtrl`, proposez deux autres classes importantes pour faire abstraction de la communication.

5 4. Dessinez un diagramme de classes UML avec toutes les classes `GameCtrl`, `NetWCtrl`, les paquetages et les classes proposées.
N'oubliez pas d'y intégrer le pattern `SAP` du cours. Il n'est pas demandé d'indiquer les méthodes des classes !

2 5. Esquissez une trame BEACON. Respectez la description donnée de cette trame dans l'énoncé.

2 6. Esquissez une trame DATA.

2 7. Esquissez une trame envoyée lors d'une demande de connexion.

2 8. Esquissez une trame envoyée lors d'une demande de déconnexion.

7 9. Complétez le diagramme de séquence en annexe, en y ajoutant les messages échangés jusqu'à l'envoi du quatrième BEACONS par le serveur.

3 (10)

Ingénierie des protocoles

Des nœuds équipés de capteurs sont reliés par radio à un concentrateur connecté à un réseau filaire. Dans le contexte de cet exercice, on considère que seuls les nœuds capteurs sont producteurs de données de mesure. Ils ne reçoivent que des trames de contrôle.

Les nœuds capteurs ne sont pas synchronisés : ils peuvent transmettre des trames à tout moment.

Chaque nœud capteur a une adresse unique **`sensorId`** codée sur 16 bits. Le concentrateur n'a pas d'adresse.

On considère un modèle en quatre couches : Physical, MAC, Transport et Application (voir Figure).

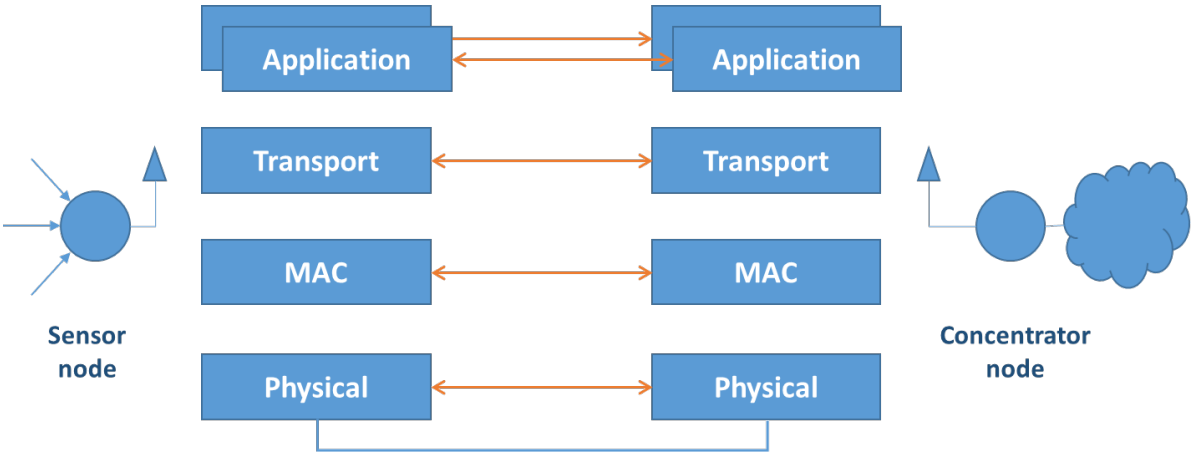


Figure 5 : Pile de communication

On dispose d'une entité implémentant un protocole Physical radio avec les services et primitives de services suivants :

```
PH_TRX.Request(ON | OFF)
// Turn on/off the radio transceiver
PH_DATA.Request(buffer)
PH_DATA.Indication(buffer)
PH_DATA.Confirm(status)
// Confirm that data in PH_DATA.Request has been sent
// (consider only "OK" status)
```

Lorsqu'une entité de la couche Application demande l'envoi de données, le protocole MAC applique la procédure suivante :

- Les données sont encapsulées dans une trame **`DATA`**.
- La trame **`DATA`** est transmise sans vérification.
- Une fois que la trame **`DATA`** a été transmise, la couche MAC attend la réception d'une trame pendant un temps **`AckWait`**.
- Si la trame reçue est une trame **`ACK`** (accusé de réception), la couche Transport est notifiée du succès de la transmission et le transceiver est déclenché.
- Dans les autres cas, on recommence l'étape 2 après un délai aléatoire **`RandWait`**.

La couche MAC dispose des primitives de service suivantes :

```
MAC_DATA.request(frameData)
MAC_DATA.confirm() // Confirm that an ACK from peer has been received
```

Deux timers sont définis :

Timer **RandWait**:**startTmRandWait**, **stopTmRanWait** Timeout event: **evTmRandWait**
Timer **AckWait**: **startTmAckWait**, **stopTmAckWait** Timeout event: **evTmAckWait**

La couche Transport attend **MAC_DATA.confirm()** avant un prochain **MAC_DATA-request()**.

La couche Transport permet à plusieurs entités de la couche Application de communiquer en parallèle. Concrètement elle permet à une entité Application :

- d'établir une connexion (service confirmé localement),
- de terminer une connexion (service non-confirmé), et
- de transporter des données (service non-confirmé).

Chaque entité Application dispose d'un identifiant **appID** (16 bits) unique au niveau d'un nœud capteur.

Questions :

Les questions se rapportent au nœud capteur (et non au concentrateur).

4 1. Dessinez la machine d'état du protocole de la couche MAC selon les spécifications ci-dessus.

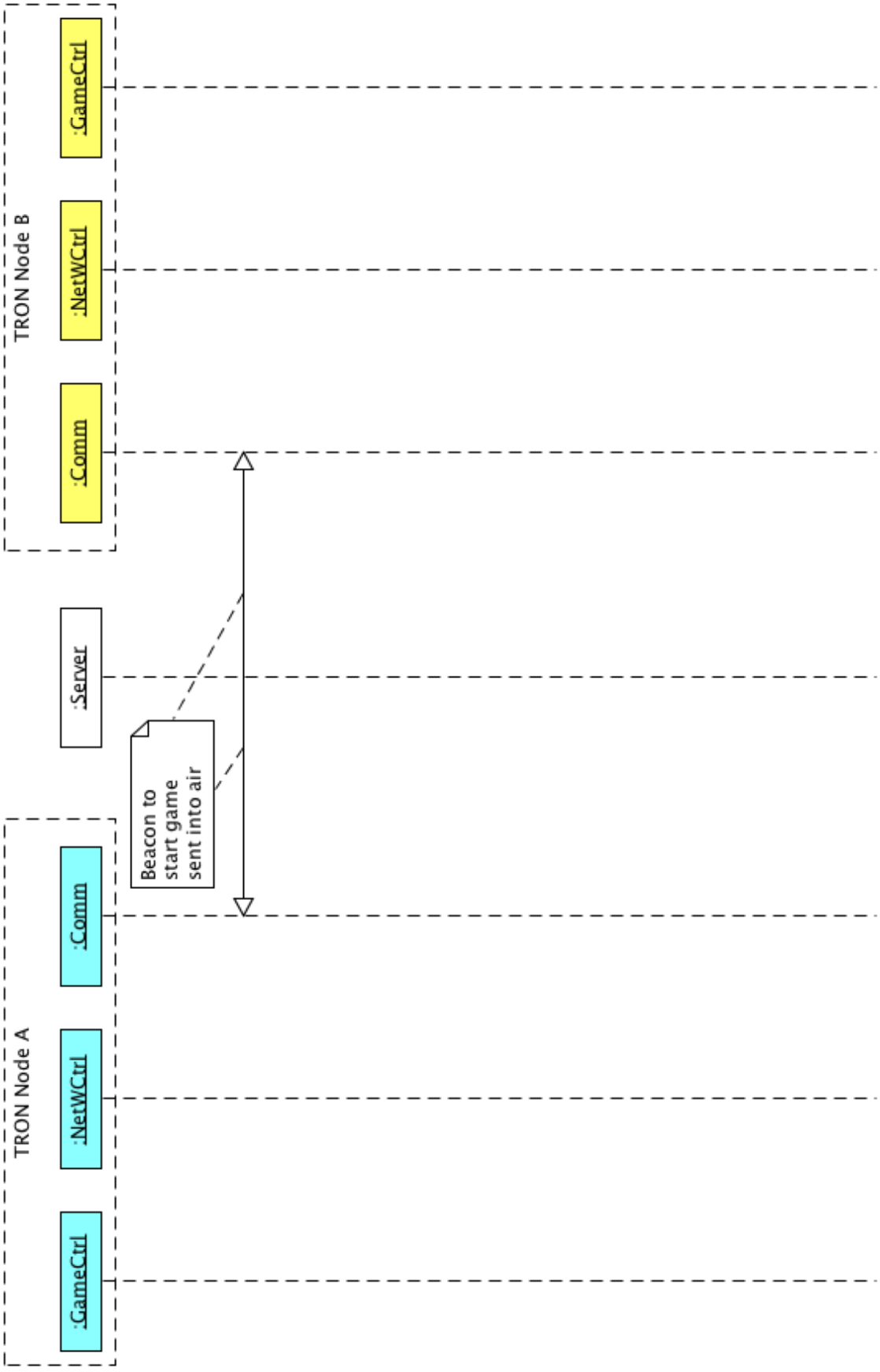
1 2. Proposez un format pour les trames de la couche MAC.

1 3. Indiquez les services avec leurs primitives et leurs arguments pour la couche Transport.

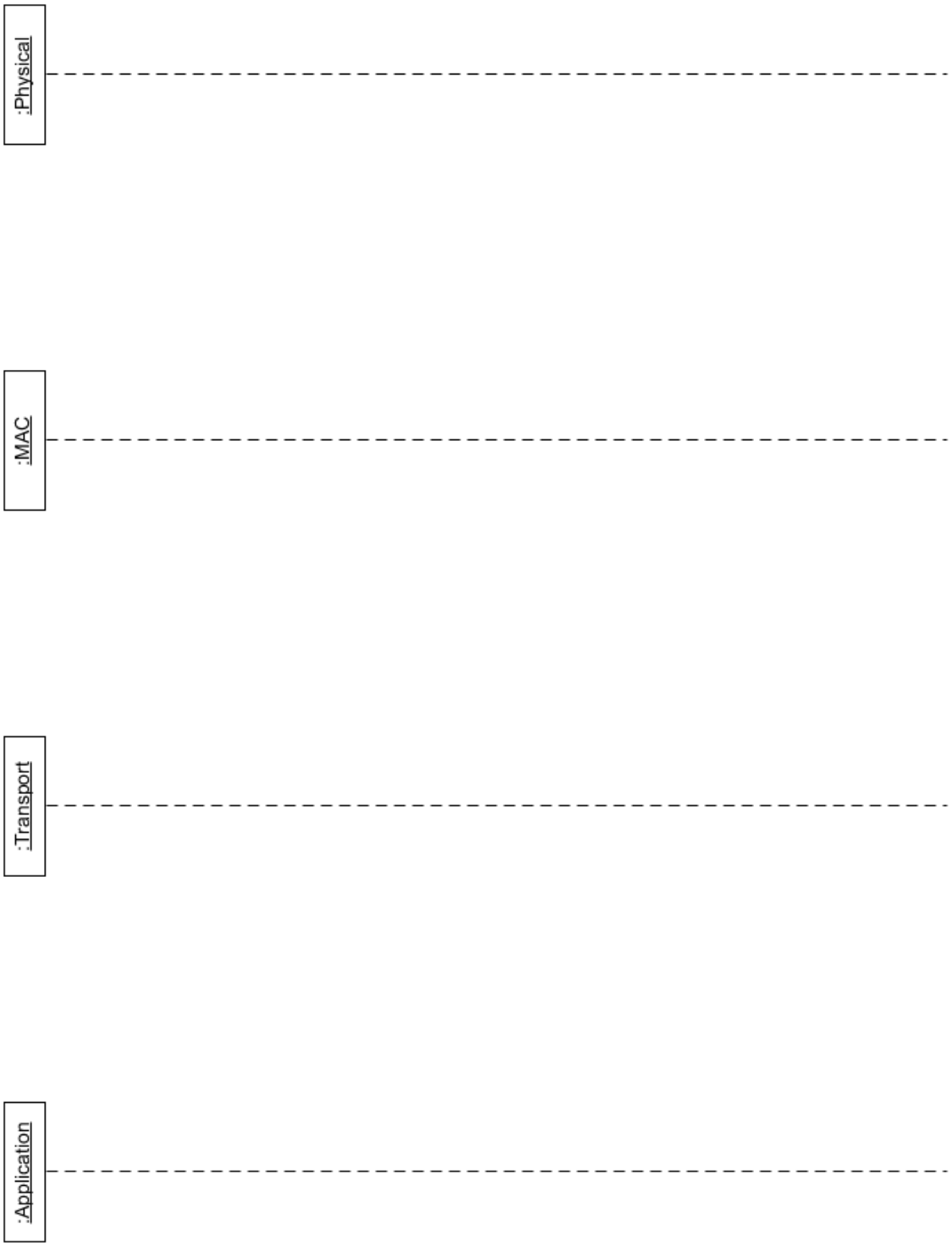
1 4. Proposez un format pour les paquets de la couche Transport.

3 5. Représentez la phase de connexion d'une entité Application sur le diagramme de séquence annexé. Ne considérez que le scénario « sunny day ». Indiquez explicitement les trames émises et reçues.

Annexe – Exercice 2



Annexe – Exercice 3



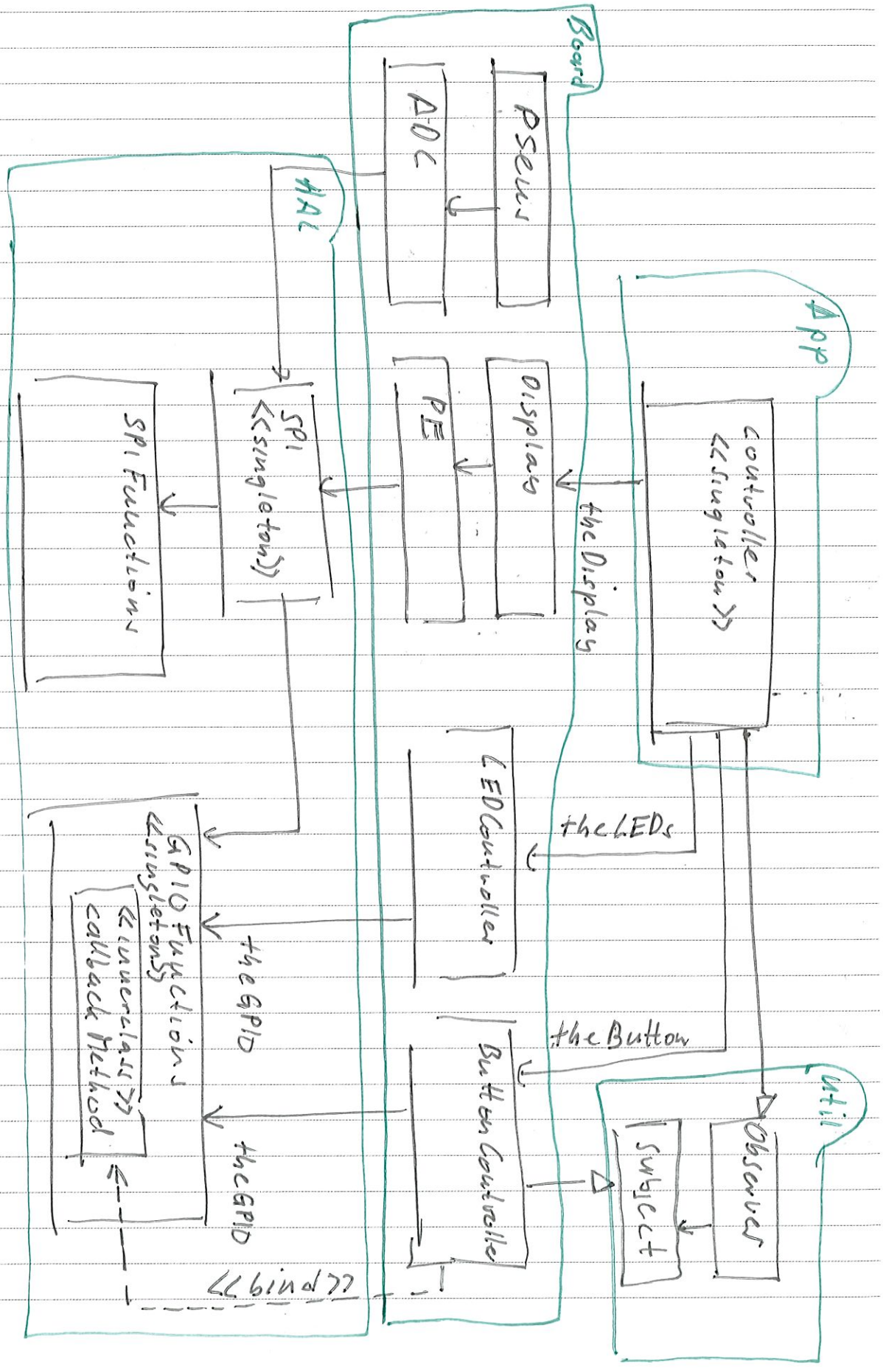
5 pt

1a)

Object	Classe
Bus SPI (2x)	HAL \ SPI
GPIO (6x)	HAL \ GPIO
ADC (1x)	Baard \ ADC
PExt (1x)	Board \ PE
Pressure Sensor (1x)	Board \ PSens
Display (1x)	Board \ Display
Button (1x)	Board \ Button
LED (2x)	Board \ LED

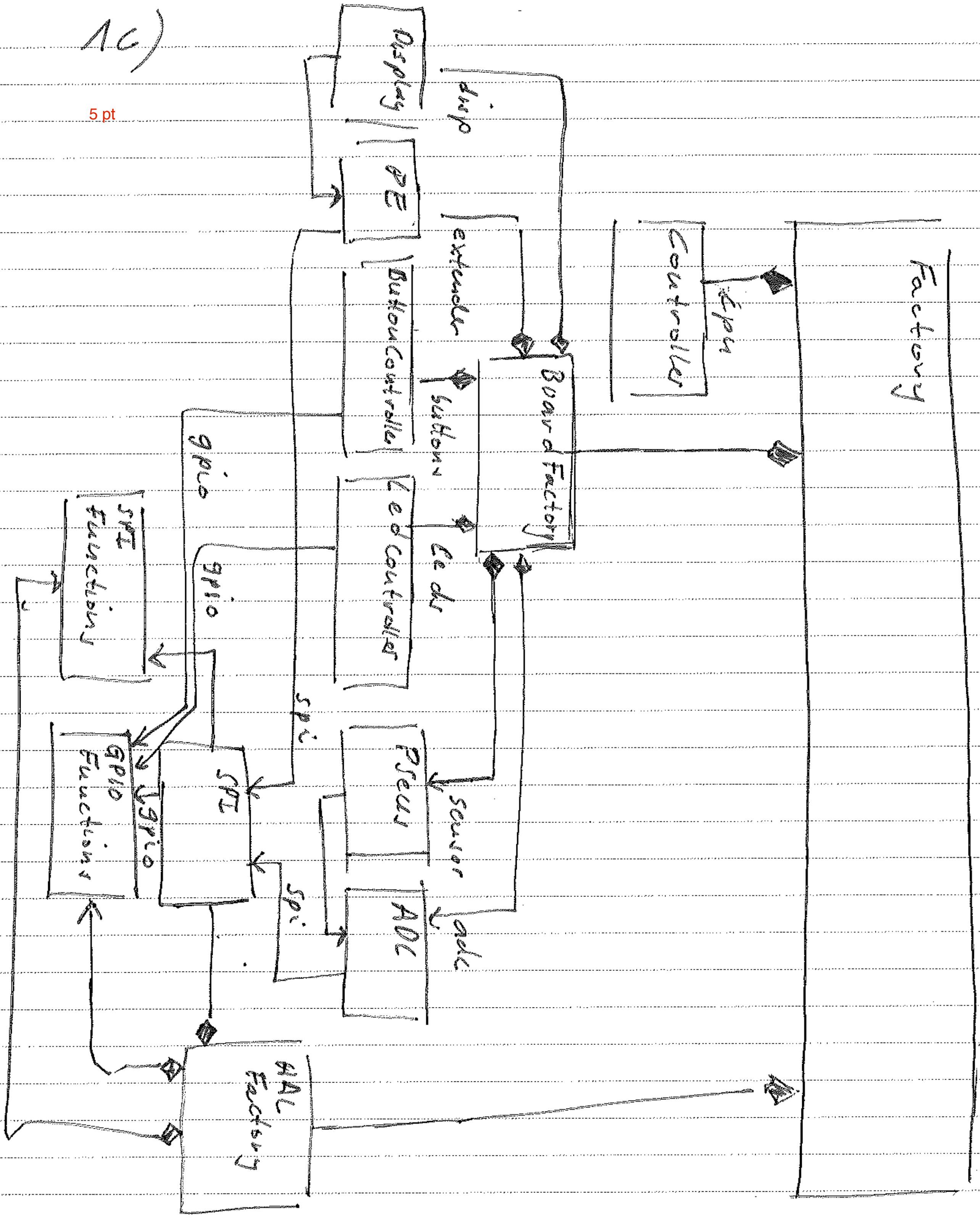
16)

10 pt


Board
Pseus
ADC
HAL
SPI
<<singleton>>
SPI Functions
Display
PE
App
Controller
<<singleton>>
theDisplay
LEDController
theLEDs
GPIO Functions
<<singleton>>
<<universal>>
callback Method
theGPIO
ButtonController
theButton
<<bind>>
Util
Observer
Subject

1c)

5 pt



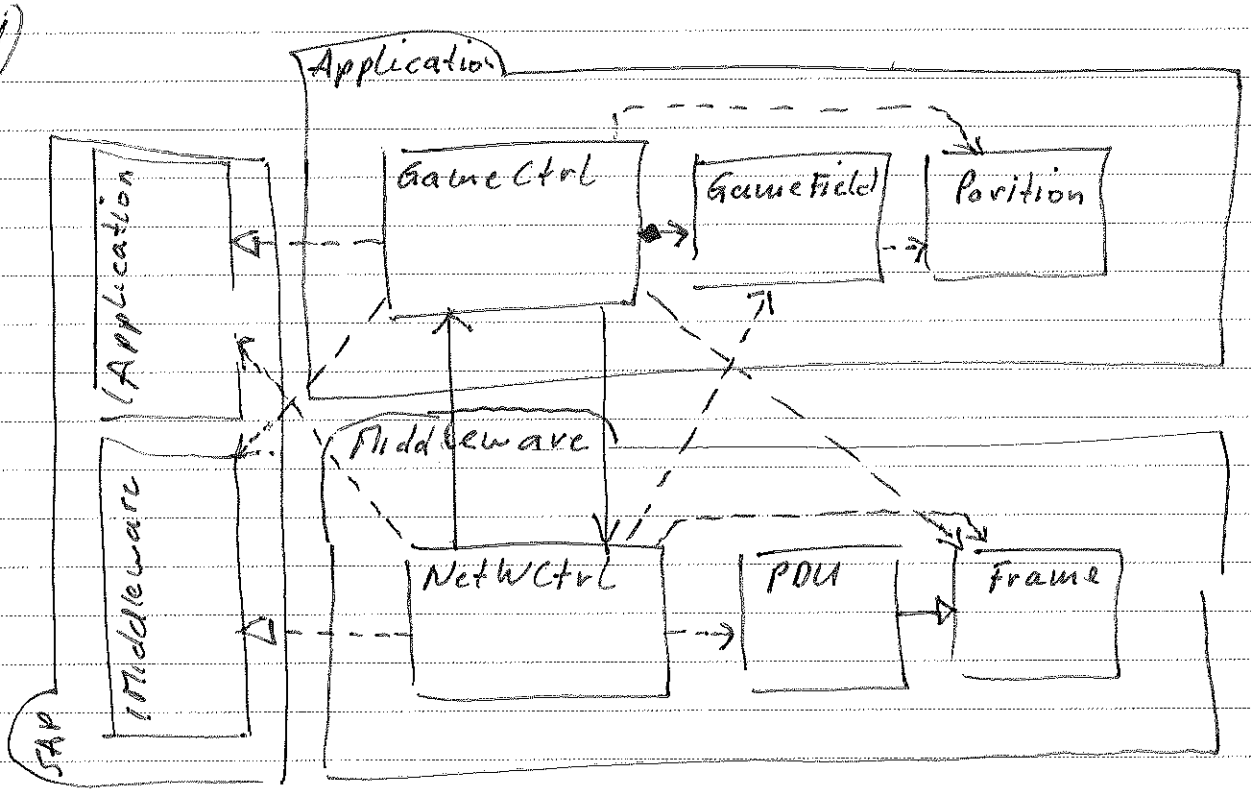
Display
PE
ButtonController
LedController
BoardFactory
Controller
Factory
PSensor
ADC
SPI
GPIO Functions
SPI Functions
HAL Factory
disp
extender
button
led
sensor
adc
spi
gpio

1 pt 2a) application → Application
communication → Middleware

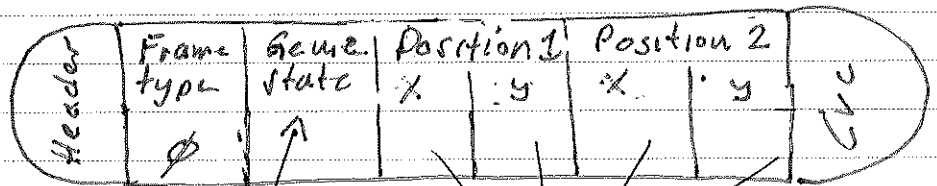
2 pt 2b) GameCtrl, gameField, Position

2 pt 2c) NetWCtrl, Frame, PDU

5 pt 2d)



2 pt 2e)



BEACON

1 octet

0 = WAIT
1 = RUNNING
2 = OVER

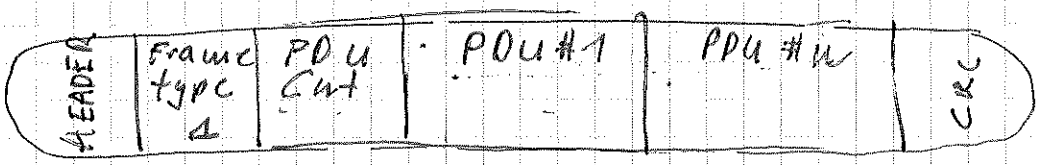
1 octet

Value (1 octet)

2 pt

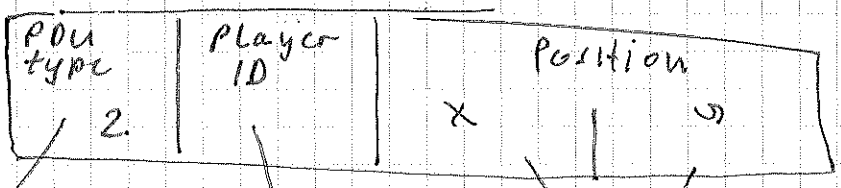
2 f)

DATA



1 octet

PDU



0 = connRQ
1 = disconRQ
2 = data

1 octet

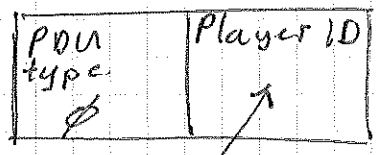
0 Player 1
1 Player 2

1 octet

Value (1 octet)

2 pt

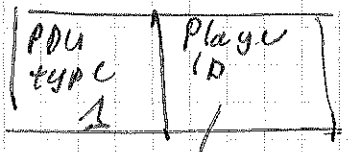
2 g)



ConnRQ = -1
ConnInd → Player ID given by Server

2 pt

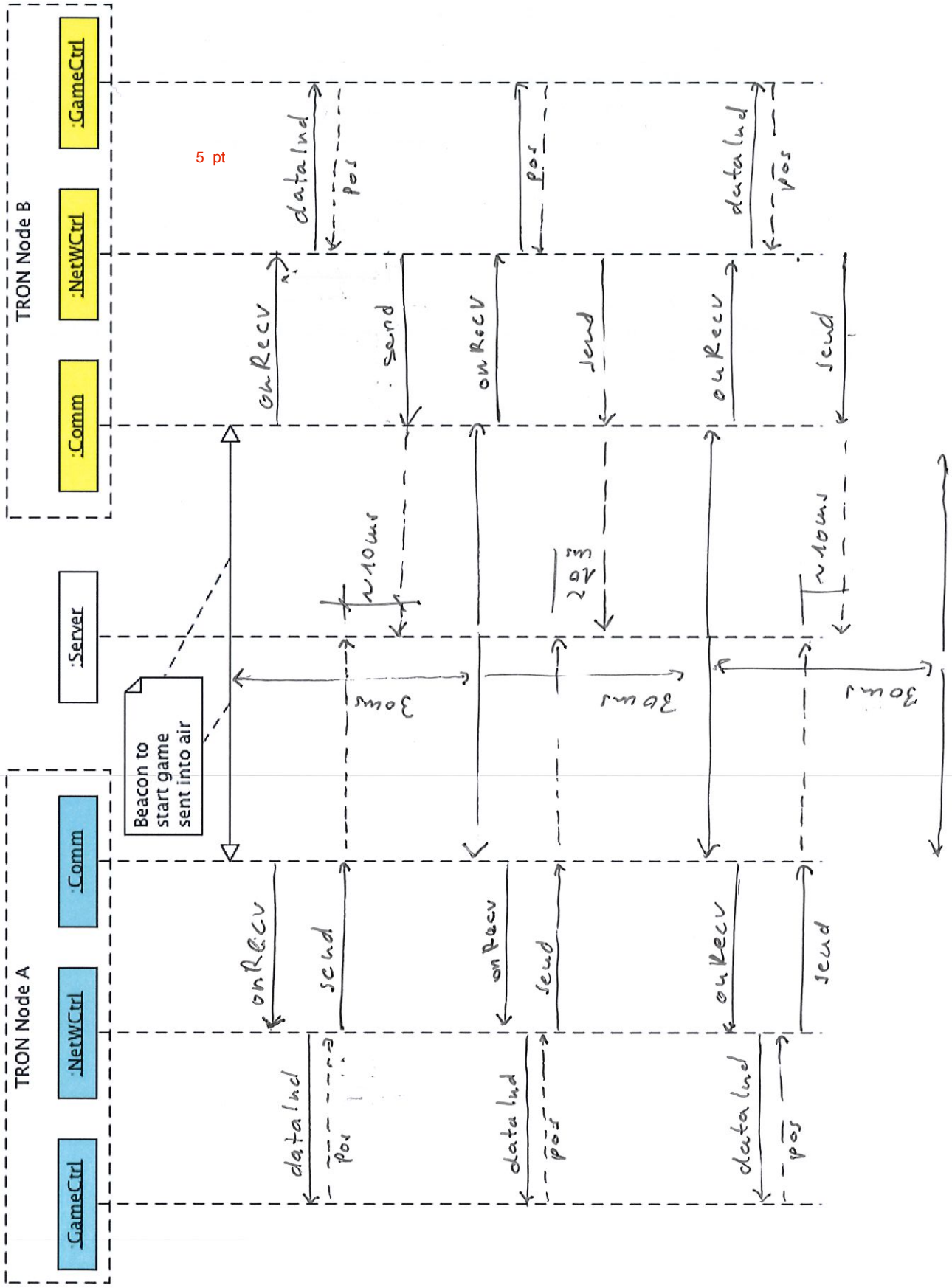
2 h)



{0, 1} Player that wants to disconnect

same for RQ + IND

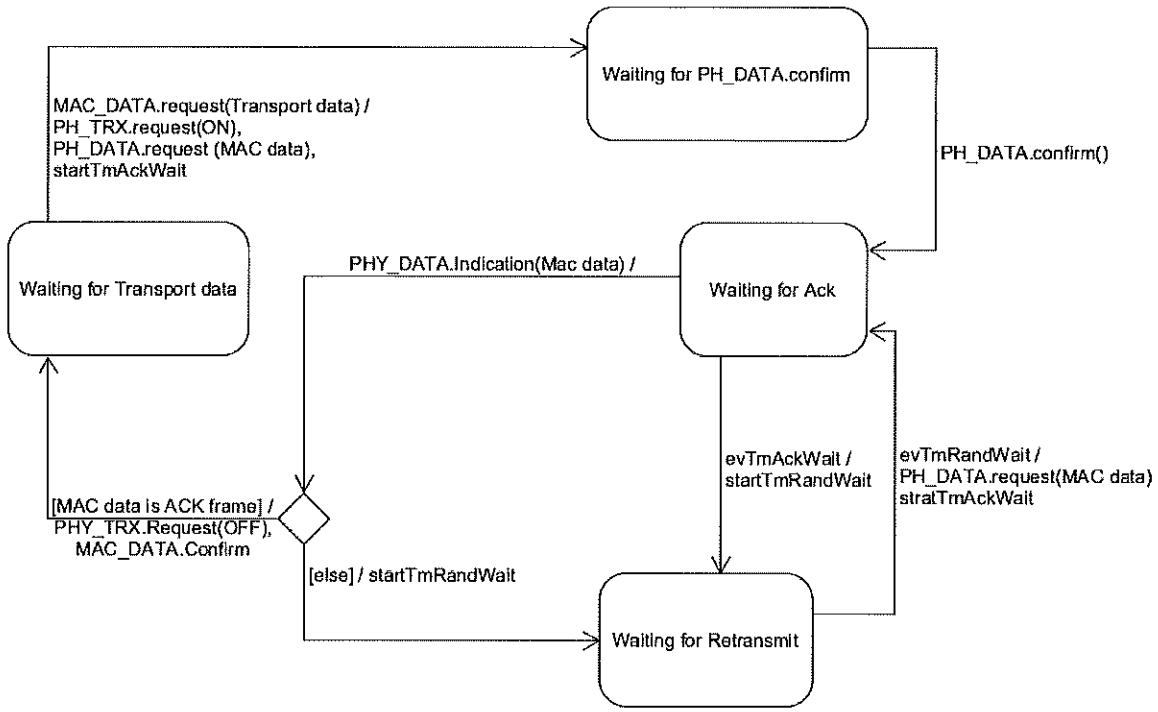
Annexe – Exercice 2



Solution

1.

4 pt



2. 1 pt

sensorId (16 bits)	frame type [0: DATA, 1: ACK] (1 byte)	frame data (for DATA frame) (n bytes)	CRC (2 bytes)

3. 1 pt

T-CONNECT.request(appID)

T-CONNECT.confirm()
(T-CONNECT.indication() and T-CONNECT.response only for concentrator)

T-DISCONNECT.request
(T-DISCONNECT.indication() only for concentrator)

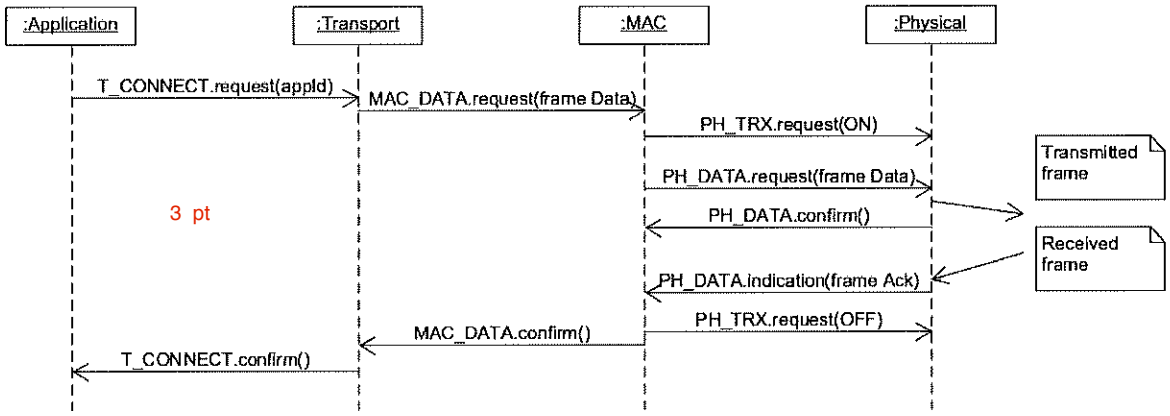
T_DATA.request(application data)
(T-DATA.indication(application data) only for concentrator)

4.

1 pt

appId	application data
(2 bytes)	(x bytes)

5.



3 pt