CSE160 - Project #1

TA: Ashish Yadav
Email: ayadav6@ucmerced.edu

Flooding

- One of the simplest methods of multi-hop communication
- Also one of the most inefficient one-to-one protocol
    - The bubble sort of Networks
- Has no knowledge of the topology

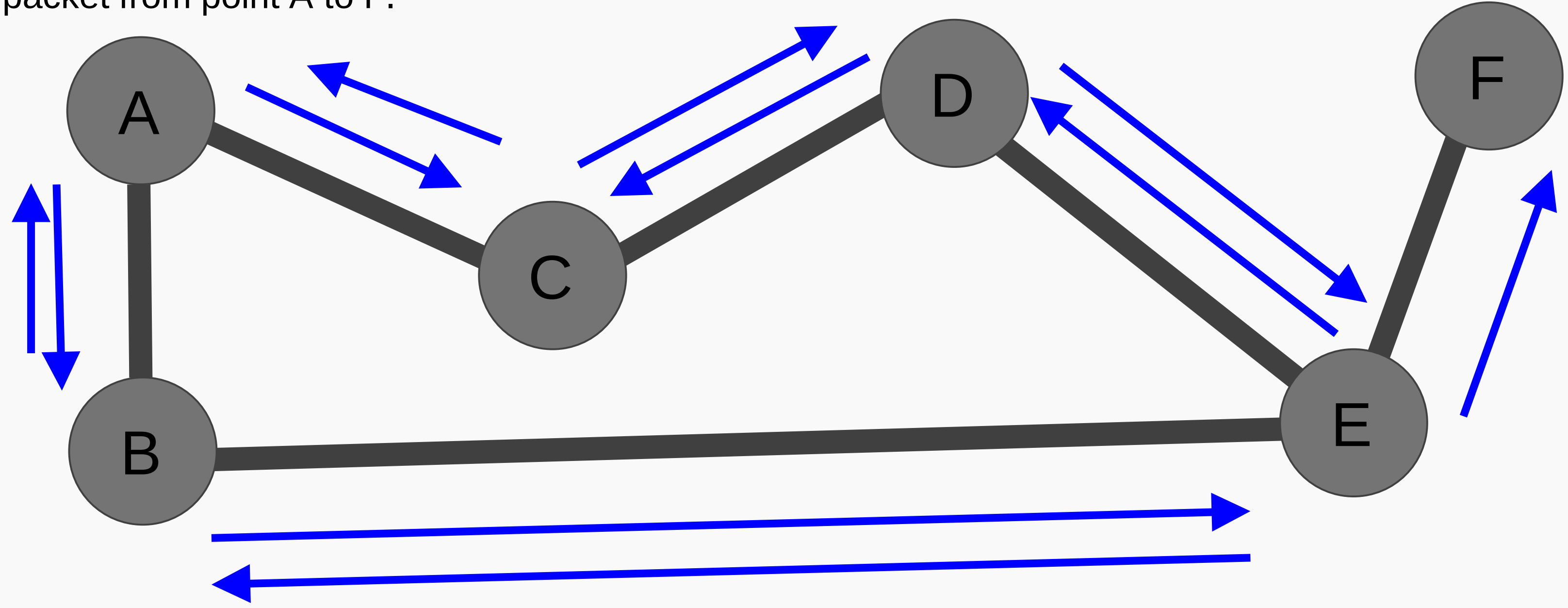
Sending a Letter using Flooding

- I want to send to a letter to my pen-pal in New York, but I don't have his address.
- The solution is easy, I'll use the protocol I learned in Networks
- I make multiple copies of the letters and give it to my friends
- Whenever they receive a letter, if it is not for them they make a copy and give it to all of their friends
- Continue to do this until the letter is received at the final destination

https://en.wikipedia.org/wiki/Six_degrees_of_separation

Flooding

Send packet from point A to F.

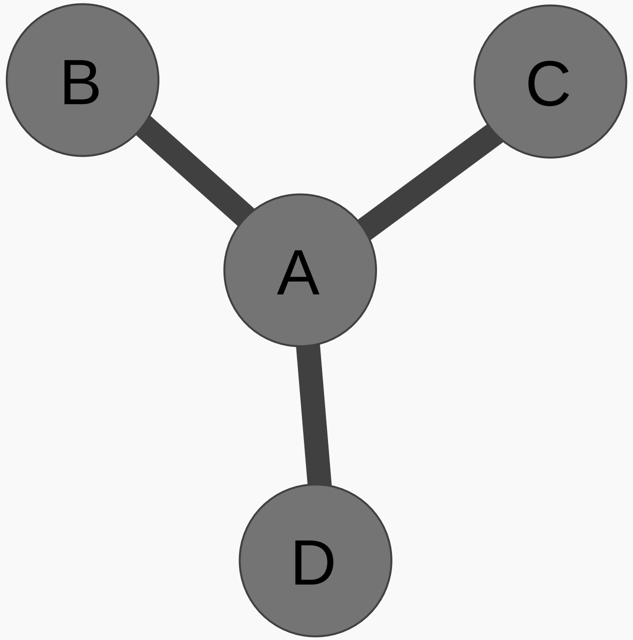


Flooding Notes

- A few key problems:
    - Question: How do we track packets we have seen?
    - A Solution: Using a sequence number and source information
    - Question: How do we know when to give up?
    - A Solution: Having a maximum time to live (TTL) : **Time to live** (**TTL**) or hop limit is a mechanism that limits the lifespan or lifetime of data in a computer or network.

Neighbor Discovery

- Neighbor discovery is built on flooding
- You are looking for your immediate neighbors



TinyOS - Mini-Guide

Quirks of TinyOS

- No dynamic allocation. E.g. malloc()
- Breaking up large functions into smaller loops
- Processor is not powerful
    - No hardware floating point operations

Interfaces

- Similar to headers in C and C++

SimpleSend.nc

```
#include "../../includes/packet.h"
interface SimpleSend{
  command error_t send(pack msg, uint16_t dest );
}
```

SimpleSendP.nc - Module Header

```
module SimpleSendP{
  provides interface SimpleSend;
  uses interface Queue<sendInfo*>;
  uses interface Pool<sendInfo>;
  uses interface Timer<TMilli> as sendTimer;
  uses interface Packet;
  uses interface AMPacket;
  uses interface AMSend;
  uses interface Random;
}
```

SimpleSendC - Configuration

```
#include "../../includes/packet.h"
configuration SimpleSendC{
  provides interface SimpleSend;
}
components as App;
  SimpleSend = App.SimpleSend;
  components new TimerMilliC() as sendTimer;
  components RandomC as Random;
  components new AMSenderC(AM_PACK);

  App.sendTimer -> sendTimer;

  App.Random -> Random;
  App.Packet -> AMSenderC;
  App.AMPacket -> AMSenderC;
  App.AMSend -> AMSenderC;


  //Lists
  components new PoolC(sendInfo, 20);
  components new QueueC(sendInfo*, 20);


  App.Pool -> PoolC;
  App.Queue -> QueueC;
}
```

Sending

```
error_t send(uint16_t src, uint16_t dest, pack *message){
    if(!busy){
        pack* msg = (pack *)(call Packet.getPayload(&pkt, sizeof(pack) ));
        // This copies the data we have in our message to this new packet type.
        *msg = *message;
        if(call AMSend.send(dest, &pkt, sizeof(pack)) ==SUCCESS){
            busy = TRUE;
            return SUCCESS;
        }else{
            dbg(GENERAL_CHANNEL,"The radio is busy, or something\n");
            return FAIL;
        }
    }else{
        dbg(GENERAL_CHANNEL, "The radio is busy");
        return EBUSY;
    }
}
```

Sending

```
  event void AMSend.sendDone(message_t* msg, error_t error){
     //Clear Flag, we can send again.
     if(&pkt == msg){
        busy = FALSE;
        postSendTask();
     }
  }
}
```

Conclusion

- There is a lot of information provided here, but there is a ton more documentation out there.
    - Read parts of the Tinyos Programing Guide by Phil Levis and David Gay
- Next step I am going into is the structure of the code

Any Questions?