Protocol Design
Sensor Mesh Protocol

CS544 - Computer Networks
Professor Mike Kain - Spring 2016

Group 10
David Flanagan, Michael Wilson, Jaidev Ramakrishna, Jugal Lodaya

Change History

The SMP protocol design has been changed to use IP Multicast instead of Broadcast to provide migratability into IPv6 and flexibility. Broadcast would fail outside of a local intranet whereas Multicast allows us to use the SMP protocol over the internet.

Additionally, the NTP time synchronization service that we originally proposed to run under the SMP protocol was removed. It was decided that the application layer should determine whether or not the timestamp of messages is critical to its operation, and if so which server to use for time synchronization. The job of the protocol is to deliver the most recently received packet, and not to perform time sync.

The Keep Alive packet payload to changed from None to the id of the publisher/subscriber sending it.

In response to our first submission, we added in additional details about the String ID and the Error code to show how many bytes they will be.

We also clarified the CRC method and created our own CRC code following the IEEE 803.2 error checking algorithm. The Central Node does the CRC checking and we added this to the Security section

Table of Contents

Protocol Design Sensor Mesh Protocol	0
Change History	1
Table of Contents	2
List of Figures	4
Terms and Definitions	5
Service Description	7
Message Definition (PDU)	9
AddPub	10
RemovePub	10
AddSub	10
RemoveSub	11
Success	11
StartPublishing	12
StopPublishing	12
KeepAlive	13
DFA	14
Publisher	14
Subscriber	15
Extensibility	16
Security	17

List of Figures

Figure 1	Service Model	**6**
Figure 2	The Control PDU format	**7**
Table 1	Exhaustive list of implemented commands	**8**
Figure 3	Success packet with port number payload	**10**
Table 2	Exhaustive list of error codes	**10**
Figure 4	Failure message format	**10**
Figure 5	Format of the Data PDU	**12**
Figure 6	Publisher DFA	**13**
Figure 7	Subscriber DFA	**14**

Terms and Definitions

Command Code	32-bit identifier included with each Control PDU.
Control Node	The central server. Syncs all other nodes and sets up and tears down sensor connections.
Control PDU	Packet used to communicate between the control node and publishers/subscribers.
Data PDU	Packet broadcast by publishers to subscribers.
Leaf Node	Any publisher or subscriber, all nodes except the central node.
Mesh	The entire network of sensors including all publishers, subscribers and the central node.
Node	Single entity in the sensor mesh.
Publisher	Node that broadcasts data periodically.
Sensor Type	Field in data packet that designates sensor type (a sensor type of 4 may be associated with a Camera Sensor which will have a well defined data format).
String ID	Identifier of a publishing node on the mesh. It can be a max 4 bytes (i.e. a string id of a camera-type publisher may be “Nikon_D7000_03”).
Error Code	Integer representing the error code. There is room for 4 bytes in the payload for this code, but it will only probably ever be 1 byte max.
Subscriber	Node that passively accepts data from publisher.
Terminate	Final command in a transaction, A command that does not expect a response.
Transaction	An entire interaction between a leaf node and the central node, it must end in a Success message, or a Failure message.

Transaction ID	Unique ID given to each transaction by the initiator.
CRC	Cyclical Redundancy Checksum
ID	Identifier
PDU	Protocol Data Unit
SMP	Sensor Mesh Protocol
UDP	User Data Protocol

Service Description

The Sensor Mesh Protocol (SMP) provides a service to produce and receive real-time sensor data using IP Multicast. The SMP protocol is implemented using a client-server architecture over UDP for control plane and peer-to-peer for data plane. UDP provides the best performance for this protocol because it has the least amount of overhead and limited packet size. The SMP protocol is designed for applications that require real-time data with limited overhead. SMP provides no flow control or packet retransmission but does provide data integrity. Because it is designed for real-time applications, packets with errors are simply dropped. The protocol is designed to provide service to a network of processes which need to send and receive real-time sensor data for a particular system application, such as robotics, Internet of Things applications, and other embedded systems.

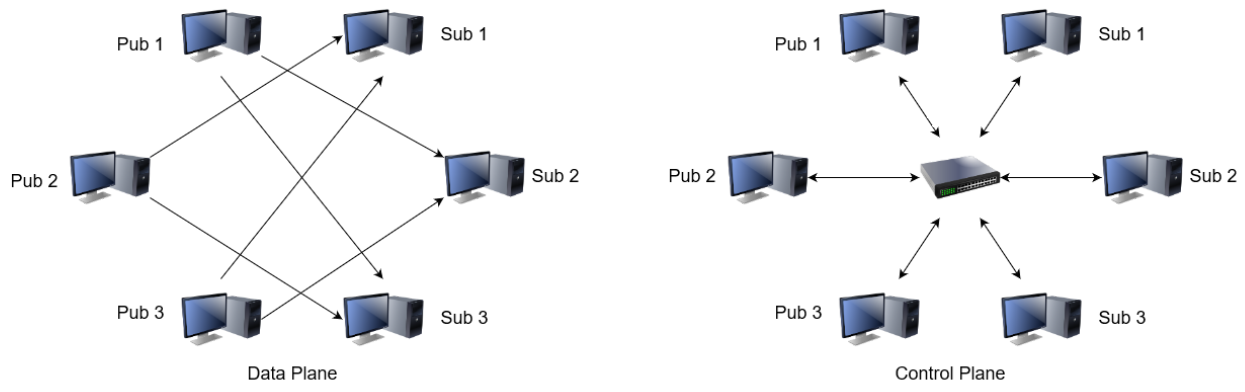

Figure 1 - Service Model.

A central server process provides services for registering a sensor data publisher. Similarly, sensor data subscribers use the central server node to retrieve network information for receiving from sensor data publisher nodes. The protocol definition removes the responsibility of monitoring data senders and receivers from the individual nodes and provides those nodes with network information necessary for unidirectional data flow. Data is sent directly to receivers and not multiplexed through the central server which limits overhead and provides scalability by avoiding a bottleneck at the central server. Real-time multi-process systems which utilize a multicast configured network can benefit from the limited overhead and simple client API service provided by the Sensor Mesh Protocol.

The central server process runs on a dedicated port, 15001. The protocol is defined to run with only one central node process on the network. The IP Multicast group chosen for the SMP protocol is *224.3.29.71*. The ports *15002* to *15555* are dedicated for the publishers to multicast data. These ports are managed by the single central node. Publishers and subscribers maintain open sockets with the central node for command messages but no data is sent to the central node. Requests to the central node to publish data are hashed using a unique identifier and the request is granted with a port address to publish data over the network. Similarly, a request to subscribe to a data publisher through the central node will be hashed using the same unique string identifier. The request to subscribe should be a success with a payload containing the port address for receiving.

A KeepAlive command is used to periodically inform the SMP central node that a client is still active. This requires the subscribers to send a KeepAlive command periodically to the central node. The data publishers must also periodically send a KeepAlive command to the central node in order to maintain the status recorded for the data publisher. This way if a data publisher were to terminate, the central node would know not to successfully grant data subscribers with bad data publishers network information. Similarly, if a publisher has no subscribers the central node will inform it to start sending data when the first subscriber registers.

Message Definition (PDU)

The SMP Protocol Data Units (PDUs) consist of two different packets. One is used for communication with the control node and the other is used for broadcasting data. The PDU used for communicating with the control node we will refer to as the “Control PDU” and the PDU that publishers use to broadcast data we will refer to as the “Data PDU”.

The Command PDU (Shown in **Figure 2)** consists of a 8-bit command code, a 16-bit transaction ID, 8-bits of reserved space, a 32-bit Sensor Type, a data payload of up to 32 bytes and a 32-bit Cyclical Redundancy Checksum (CRC) field. Depending on the command the payload, which is always an ASCII string unless otherwise specified, may be between 0 and 32 bytes. The string is always used to specify the broadcasting node the command is referencing. The CRC field is calculated using the IEEE 802.3 CRC error checking algorithm, and is used to ensure data integrity.

1 Byte	2 Bytes	1 Byte	4 Bytes	1-32 Bytes	4 Bytes
Cmd Code (UINT8)	Transaction ID (INT16)	Reserved (8 bits)	Sensor Type (UINT32)	Payload (String 32 bytes Max)	CRC (UINT32)

Figure 2 - The Control PDU format.

The Transaction ID acts as a session token. The node that initiates a control transaction should generate the ID. The value of the ID does not matter, but each transaction thereafter should be incremented or decremented by 1. To avoid confusion when both the central node and a leaf node start a transaction simultaneously all transactions that start on the central node should have negative transaction IDs, all transactions that start on a leaf node should have positive transaction IDs. Transaction IDs can be recycled once all other possibilities in the range have been used up.

The sensor type is used to let the subscriber know if the data adheres to a known format. If the data doesn’t adhere to one of the prespecified known formats then this field should be set to zero. This field is added for extensibility and convenience so a subscriber can know the format of a sensor’s data.

Name	Direction	Code	Payload
KeepAlive	Pub/Sub -> Serv	0	String ID
AddPub	Pub -> Serv	1	String ID
RemovePub	Pub -> Serv	2	String ID
AddSub	Sub -> Serv	3	String ID
RemoveSub	Sub -> Serv	4	String ID
Success	Serv -> Pub/Sub	5	None/Port
Failure	Serv -> Pub/Sub	6	Error Code
PubRemoved	Serv -> Sub	7	String ID
StartPublishing	Serv -> Pub	8	String ID
StopPublishing	Serv -> Pub	9	String ID

Table 1 - Exhaustive list of implemented commands.

Table 1 shows all of the possible commands and their respective command codes. We will go through each one of them in a bit more detail here.

AddPub

The AddPub command is sent to the control node from a publisher it is used to register a sensor publisher with the mesh network. It must contain a unique string identifier. In the event that the string identifier already exists as a publisher the central node returns the AlreadyExists failure.
Possible return values are:

1. Success - The sensor publisher was successfully added to the mesh.
2. Failure - The request was received but no string identifier was sent, the CRC check failed or a publisher with the same name already exists.

RemovePub

The RemovePub command is sent from a publisher to the control node. It is used to unregister a sensor publisher. It must contain a unique string identifier. Possible return values are:

1. Success - The sensor publisher was successfully removed form the mesh.
2. Failure - The request was received but no string identifier was sent, the CRC check failed, the requesting node does not own the publisher or the publisher does not exist.

AddSub

The AddSub command is sent to the control node from a subscriber it is used to register a sensor subscriber with the mesh network. It must contain a unique string identifier. In the event that a node tries to subscribe to a publisher he is already subscribed to the control node should return Success.
Possible return values are:

1. Success - The sensor subscriber was successfully added to the mesh.
2. Failure - The request was received but no string identifier was sent, the CRC check failed or a publisher with the given identifier does not exist.

RemoveSub

The RemoveSub command is sent to the control node from a subscriber it is used to unregister a sensor subscriber with the mesh network. It must contain a unique string identifier. Possible return values are:

1. Success - The sensor subscriber was successfully removed from the mesh.
2. Failure - The request was received but no string identifier was sent, the CRC check failed or a sensor with that identifier does not exist.

Success

The Success command is sent from the control node to publishers and subscribers. The success command indicates that the last command sent to the control node was accepted and correct. This is a terminating command there are no possible return values. The success command has an optional parameter that is the port number of the sensor referenced in the transaction belongs on if there is one. This parameter is mainly used in response to an AddSub or AddPub command to let the leaf nodes know what port they should be using. In all other cases the payload should be empty. The port number should be a 16-bit unsigned integer as shown in **Table 1**.

1 Byte	2 Bytes	1 Byte	4 Bytes	4 Bytes	4 Bytes
Cmd Code (UINT16)	Transaction ID (INT16)	Reserved (8 bits)	Sensor Type (UINT32)	Payload (UINT32) Port number	CRC (UINT32)

Figure 3 - Success packet with port number payload.

Failure

The Failure command can be sent in either direction depending on the case. This command should have a payload of an error code to let the receiver know the reason for the failure. An exhaustive list of failure codes is shown in **Table 2.** The format of the Failure command is shown in **Figure 4.**

Error Code	**Meaning**
1	Invalid Command
2	CRC Check Failure
3	Indicated publisher does not exist
4	Indicated publisher already exists.
5	Permission Error, attempt to manipulate publisher not owned by sending node.

Table 2 - Exhaustive list of error codes.

1 Byte	2 Bytes	1 Byte	4 Bytes	4 Bytes	4 Bytes
Cmd Code (UINT8)	Transaction ID (INT16)	Reserved (8 bits)	0	Payload (INT32) Error code	CRC (UINT32)

Figure 4 - Failure message format.

PubRemoved

The PubRemoved command is sent from the control node to all subscribers of a publishing node when the control node receives a RemovePub request from the publisher. This is done so subscribers know they can stop listening for incoming data. Possible return values are:

1. Success - The subscriber node acknowledges that his publisher has stopped communicating.
2. Failure - The request was received but no string identifier was sent or the CRC check failed.

StartPublishing

The StartPublishing command is sent from the control node to any publisher when the number of subscribers to that publisher transitions from 0 to 1 or more. This is done so a publisher does not congest the network if noone is listening. Possible return values are:

1. Success - The publisher node acknowledges that he now has subscribers and should start communicating.
2. Failure - The CRC check failed.

StopPublishing

The StopPublishing command is sent from the control node to any publisher when the number of subscribers to that publisher transition from 1 or more to 0. This is done so a publisher does not congest the network if no one is listening. Possible return values are:

1. Success - The publisher node acknowledges that he no longer has subscribers and should stop communicating.
2. Failure - The CRC check failed.

KeepAlive

The KeepAlive command is sent by all leaf nodes to the control node at least every 1s but not more than once every 500ms. This is a way for each node to check in with control node so if a subscriber or publisher should fail, the control node knows to react to it. Possible return values are:

1. Failure - The CRC check failed.

The Data PDU (Shown in **Figure 5**) consists of a UINT64, a payload of 1 to 2^16 bytes and a four byte CRC32 calculated in the same manner as the Control PDU. Each data packet should be stamped with the synced system time of the leaf node in milliseconds when the data was harvested. Note that there is no sensor string ID present in the data packet. That is because the subscriber should determine what sensor this message comes from by the port it is received on. Likewise the sensor type is known in the same way.

8 bytes	4 bytes	0 bytes to 65516 bytes
Time In Millis (UINT64)	CRC (UINT32)	Payload (Varies: Max Size of 2^16 - 8 - 4 - UDP header (8))

Figure 5 - Format of the Data PDU.

DFA

Publisher

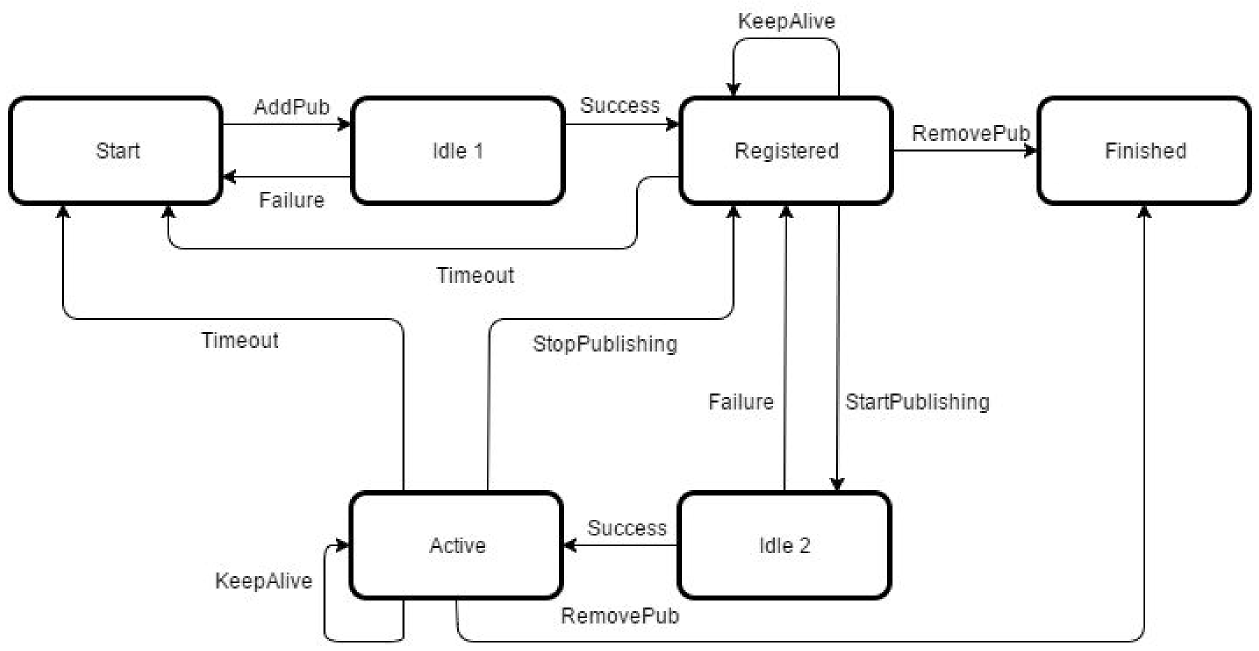

Figure 6 - Publisher DFA.

- **Start –** It is the initial state. The Publisher returns to it in the event of a timeout, or if the AddPub message results in an error.
- **Idle 1 –** The Publisher goes into this state while waiting for the Server's reply to its AddPub message.
- **Registered –** The Publisher reaches this state after a successful AddPub. In this state, it is not actively publishing data, but keeps itself registered with the Server by sending KeepAlive packets at fixed intervals.
- **Idle 2 –** The Publisher reaches this state from Registered when it receives the StartPublishing message from the Server. If it is unable to prepare itself to begin publishing, it returns to Registered.
- **Active –** This is the working state of the Publisher, in which it is publishing data. It periodically sends KeepAlive packets to

the Server. It returns to Registered when it receives the StopPublishing message from the Server.

- **Finished –** The Publisher reaches this state after it has been removed from the registration list of the Server, using the RemovePub message.

Subscriber

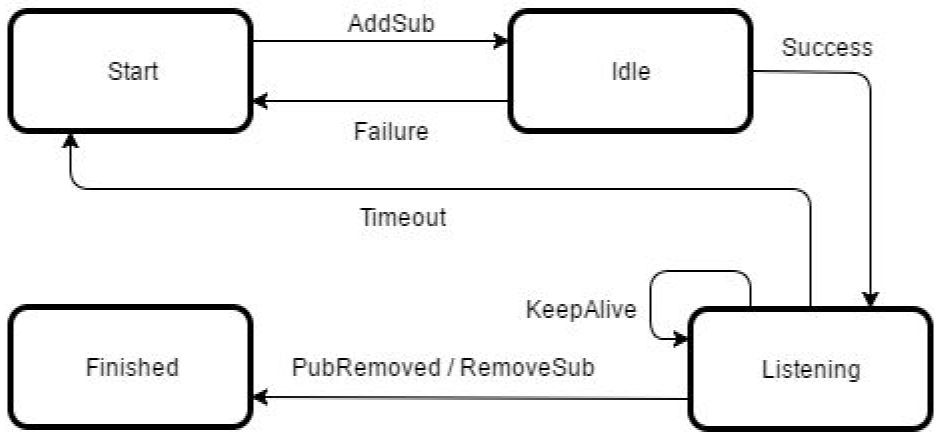

Figure 7 - Subscriber DFA.

- **Start –** It is the initial state. The Subscriber returns to it in the event of a timeout, or if the AddSub message results in an error.
- **Idle –** The Subscriber goes into this state while waiting for the Server’s reply to its AddSub message.
- **Listening –** This is the working state of the Subscriber, in which it is receiving data. It periodically sends KeepAlive packets to the Server. The Subscriber reaches this state after a successful AddSub.
- **Finished –** The Subscriber reaches this state after it has been removed from the registration list of the Server using the RemoveSub message, or if it receives the PubRemoved message from the server.

Extensibility

The Sensor Mesh Protocol (SMP) has some built-in features that provide extensibility for others to adapt the protocol to their needs. For example, custom sensor types can be defined with their own protocols to be used with the generic PDU’s provided by SMP. So, for example we may have a camera sensor that needs to send over images that don’t fit in one payload. The solution is to send the image over in pieces where the payload is split into fields that designate a piece index and the piece data. Then the subscriber can reassemble the image at the application layer.

The simplistic nature of this protocol allows it to be maintained for years to come without many if any changes required. It organizes publishers and subscribers so that they may communicate with each other. Any change will occur in the data payload itself, not the control packet. This allows for the focus and limitation of our protocol to be on the devices themselves on how many ports are available. As many publishers and subscribers can be linked as the number of available ports allows.

There are 8 bits for the Command Code field which leaves room for 256 commands minus the 10 we already specified. There is also a 1 byte reserved field for any additional fields that may need to be added in for any version following SMP 1.0 in the future.

Security

Due to the nature of our protocol only being implemented amongst devices on the same network, the limitations of space make it difficult for a remote client to attach to the network and cause havoc. Subscribers may only attach to devices sharing the same network, so the only way that a stranger or unexpected client can join the network is via a Virtual Private Network in which case the client must have a way to login meaning that they must have access to a profile with the VPN provider. This pushes all responsibility of unwanted guests onto the maintenance and care of the VPN provider as well as the individual in charge of adding and removing accounts to the VPN.

The second form of security emphasized in our protocol is data integrity. Each type of our packets have a CRC that must stay unchanged through transmission in order for the receiving client to validate it. This is necessary because of the underlying transport layer being UDP. CRC will protect data from common noise errors, but will not protect the data from deliberate modification to the packets. The Central Node is responsible for checking the CRC of the messages, and it uses the CRCCalculator.py file we wrote based on the IEEE 803.2 error checking algorithm.

Within each control packet resides a transaction ID which allows the central node to maintain what conversation it’s having with the publisher and subscribers. If a response is not received in time, the connection will timeout and the transaction will have to retry if still interested.

When a subscriber tries to subscribe to a publisher that they are already linked to or tries to remove a publisher they already are disconnected from, then a failure message is returned from the central node which specifies an error code regarding the reason why the request failed. This is important because the node should know why a failure/error resulted, and we don’t want duplicate connections to be made between a publisher and subscriber. This security feature also prevent denial of service attacks from occurring.