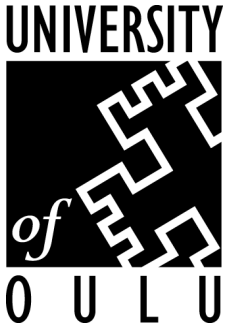
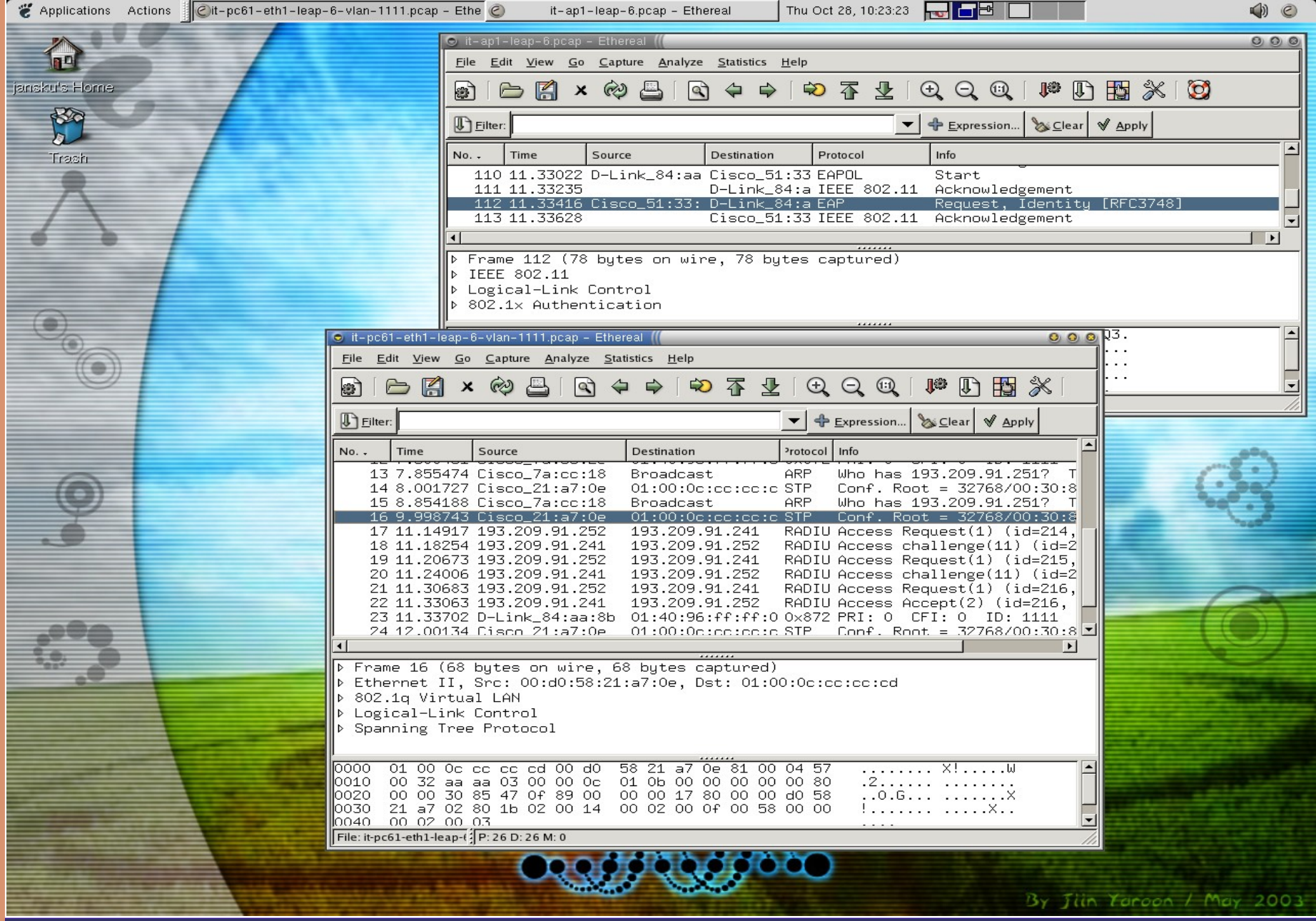


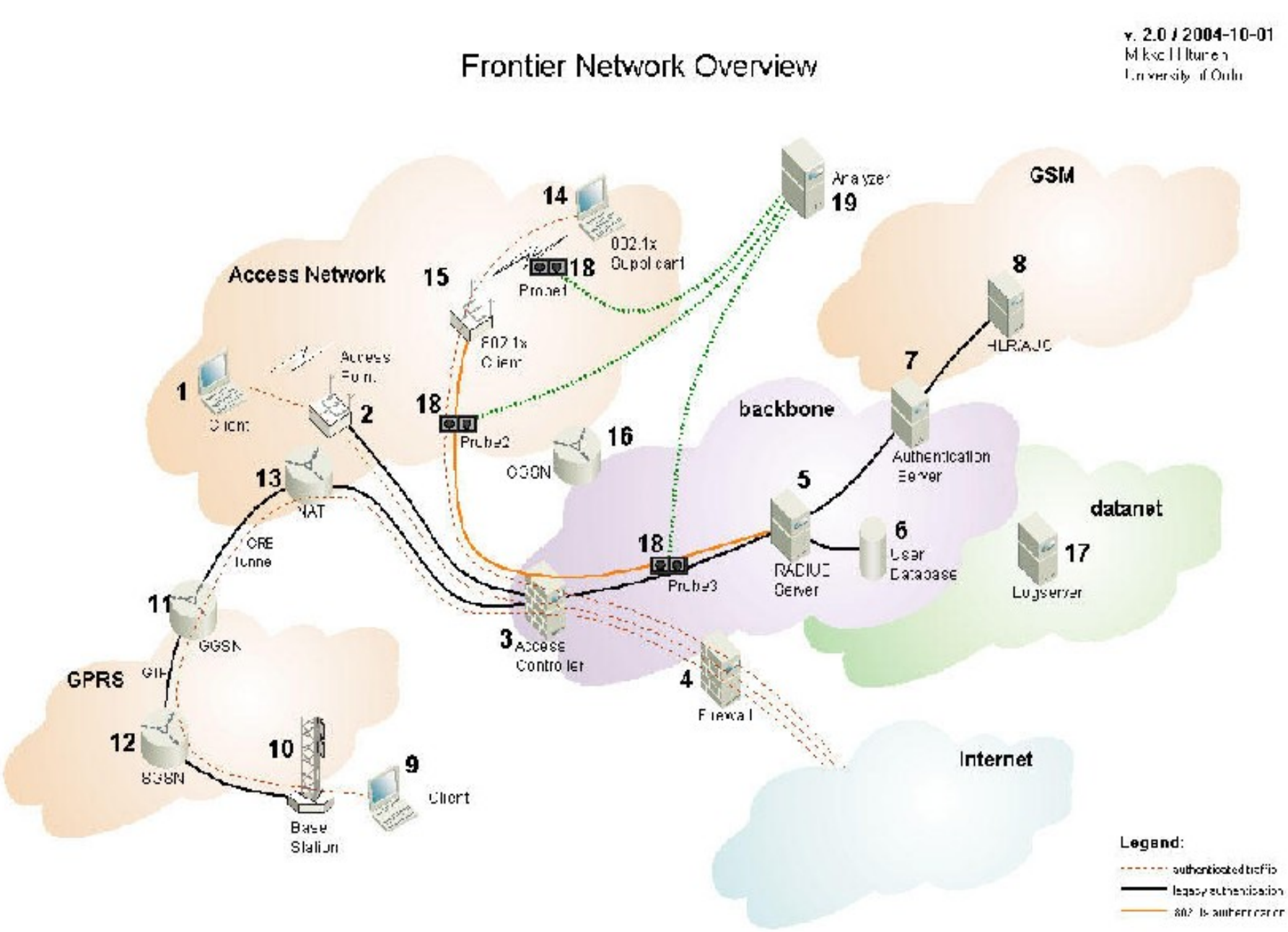

# Exploiting Causality and Communication Patterns in Network Data Analysis

Pekka Pietikäinen (pp@ee.oulu.fi)

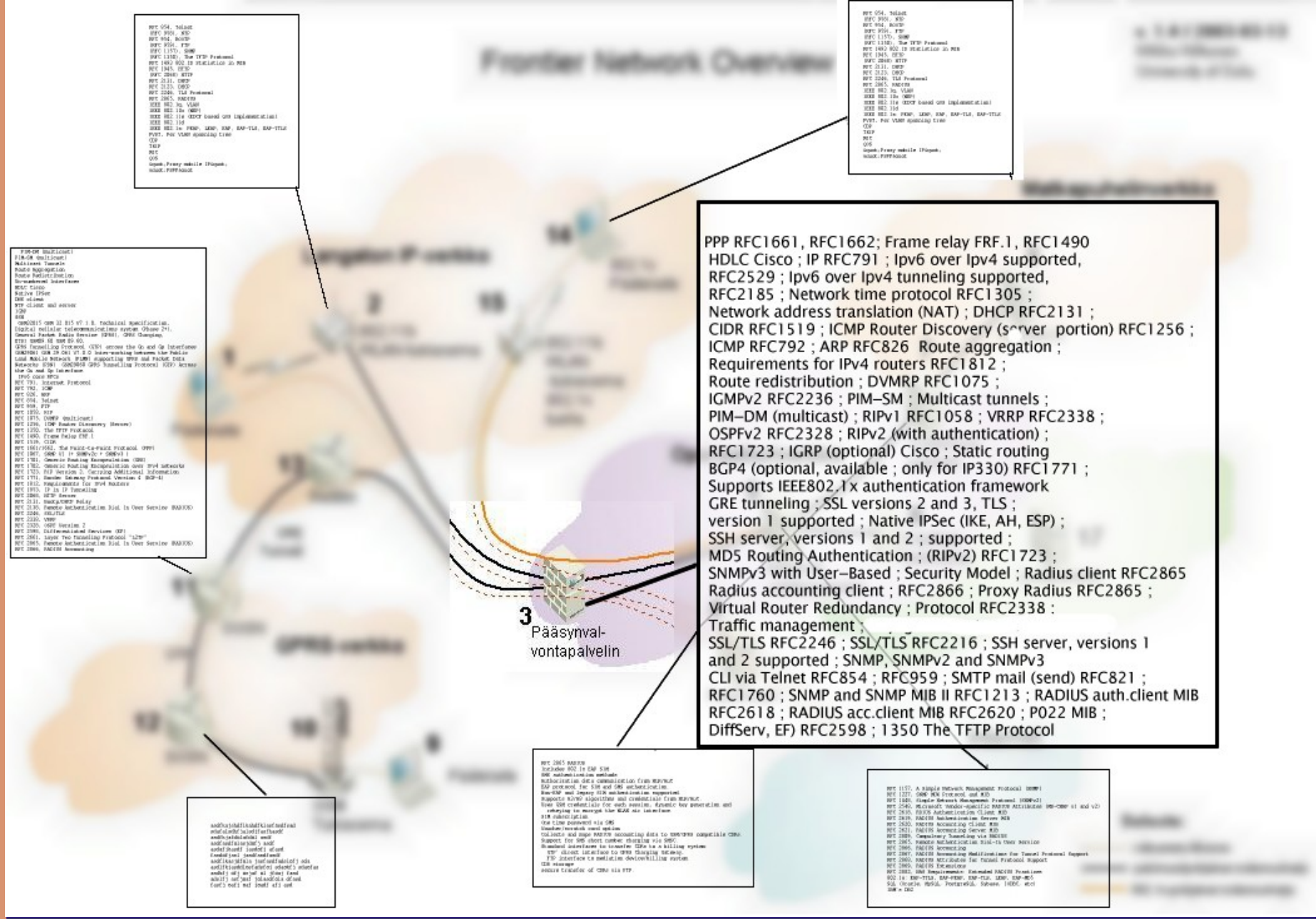

Oulu University
Secure Programming
Group



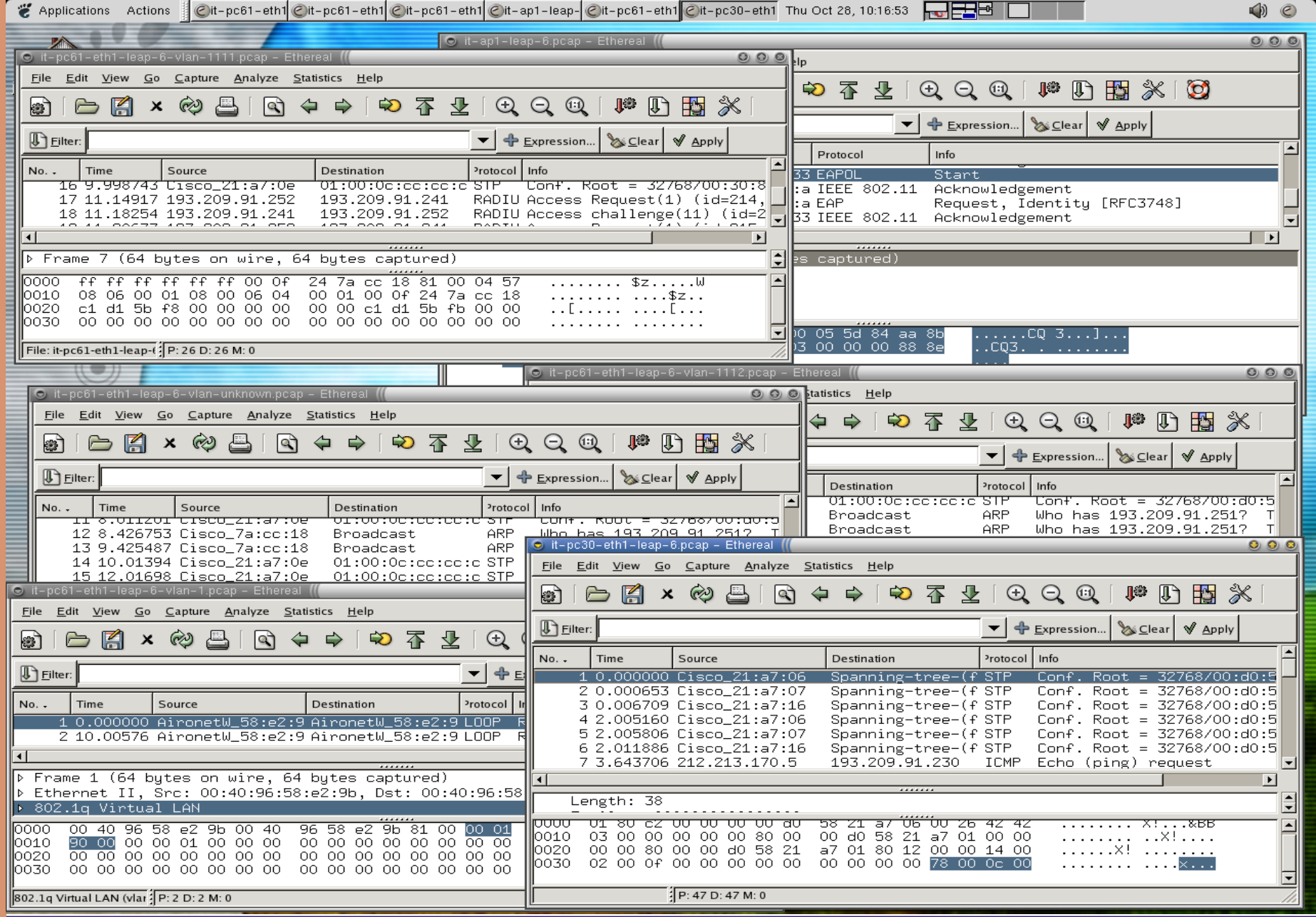

# Network analysis is fun...




Let's add some complexity.




# ... and related protocols


Applications Actions
it-pc61-eth1
it-pc61-eth1
it-pc61-eth1
it-ap1-leap-
it-pc61-eth1
it-pc30-eth1
Thu Oct 28, 10:16:53
it-pc61-eth1-leap-6-vlan-1111.pcap - Ethereal
it-ap1-leap-6.pcap - Ethereal
it-pc61-eth1-leap-6-vlan-unknown.pcap - Ethereal
it-pc61-eth1-leap-6-vlan-1112.pcap - Ethereal
it-pc61-eth1-leap-6-vlan-1.pcap - Ethereal
it-pc30-eth1-leap-6.pcap - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
No. Time Source Destination Protocol Info
Frame 7 (64 bytes on wire, 64 bytes captured)
Frame 1 (64 bytes on wire, 64 bytes captured)
Ethernet II, Src: 00:40:96:58:e2:9b, Dst: 00:40:96:58
802.1q Virtual LAN
Length: 38
P: 26 D: 26 M: 0
P: 2 D: 2 M: 0
P: 47 D: 47 M: 0

End result

- Analyzer shows what it can decipher from the data
- Filters are simple, efficient rules that reduce the amount of data that is shown
  - BPF etc.
- You have to know what you're looking for
  - Sometimes it's data that's **MISSING**
    - Firewall blocking packets, MTU issues etc.
- Multiple networks, multiple addresses for different components

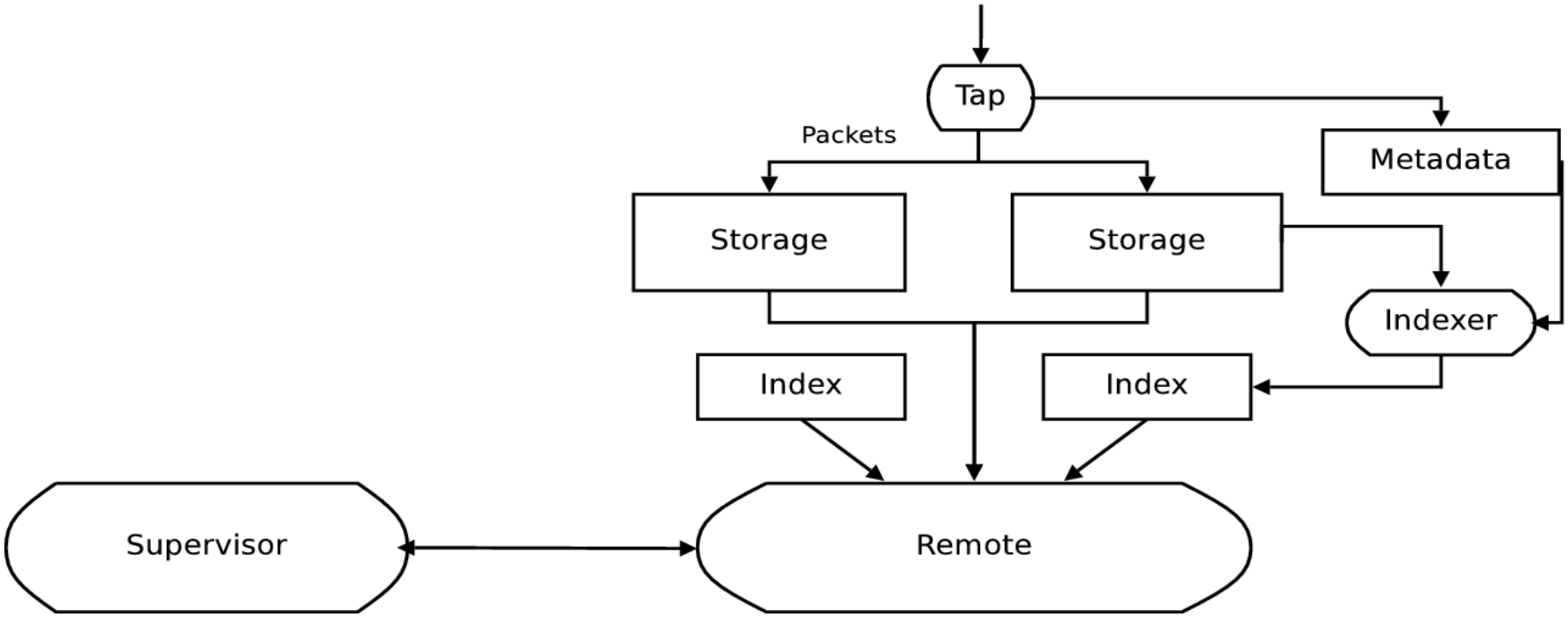
## In other words

- Treat network traffic as flows consisting of packets
- When a new flow is discovered on the network, find candidates that are potentially related
- Candidates are chronologically preceding flows, for which one of the endpoints of the new flow is involved.
- Have to have understanding on what addresses different components have

## Causality in network traffic

- In first iteration all packets were transmitted to central location for analysis
- Maintaining a graph for all traffic is resource intensive.
- Usually you can work on the flow level, sometimes you need the packets.
  - Architecture should allow quick retrieval of flows and corresponding packets
- Causality is an useful abstraction

## Drawbacks and lessons learned




# Architecture of probe

- “Regular expressions”  for network traffic
- Simple Python library for fetching flows/packets and matching attributes.
- Packets are fetched only if necessary

```
@matcher
def matchexample((flow, packet)):
   def flowMatcher():
      return flow.src=='10.0.0.1' and
            flow.probe=='probe-eth2'
   def packetMatcher():
      # Accept any packet belonging to
      # a matching flow
      return True
   return flowMatcher, packetMatcher
```

# Example #1

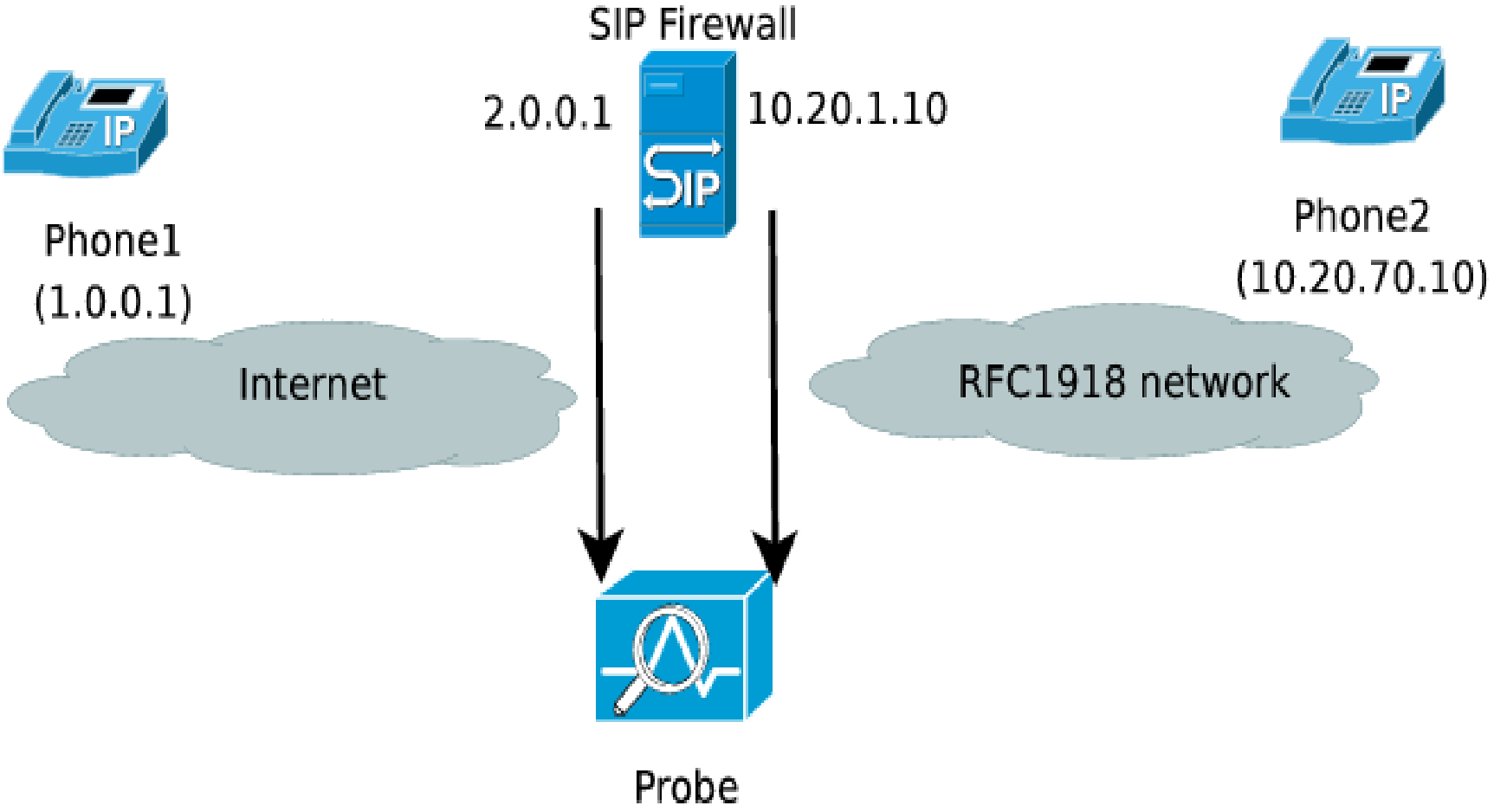
```
@matcher
def icmpMatcher(flow, prev):
   def flowMatcher():
       return flow.dst == prev.src and
              flow.proto == 'ICMP'

matcher2 = sequence(matchexample,
                    icmpMatcher)
```

# Chained matchers

- NoPair
  - Triggers when a given sequence does not occur within a given time window
  - eg. NoPair(matchexample, icmpResponse)

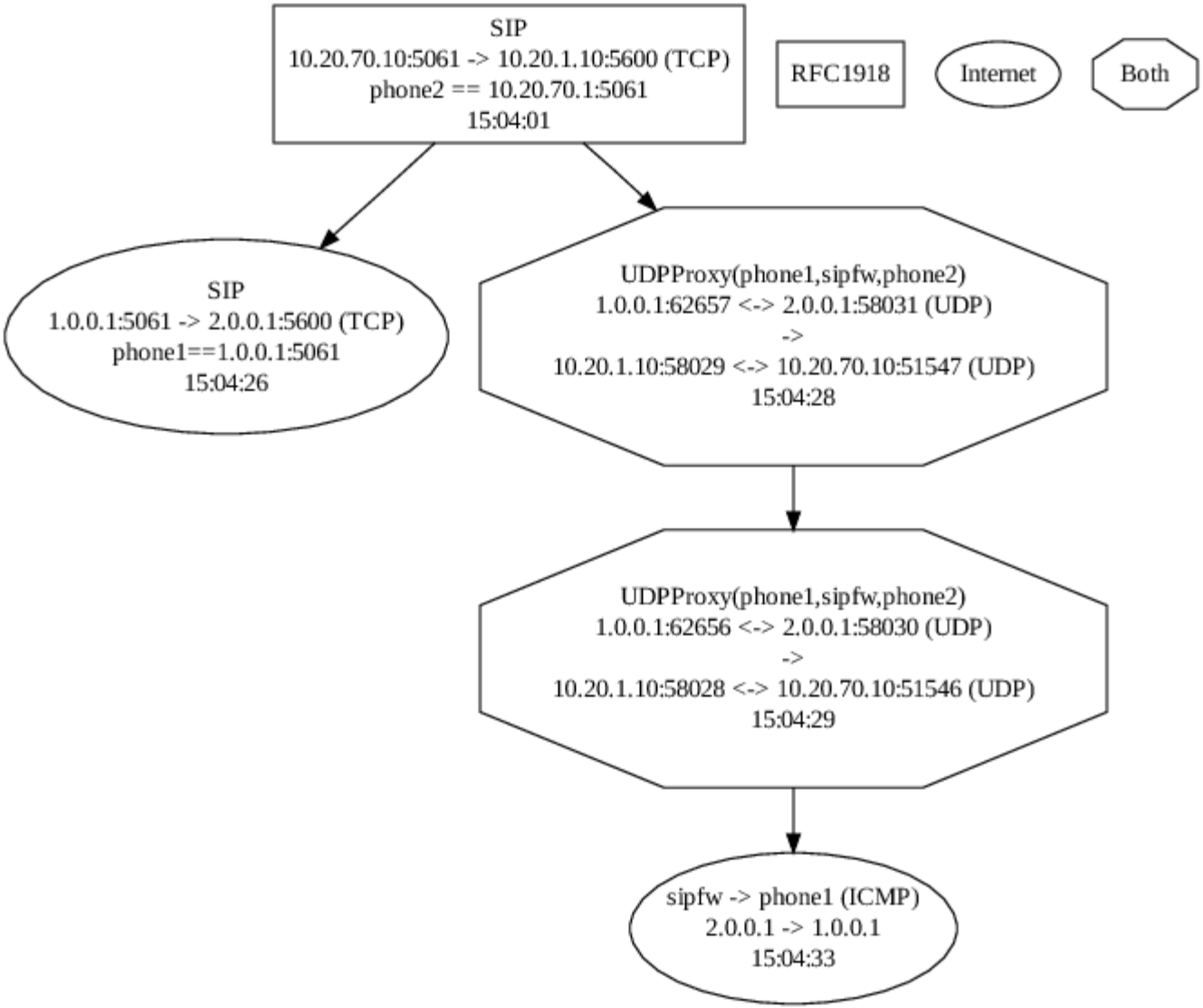
```
gi = GeoIP.new(GeoIP.GEOIP_MEMORY_CACHE)
def foreignDstMatcher(flow):
   def flowMatcher():
      return gi.country_code_by_addr \
                  (flow.dst) != 'US'
```




# Simple example

- Isolating traffic that involves a call between phone1 and phone2
  - Addresses are dynamically assigned
- SIP
  - matches TCP port 5060 traffic on the firewall, parses session setup with a simple string regexp to find identities of phones
- UDPProxy
  - Sequence matcher that matches flow pairs that are both UDP, have the firewall as an intermediary AND matches phones
- Causal

## Pieces needed


SIP
10.20.70.10:5061 -> 10.20.1.10:5600 (TCP)
phone2 == 10.20.70.1:5061
15:04:01
RFC1918
Internet
Both
SIP
1.0.0.1:5061 -> 2.0.0.1:5600 (TCP)
phone1==1.0.0.1:5061
15:04:26
UDPProxy(phone1,sipfw,phone2)
1.0.0.1:62657 <-> 2.0.0.1:58031 (UDP)
->
10.20.1.10:58029 <-> 10.20.70.10:51547 (UDP)
15:04:28
UDPProxy(phone1,sipfw,phone2)
1.0.0.1:62656 <-> 2.0.0.1:58030 (UDP)
->
10.20.1.10:58028 <-> 10.20.70.10:51546 (UDP)
15:04:29
sipfw -> phone1 (ICMP)
2.0.0.1 -> 1.0.0.1
15:04:33

# Result

- Capture: speed of disk (Multiple GigE)
- Indexing: 40kpps
- Fetching flows: 2500 flows/s
- Simple matches: ~= same
- Sequence matches: ~= 60s to process 10s of traffic at 250Mbps

- Add more understanding of network topology to analysis
- Optimize
- Move more processing to edges
  - Do packet level matching there?
- Don't limit to just network traffic, there are other sources that provide useful information (e.g. WLAN association logs can be used to pinpoint IP/MAC address)

# THE END

http://www.ee.oulu.fi/research/ouspg/