Contentify Evaluation

Experimental Setup

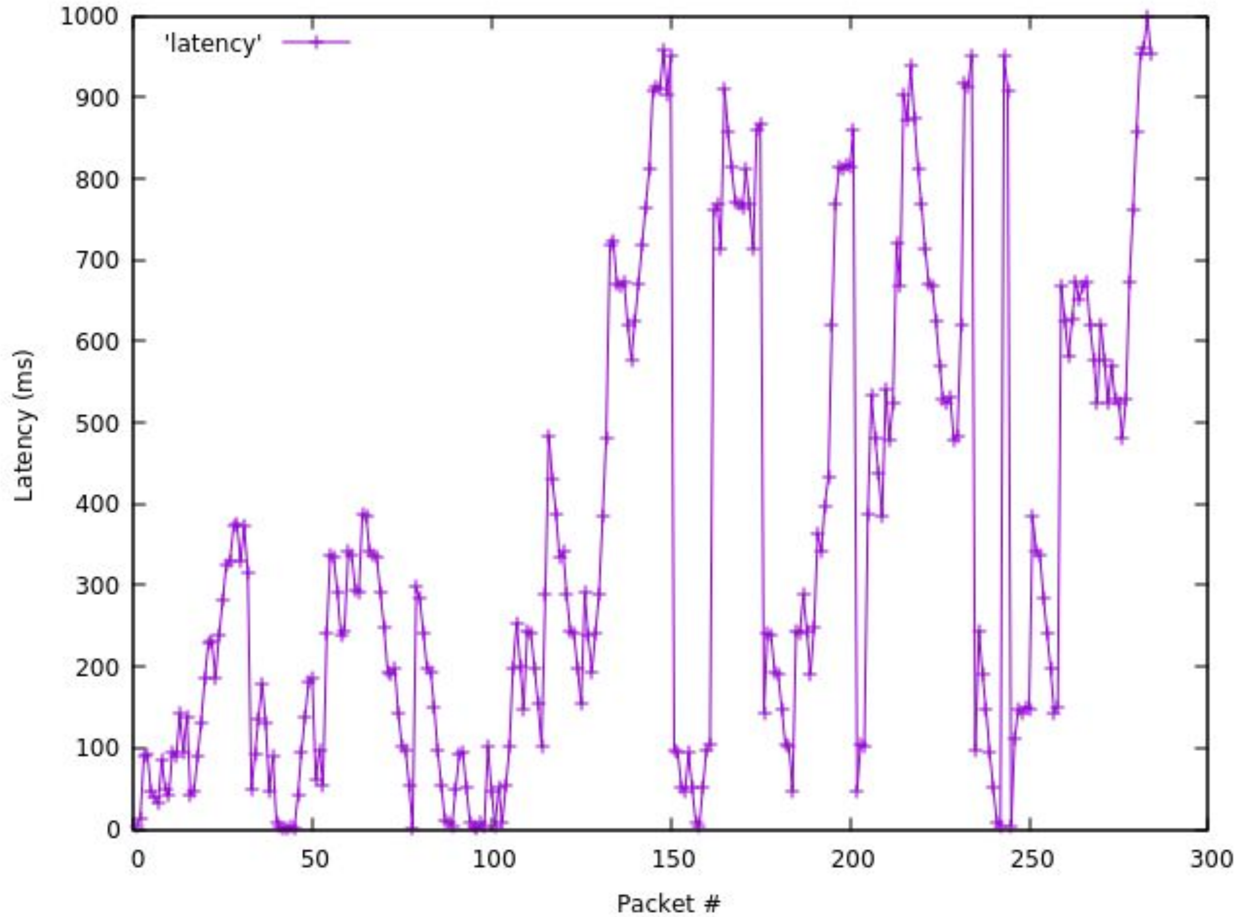
Major part of the set up (as well as the project) was devoted to getting a modular, generic implementation for x86_64 as well as ARM processors and using Raw Wireless Frames.
In order to comply with the current drivers and infrastructure, nodes in the network were set up in monitor mode which limited the size of the network.

A 3-node network was created with raspberry pi's in monitor mode, the following type of nodes were used:

- **Client / Host**
- **Sensor**
  The client and the sensor broadcast their id and type to the switch
- **Switch**
  The switch listens for known type of frames and registers advertisement frames. Forwards the request based on the type and destination.
  The client and sensor communicate via the switch and the response times of such a set up were evaluated. The switch starts up with an hierarchical address of /CMU/WEH/4F/S.
  The client sends a request: type = temp da = /CMU/WEH/4F/S.

Fig 1. Experimental Setup

Evaluation



Average latency: 354.4882258900649 packet loss = 5.33333%

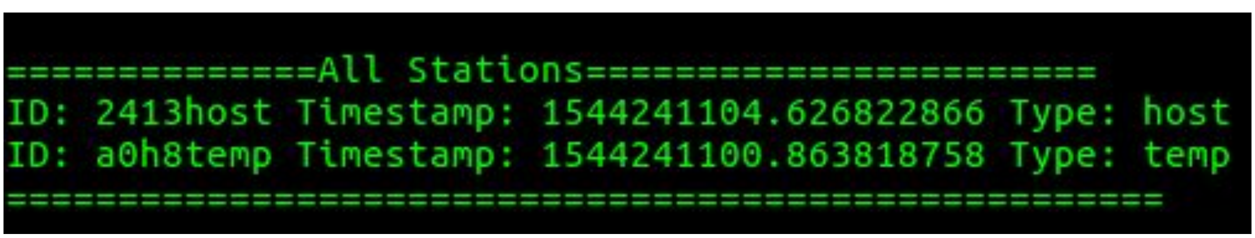


Fig 2. Switch output

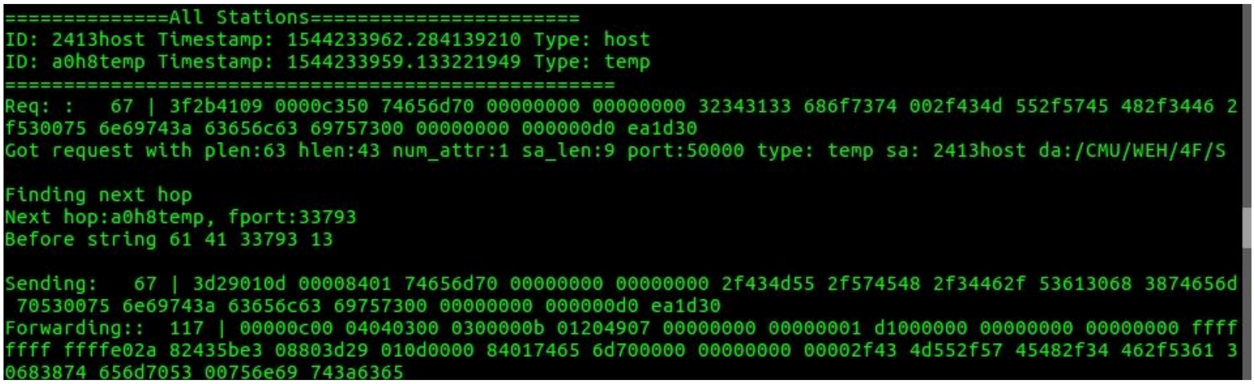


Fig 3. Forwarding to next hop

Software

Operating System: Debian Stretch

Patched click with ccn element executable on Raspberry Pi after cross-compilation: https://github.com/MirrorZ/Contentify/tree/master/rpi

(Also contains the compilation instructions) Configuration files: https://github.com/MirrorZ/Contentify/tree/master/r

Hardware

- Raspberry Pi 3 Model B+
- Raspberry Pi 2
- Linux Laptop
- Wifi USB Adapters with Monitor Mode Support
  - Panda Wireless Adaptor
- TP-Link TL-WN721N

Challenges

- Working with existing radio interfaces and drivers

This is a general challenge for any new addressing scheme that moves away from the way frames are read and written by most devices, operating systems. During the project, I faced difficulties in avoiding custom packets rejected by the radio interface driver. While most of the ethernet traffic can handle custom frame types, radios are much more troublesome and require considerable effort to get right.

RadioEncap() and EtherEncap() from click are generally helpful but radio drivers strip and append their own headers and this gets harder to customize. Therefore the experiment was carried out in monitor mode to ignore the packets being dropped by the driver. This limited the network size due to scarcity of external wireless adapters.

- x86_64 vs ARM
  Most of the testing in the project was first conducted using virtual interfaces on x86 machines and then using several wifi adapters on the same machine. On extending the codebase to run on arm devices, the communication between x86 and arm devices suffered from alignment issues leading to garbled data.

- Monitor mode not ideal, Multicast in Wireless is hard
  While a lot of packets can be missed if not processed due to buffer queues, monitor mode is just too noisy for a stable setup. Multicast in Wireless faces issues with retransmissions and is not straightforward.

Learning

- Getting click working on a raspberry pi
  There is no documentation on this whatsoever, but now the instructions are present in the
  public repository of this project, which would make network prototyping using Click easier for sensor networks!

- Framing Wireless Packets
  Apart from first hand experience of framing wireless packets, this let me explore more aspects of setting up wireless networks using heterogeneous devices.

- Networking with sensor like devices
  Sensor devices have various constraints in the form of power and compute (and software support!), the project helped me build some perspective in the area of sensor networking

- Designing wireless Protocols
  Learned to design basic communication protocols and addressing formats.