UNLEASH THE POWER OF LIMITLESS CONNECTIVITY
Wireline Access Network

Tracking Round Trip Time Latency in the MSO Network

Michael Overcash
Principal Engineer
Cox Communications
WHY LAG MATTERS

People really hate lag and packet loss

NRG HusKerrs @HusKerrs - Jul 13
Nice, my upload speed with *is now randomly at 8 Mpbs this morning, so I can’t even stream. Swear there’s some sort of issue every week with cable internet.

Ryan Tow @TowGott - 21:06
Hey fix your internet. This is absolute garbage 😞

Skylerguns @skylerguns
This internet issue is going to kill me. support is useless and just try and upsell you. I don’t know what else to do...constant packet loss when streaming causing lag in game, stream delay, stream quality decrease, etc....

Only started 3 days ago - no problems before.

FaZe Dirty @FaZeDirty - Feb 4, 2017
is at all time suck. These lag spikes are mad.

100T steel @JoshNissan - Jul 20
another day of packet loss despite admitting that the problem is on their end. hopefully they can get this resolved so that i can do my job effectively (business line btw)
GOALS

- Use realistic UDP streams to measure latency, rather than ICMP pings
- Ability to distribute test points widely throughout the Access Network
- No special configuration of subscriber CPE equipment (e.g. no need for port forwards)
- Upgradable with ability to add new features and test protocols over time
- Low hardware cost
- Configurable network utilization
- Portable software
LAGSPY PROOF OF CONCEPT

- Raspberry Pi 4B mailed to employee volunteers
  - Lag-Pi
- Plugs into existing home router, no special configuration
- Managed and controlled by central Poller
- Lag-Pi’s run IRTT tests to server(s) to measure latency and jitter

Start collecting real data from the real access network

Ultimate goal is to develop a framework that can be implemented on managed gateways
LAGSPY SYSTEM ARCHITECTURE

- RPi
- Home Router
- CM
- DOCSIS
- CCAP
- SLR
- Container
- irtt server
- Hub Router
- Metro
- Cox Backbone
- ONT
- PON
- OLT
- AAR

MQTT Broker: Polling logic and database

Additional irtt servers can be placed anywhere in the Metro network, backbone or even the internet.
IRTT OVERVIEW

• Bidirectional **UDP** test stream that simulates an audio or video stream
• Open source, widely available in Linux distros
• Parameters are configurable
  • Packet size
  • Interpacket interval
  • UDP port and optional HMAC
• Measures
  • Round Trip Time (RTT)
  • Jitter
  • Data sent/received
• Limitations
  • Bidirectional only
  • Tries to decompose RTT into send and receive components ... badly
SO WHY UDP INSTEAD OF ICMP?

- Different QoS
- Control Plane (ICMP) vs Data Plane (UDP)
- Real applications use TCP or UDP to transmit data. Nothing uses ICMP.
- Many devices rate limit ICMP handling for DDoS protection.
IRTT EXAMPLE

```
irtt client -i 20ms -l 172 -d 30s --fill=rand --sfill=rand --hmac=0x<redacted> -q irtt-telemetry.coxlab.net:22112
[Connecting] connecting to irtt-telemetry.coxlab.net:22112
[184.176.185.20:22112] [Connected] connection established
[184.176.185.20:22112] [WaitForPackets] waiting 352ms for final packets
```

```
<table>
<thead>
<tr>
<th></th>
<th>Min</th>
<th>Mean</th>
<th>Median</th>
<th>Max</th>
<th>Stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTT</td>
<td>78.92ms</td>
<td>84.55ms</td>
<td>83.55ms</td>
<td>117.3ms</td>
<td>3.17ms</td>
</tr>
<tr>
<td>send delay</td>
<td>-1.24s</td>
<td>-1.23s</td>
<td>-1.23s</td>
<td>-1.21s</td>
<td>2.25ms</td>
</tr>
<tr>
<td>receive delay</td>
<td>1.31s</td>
<td>1.32s</td>
<td>1.32s</td>
<td>1.35s</td>
<td>2.22ms</td>
</tr>
<tr>
<td>IPDV (jitter)</td>
<td>1.93µs</td>
<td>2.32ms</td>
<td>1.13ms</td>
<td>34.77ms</td>
<td>3.15ms</td>
</tr>
<tr>
<td>send IPDV</td>
<td>110ns</td>
<td>1.89ms</td>
<td>925µs</td>
<td>19.11ms</td>
<td>2.41ms</td>
</tr>
<tr>
<td>receive IPDV</td>
<td>754ns</td>
<td>740µs</td>
<td>274µs</td>
<td>34.42ms</td>
<td>2.31ms</td>
</tr>
<tr>
<td>send call time</td>
<td>12.9µs</td>
<td>72µs</td>
<td></td>
<td>932µs</td>
<td>46.5µs</td>
</tr>
<tr>
<td>timer error</td>
<td>100ns</td>
<td>129µs</td>
<td></td>
<td>827µs</td>
<td>107µs</td>
</tr>
<tr>
<td>server proc. time</td>
<td>4.45µs</td>
<td>9.39µs</td>
<td></td>
<td>128µs</td>
<td>4.86µs</td>
</tr>
</tbody>
</table>
```

duration: 30.3s (wait 352ms)
packets sent/received: 1471/1471 (0.00% loss)
server packets received: 1471/1471 (0.00%/0.00% loss up/down)
bytes sent/received: 253012/253012
send/receive rate: 67.5 Kbps / 67.5 Kbps
packet length: 172 bytes
timer stats: 28/1499 (1.87%) missed, 0.64% error
WHERE TO PUT THE IRTT SERVER?

• Wherever you want to!
  • Containerized with minimal resource requirements

• Cox is using the Service Layer Router (SLR) attached to the Hub Router
  • SLR is as close to access network as we can get
  • Router container implementation limits resource usage, preventing impact to other services
  • Access controls on router ensure only a single port/service is accessible from the public IPv6 address

• A VM will be used for IPv4-only households (~50%)!
  • Not as close as SLR, we are assessing the impact
IRTT TRAFFIC PROFILES

- Currently simulating audio stream (67.5 Kbps, 172 byte UDP payload)
- Next step is to simulate gaming and video conferencing traffic
- An application can easily be characterized using Wireshark
  - Perform inline sniffer capture using switch with mirror port
  - Analyze capture using Wireshark IO Graph and Packet Length analysis tools
- Full details in paper
LAGSPY SOFTWARE ARCHITECTURE

Container
LagSpy Test Client

Container
MQTT

Container
Wireguard

Lag-Pi

Cox Network

VPN

Container
Lag Spy Poller

Container
MQTT

Container
Wireguard

Poller VM

IRTT servers can be deployed in any VM or container platform
The principal components of the Lag-Pi are:

- The LagSpy Test Client, written in Python 3.8.
- Eclipse Mosquitto to implement an MQTT client.
- Wireguard to establish a VPN connection to the Poller for command and control.

The principal components of the Poller are:

- The Lagspy Poller, written in Python 3.8.
- Eclipse Mosquitto to implement an MQTT broker and localhost client.
- Wireguard for a VPN endpoint.
- InfluxDB to import and aggregate data from the Poller for visualization.
- Grafana for visualization of test results.
- Lighttpd (primarily used to upgrade the Lag-Pi.)
<table>
<thead>
<tr>
<th>Topic</th>
<th>Arguments</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connect/hello</td>
<td>n/a</td>
<td>Lag-Pi → Poller</td>
<td>Register with poller and keepalive</td>
</tr>
<tr>
<td>connect/enroll/&lt;mac&gt;</td>
<td>MAC address of Lag-Pi</td>
<td>Poller → Lag-Pi</td>
<td>Provide VPN credentials to Lag-Pi</td>
</tr>
<tr>
<td>connect/link_ok/&lt;mac&gt;</td>
<td>MAC address of Lag-Pi</td>
<td>Poller → Lag-Pi</td>
<td>Keepalive response</td>
</tr>
<tr>
<td>irtt/start/&lt;mac&gt;</td>
<td>MAC address of Lag-Pi</td>
<td>Poller → Lag-Pi</td>
<td>Start IRTT test</td>
</tr>
<tr>
<td>irtt/results</td>
<td>n/a</td>
<td>Lag-Pi → Poller</td>
<td>Results of IRTT test</td>
</tr>
<tr>
<td>iperf/start/&lt;mac&gt;</td>
<td>MAC address of Lag-Pi</td>
<td>Poller → Lag-Pi</td>
<td>Start IPERF3 test</td>
</tr>
<tr>
<td>iperf/results</td>
<td>n/a</td>
<td>Lag-Pi → Poller</td>
<td>Results of IPERF3 test</td>
</tr>
</tbody>
</table>
YAML POLICY FILE

- Need policy framework to minimize manual configuration of devices
  - We mailed identical devices to volunteers!
- Assign Lag-Pi’s into groups that can run different tests
  - We are leveraging this framework to automate IPERF3 testing
- Define IRTT Server groups (server selected by network hops)
  - No need to manually provision individual Lag-Pi to closest server

```yaml
groups:
  irtt-testing:
    group-name: irtt-testing
    permissions:
      - run-irtt-tests
      - write-irtt-results
    enabled: true
    devices: default
  iperf3-testing:
    group-name: iperf3-testing
    permissions:
      - run-iperf3-tests
      - write-iperf3-upstream
      - write-iperf3-downstream
    enabled: true
    devices:
      - e4:5f:01:3b:18:23
      - e4:5f:01:3b:17:43
  irtt-IPv6-server-IPs:
    group-name: irtt-IPv6-server-IPs
    IPs:
      - irtt-telemetry.coxlab.net
    enabled: true

  irtt-IPv4-server-IPs:
    group-name: irtt-IPv4-server-IPs
    IPs:
    enabled: true

  iperf3-server-IPs:
    group-name: iperf3-server-IPs
    IPs:
      - 192.168.0.43
    enabled: true
```
Mean Round Trip Time – Note cyclic increases for some devices
Mean Jitter. Again note cyclic behavior.
Round Trip Time Standard Deviation
UDP Packet Lost (out of 1500)
RESOURCE USAGE

- CPU usage < 0.1% idle, 4.3% during active test
- Light memory footprint
- Application is easily portable to a Linux-based gateway platform (especially if platform already supports Docker)

<table>
<thead>
<tr>
<th>Component</th>
<th>% Memory</th>
<th>Virtual Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docker overhead</td>
<td>6.2%</td>
<td>3.4 MB</td>
</tr>
<tr>
<td>Python Test Client</td>
<td>0.6%</td>
<td>52 KB</td>
</tr>
<tr>
<td>IRTT Client</td>
<td>0.1%</td>
<td>879 KB</td>
</tr>
</tbody>
</table>
LESSONS LEARNED

- 50% of households have home gateway with IPv6 disabled
- Many home network topologies – big benefits to router integration
- Wireguard NAT keepalive
  - PersistentKeepalive keyword
- Many internal reviews needed
- Ability to remotely upgrade Lag-Pi is critical from day 1
- Outbound firewall rules are still a thing
- NOOBS 3.5 compatibility issue with Raspberry Pi 4
Thank You!

Michael Overcash
Principal Engineer
Cox Communications
michael.overcash@cox.com