Reliable Stream Analysis on the Internet of Things

ECE6102 Course Project

Team IoT

[redacted]

[redacted]

Submitted

April 30, 2014

1. Introduction

Team IoT is interested in developing a distributed system that supports live stream analysis on the Internet of Things network. In particular, the system will allow resource-poor devices (e.g. smartphones, tablets, laptops) with well-developed sensors and cameras to perform live stream video analysis with the help of nearby peer devices that provide necessary computing resources. With the assistance of highly available cloud nodes, the IoT devices will be able to make a request to off-load processing onto other IoT nodes in vicinity. While developing the system, the team will address the fault-tolerance and latency issues that could occur because of the dynamic nature of the IoT devices. The team would like to also address advanced communication protocols involving live stream processing via WiFi Direct-enabled android devices.

2. Design Objectives

The IOT live stream analysis will achieve the following goals:

- Dynamic provisioning of available nearby computing resources for offloading
- Offload computing mechanism for stream analysis
- Fault- tolerance mechanism to deal with failing nodes

3. System Architecture

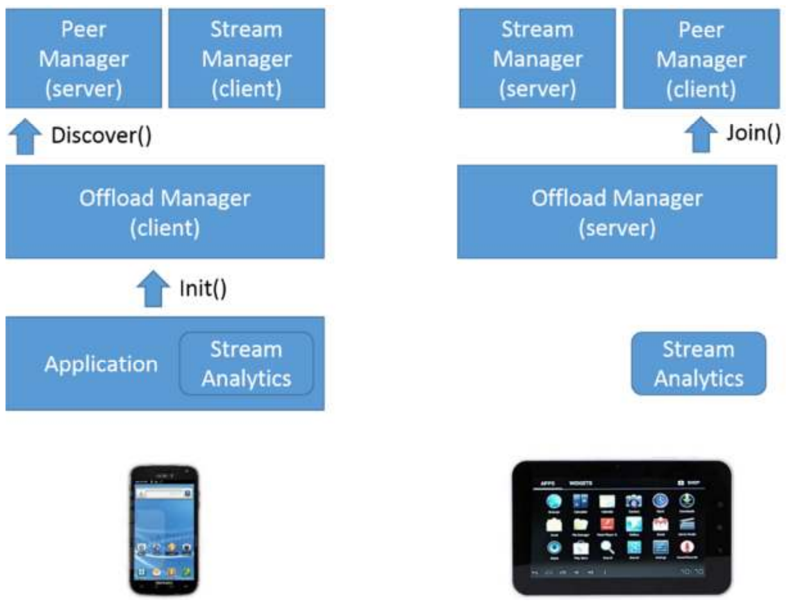

Figure 1. System Overview

3.1 Offloading

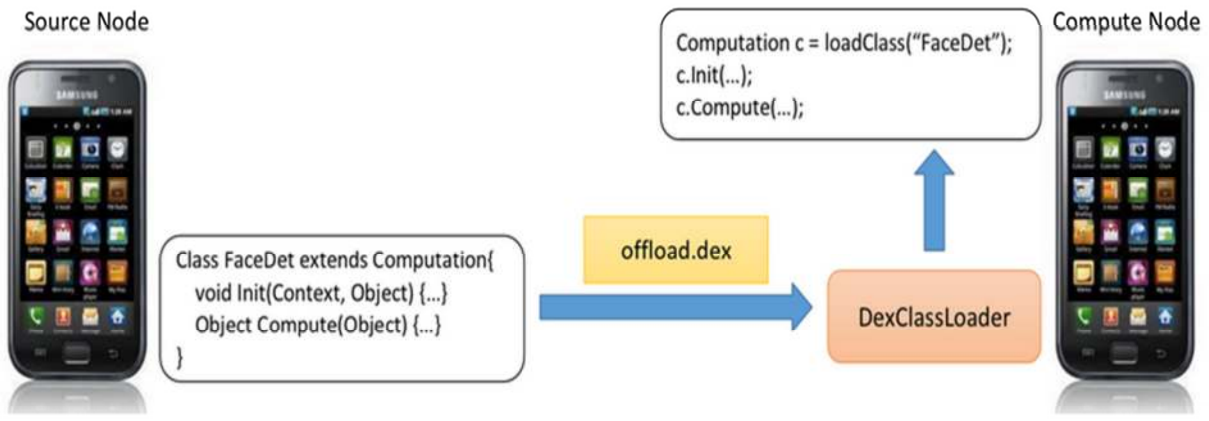
Source Node
Compute Node
Computation c = loadClass("FaceDet");
c.Init(...);
c.Compute(...);
Class FaceDet extends Computation{
void Init(Context, Object) {...}
Object Compute(Object) {...}
}
offload.dex
DexClassLoader

Figure 2. Offloading Manager

For offload computing, we developed simple interfaces that will allow a compute node to run an arbitrary computation without knowledge of the detailed structure of the computation. Our interface for offload computing, called 'Computation' includes two java methods, namely 'Init()' and 'Compute()'. The 'Init()' method is invoked when a computation is newly installed onto a compute node, helping the application-specific computation to initialize itself by inspecting the device context (e.g., network address, file system, etc.). Once the computation is installed and 'Init()' method is invoked, our system invokes 'Compute()' method for each input data from a source node. This 'Compute()' method essentially implements an application-specific computation that the source node wants to offload. Each invocation of 'Compute()' method returns an application specific result as a general 'Object' class. Once a compute node transmits results back to the source node, the source node dynamically typecasts the Object class into an application-specific type of result.

We implemented the offload computing mechanism using java class loader. Android has slightly different mechanism for dynamic class loading from regular Java since it uses a customized class file called 'dex', but both dynamic class loading mechanisms are quite similar. Such dynamic offloading scheme can be also implemented in other languages / systems (e.g., using dlopen() in Linux environment).

3.2 User Interface (UI) and Thread Management

The User Interface will have a start button, which upon pressed will initiate the live stream analysis. The node which initiates the live stream (source node) will redirect the stream into another thread, known as the sender thread. The sender thread is in charge of transforming the data into a supported format (if necessary) and passes the data to the stream manager. The stream manager then streams data to a remote node, which processes the data (compute node). The compute node has a pool of workers to perform the processing. The default value size of the worker pool is four workers but can be reconfigured by the developer to improve throughput. Each worker on the compute node is a thread that processes one data segment at a time. Each thread grabs the next segment of data from the stream to process. After the worker finishes processing the data segment, the output is pushed to the send queue on the compute node. The send queue passes the data to the stream manager, which delivers the output stream to the source node. The source node receives data from the output stream via a receiving thread. The receiving thread compiles the results and/or updates the UI based on the application requirements.

The Application used to analyze the IOT live stream analysis infrastructure was real time face detection. In addition to the start button, the source node UI has two frame. The first frame displays the input frames captured by the camera, while the second frame displays processed frames from the compute node. The compute node UI has four frames to represent four workers. Each frame displays the most recent processed image by the respective worker. Figure 3. displays the control flow for face detection at the application layer.

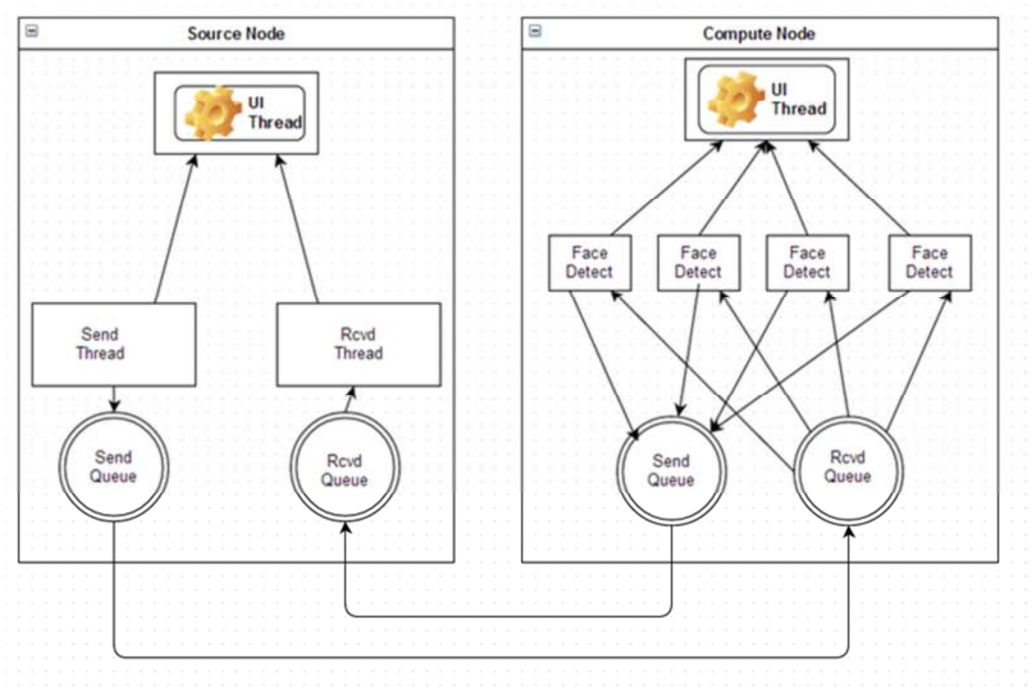

Figure 3. Application Layer Overview

3.3 WiFi Peer to Peer Streaming

Technology trends in communication has been towards going wireless and higher bandwidth. In the IoT environment, direct wireless communication is the means to exchange data between a variety of smart devices like laptops, smart phones, tablets. Amongst the wireless technologies like Wi-Fi, Bluetooth, RFID, the most popular one is Wi-Fi. Almost every electronic device has Wi-Fi capabilities. However, this direct device to device Wi-Fi requires an intermediate network access point that's established by physical routers. We term this as "Infrastructure mode Wi-Fi". This traditional Wi-Fi is implemented over traditional Wi-Fi radios.

WiFi direct, also called as WiFi P2P, is a new technology defined by the Wi-Fi Alliance aimed at enhancing Infrastructure WiFi. Wi-Fi capabilities can now be entirely implemented in software. This technology enables device to device connectivity without requiring the presence of an Access Point(AP). WiFi Direct is directly built upon Infrastructure Mode and lets devices negotiate as to who will take over the Access Point like functionalities. This device is referred to as the "Group Owner". This gives WiFi Direct added advantages of inheritance from Infrastructure mode for Power Savings, Security, QoS.

WiFi Direct, as compared to Bluetooth, supports higher bandwidth and allows speedy connections over longer distances. Wifi P2P allows Android 4.0 or later devices with the appropriate hardware to connect directly to each other via WiFi without an intermediate access point. Android's WiFi P2P framework complies with the WifiDirect technology. Another important point to note is Android WiFi P2P works without Infrastructure WiFi mode turned on. This shows the independence between WiFi P2P and Infrastructure WiFi. Using Android API's, you can discover and connect to other devices when each device supports Wi-Fi P2P as shown in Figure (4) below:

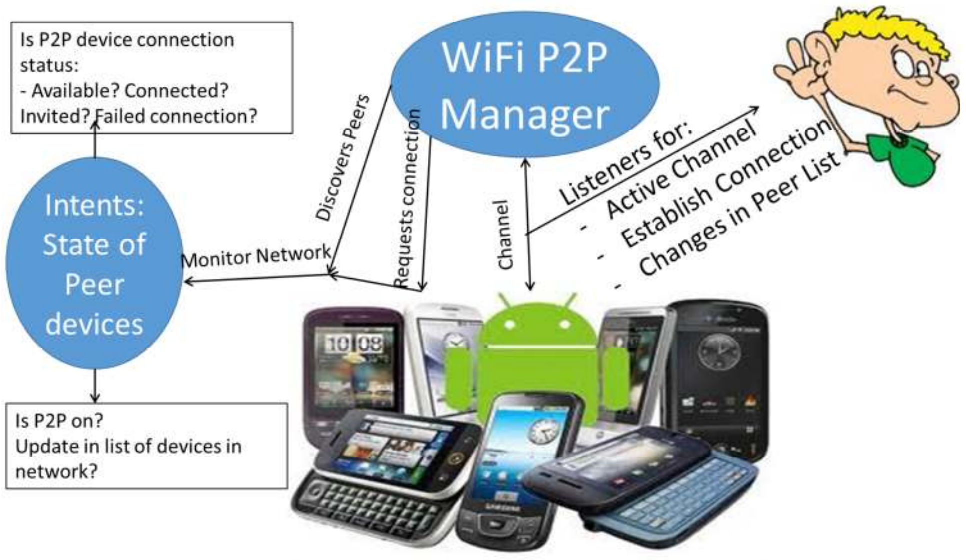

Figure 4. WiFi-P2P Manager Overview

3.4 Stream Manager

The Stream Manager is one of the key components for our live analysis system. It handles the streaming protocols between the IoT devices on the network. It also implements a fault-tolerance mechanism for our system. The Stream Manager constantly monitors the status of the connection between the source node and the compute node. If the source node dies, the Stream Manager on the compute node will tell the Offload Manager to cease processing. If the compute node dies, the Stream Manager on the source node will notify the source node's Offload Manager, and wait for the Offload Manager to provide it with the IP address of another compute node that it can connect too. The Stream Manager also handles the re-ordering of packets between the source and compute node. While the compute node is able to process data out of order, the Stream Manager compensates for this by behaving like a FIFO buffer. It accepts out-of-order packets from the compute node, and reorders them so that the source node always receives packet in the same order that they were sent.

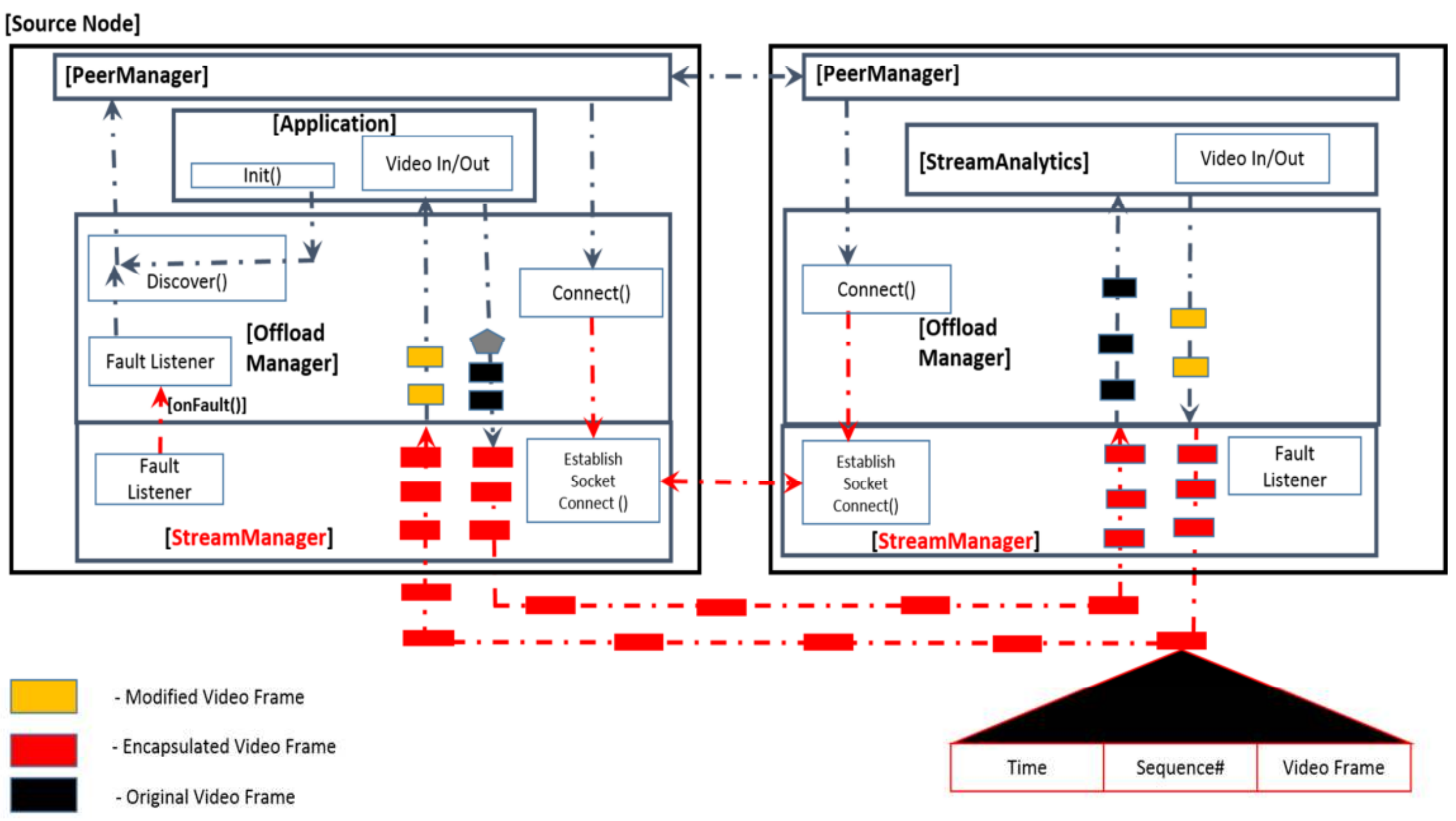

Figure 5. Stream Analysis System Design

Implementation

The Stream Manager was designed to only interact with the Offload Manager. The following describes the Stream Manager's implementation:

1. The Source-SM is initialized when the Source-OFFM calls Connect (), once the Source-OFFM receives the IP address of an available compute node from the Source-Peer Manager (PM).
2. Once Source-OFFM calls the function Connect () the Source-SM proceeds to establish a socket with the Compute-SM.
3. After the connection between the Source-SM and the Compute-SM is established, the Source OFFM begins offloading packets to the Source-SM.
4. The SourceSM buffers each packet it receives (for reliability) and sends each packet to the Compute node.
5. The Compute-SM buffers each packet until the Compute-OFFM submits a request for the next packet.
6. After each packet undergoes stream analysis, the Compute-OFFM submits a request to have the packet transmitted back to the source node.
7. Once the Source-SM receives a packet, it does one of two things
    a. If the packet sequence number (n+1) and the most recent packet in the Source-SM's buffer has sequence number (n), then the packet will be added to the buffer

b. If the packet sequence number (n+1) and the most recent packet in the Source-SM's buffer does not have sequence number (n), then the Source-SM will put the packet in an alternate buffer until it places packet with sequence number (n) into the buffer.
8. The Source-SM submits a packet back to the Source-OFFM in the same order that the Source-OFFM submitted packets.

If failure occurs while two nodes are connected, the Source-SM will notify the Source-OFFM that of this failure, and wait for the Source-OFFM to provide it with the IP address of a new compute node for it to connect to in order to complete processing. Even though the Source-SM has reported a failed compute node, the Source-SM still allows the Source-OFFM to offload packets to it. Upon successfully connecting to a new compute node, the Source-SM redelivers all packets that it did not receive back from the last compute node that it was connected to. This capability can be seen in our UI. When a failure occurs the user can see a lag in the video display, but the video frames will not be seen as out of over. Once the Source-SM reconnects to a compute node, the user will observe that the video will resume normal display speed.

The Stream Manager node implemented meets all the requirements for the streaming protocol discussed in our proposal. It is able to connect two nodes reliably, handle communication faults, and ensures that packets are delivered in FIFO order to the source node.

4. Evaluation and Performance

The most important bottleneck for the IOT stream analysis infrastructure is communication latency. Since, the computation will encounter an overhead of round trip data transfer for each data segment from source to compute node. The team decided to use WiFi Direct as the communication channel over infrastructure WiFi. WiFi direct, as mentioned earlier, does not require an existing wireless network for connecting to other peers. In order to measure the effectiveness of WiFi Direct, the team evaluated the IOT stream processing using both WiFi Direct and Infrastructure WiFi.

4.1 End -to -End latency for offload computing

The setup to test the end-to-end latency for offload computing involved one source and compute node. The source node was connected to a computer to measure the end to end latency via Android USB real time debugging. For each trial, the source node would send 200 frames to the compute node. For each frame, the source node would measure the round trip time (RTT). Then the team measured the average of the 200 RTTs. For each trial the distance between the source and compute node was varied between 0 and 80 ft. The experiment was repeated twice: the first time with WiFi Direct and the second time with Infrastructure WiFi. The results of the experiment are displayed in the graph below (Fig. 6):

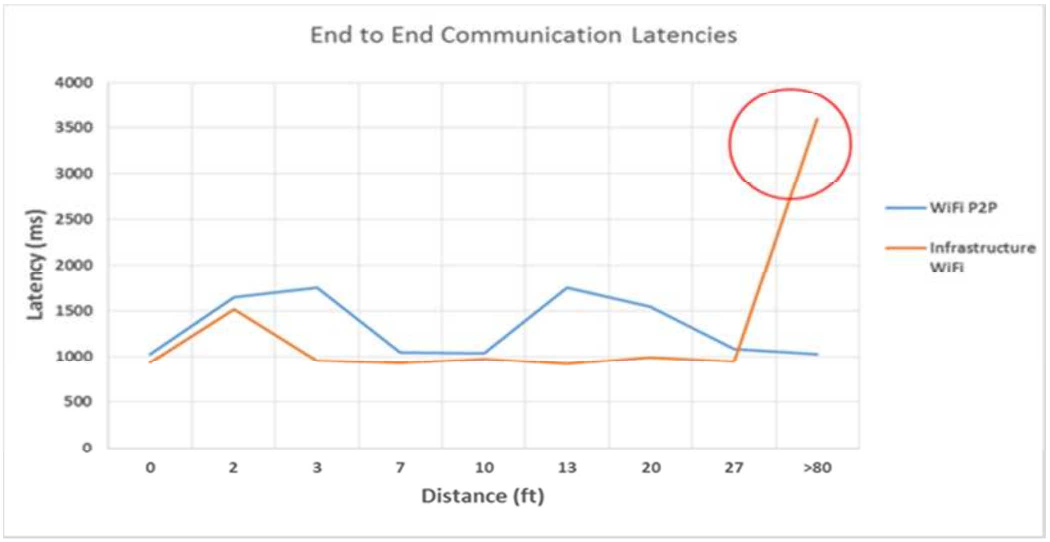

Figure 6. End to End Communication Latencies

According to Figure 6, WiFi Direct’s average latency is slightly higher than infrastructure WiFi. The difference between the two modes was a 100 ms on average. The measurements have run-to-run variation. This is noticed in Figure 7, when the experiment was carried out at the same distance several times. Figure 7 shows a variation of end to end latencies for three trials. The latencies varied from 1050 ms to 1020 ms for a distance of 20 feet. When the distance was 80 ft, infrastructure WiFi had a latency of 3.6 seconds while WiFi P2P had a latency of 1024 ms. The higher latency on infrastructure WiFi was due to change of access points during the experiment. Since, for majority of the distances, WiFi Direct’s latency is only slightly higher than Infrastructure WiFi’s latency, the team drew a conclusion from the experiment that the application can provide stable minimum delay for the remote stream processing for up to 80 ft.

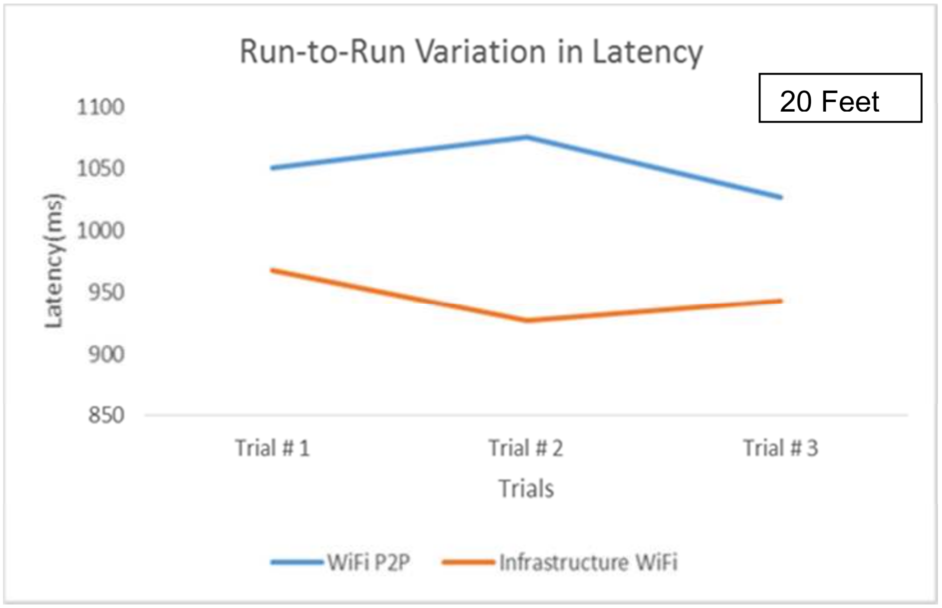

Figure 7. Run to Run Variations

4.3 Latency for handling device failure scenario

To better understand the effects that Wifi P2P and infrastructure Wifi have on the communication latency between nodes, we also tested the latency experienced by devices during a failure recovery scenario. Regardless of the type of WiFi channel used, the source node's Peer Manager must still perform some task which acquires the IP address of a live computing node in the network. With each method there are inherent overhead costs involved with recovering from a failure. Testing the latency from the moment the source node recognizes that a failure has occurred to the moment a new connection is established will bring to light the extent of the effects these overhead costs have on latency.

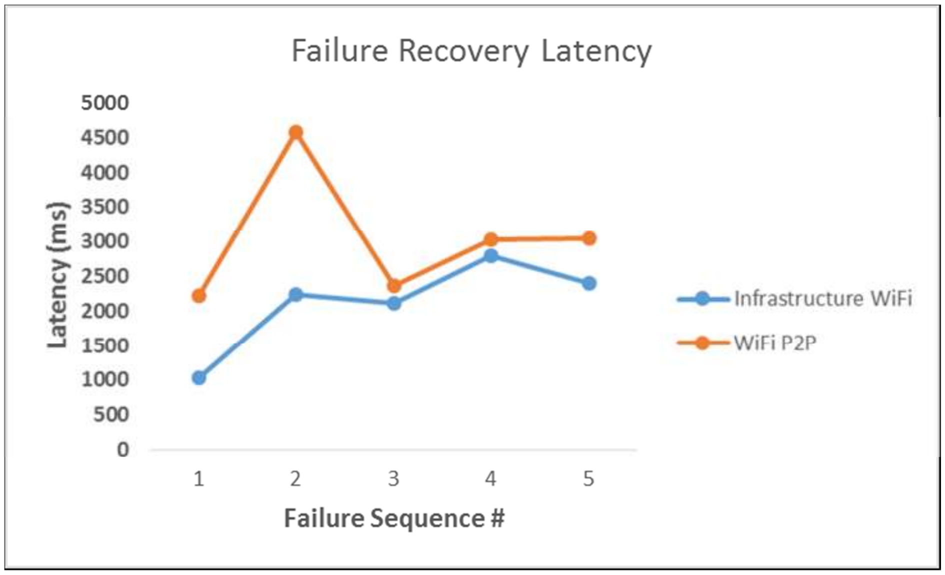

Figure 8. Failure Recovery Latency

Figure 8 reveals that infrastructure WiFi outperforms WiFi P2P during a failure recovery scenario. On average, their performance differs by 1 second. Infrastructure WiFi's ability to outperform Wifi P2P can be due to the fact that it does not have to search for long in order to find a new compute node to connect to. In Infrastructure Wifi, all compute nodes in the network are aware of the source node, and are continuously pinging the source node attempting to establish a connection with it when the source nodes presents itself as available. In this scenario, the source node never has to wait long to connect to a live node, because live computing nodes are always trying to connect to it. This is not the case for Wifi P2P. For an implementation using Wifi P2P, the source node has to have its Peer Manager search for available nodes that previously made themselves available in the network. The difference in this scenario is that the compute nodes in the network are passive, so that the source node PeerManager has to perform the steps necessary to verify that the connection with a compute node would be viable. While the PeerManager implementing infrastructure Wi-Fi merely has to wait on a compute node to request a connection, a Peer Manager implementing Wi-Fi P2P must search the peer group list, and attempt to establish connections with nodes in the list. When it is able to establish a successful connection with a node in the list, then the Peer Manager provides the node's IP address to the Offload manager so that the Stream Manager can proceed to establish a connection with this node. According to the results displayed, the overhead involved with WiFi P2P is enough to cause a slightly higher delay than infrastructure WiFi.

Overall, infrastructure WiFi outperforms WiFi P2P at short distances, but based on the results of our end-to-end latency test, we believe that WiFi P2P will continue to outperform infrastructure WiFi at distances greater than 80 ft. but less than 656 ft., where 656 ft. is the maximum distance that WiFi P2P can transmit reliably, according to Wifi Alliance [1].

5. Conclusion

Performing live stream analysis on resource poor IoT devices is a challenging endeavor that can be overcome with proper implementations. Reliability is a key requirement a stream analysis between devices. This project aimed to investigate various implementations of stream analysis protocols in an IoT environment. In the end, we were able to build a fault-tolerant dependable distributed system which was built around the idea of allowing source nodes to record videos, offload their facial recognition processing to other nodes, and display the processed results in real time within the source node's UI. We determined also that a live stream analysis system should be designed based on the environment that it would be used in. If the system is meant to be used within a building or local area, then WiFi P2P is the communication protocol that should be used. If the system is meant to be used in a global manner, then infrastructure WiFi should be the communication protocol of choice.

6. Future Work

6.1 Cost Estimation

Currently there is no policy to choose a compute node that gives the highest performance. One possible metric to use is the compute node's signal strength. Higher signal strength indicates reliable data transfer over the connection without any packet losses. However, the source node cannot determine the resources on the compute node by just observing signal strength. Thus, another method in picking a compute node is to measure end to end latency by broadcasting a computation to all nodes in the network and then streaming data to all the nodes. The source node can then measure the latency of the nodes to respond back with the processed data. Lower latency from a compute node indicates that offloading computation to this node will yield a higher performance. However, the downside of this cost estimation implementation is it takes too long since it needs to determine the cost of all nodes in the network and can also consume network bandwidth since all nodes in the network will perform their own cost estimation.

6.2 Relaxed Security

The current infrastructure has no security features to protect nodes from malicious users who join the peer to peer network. However, the current trend is all devices in the peer to peer network are owned by a single owner. Therefore, all devices in the system can be trusted. In order to expand the scale of the infrastructure from a single user to multiple users, several logical domains can be implemented across the peer to peer network. Each logical domain will have an admin, who will determine which devices are allowed to join the domain. The responsibility of determining which devices are malicious is determined by the users who use the IOT stream analysis infrastructure. Also, WiFi P2P can inherit all security features of Traditional WiFi.

7. Work Breakdown

[[redacted] Streaming Protocol and Fault Tolerance among IoT Devices
[[redacted] High-Bandwidth Peer-to-Peer Communication between IoT devices for improved Performance
[[redacted] User Interface and Thread Management
[[redacted] Dynamic Code Installation and Execution Mechanism for Offload Computing
[[redacted]] Performance Evaluation
[[redacted]] Component Integration

8. References

[1]Daniel Camps-Mur, Andres Garcia-Saavedra and Pablo Serrano; Device to device communications with WiFi Direct: overview and experimentation