Small Form Computing

A bump in the wire

The questions

- What can we do with an inexpensive small computer?
- Can we make it a part of a seamless wireless mesh network by installing SMesh?
- Can we make it a robot controller by performing computations on it?
- Can it encrypt/decrypt? At what rate?

The answers

- Let’s try and run something on it
- Learn about its target platform
  - MIPS 32 architecture
  - Single Core
  - Little Endian
  - Memory – 61 MB
- What about its capability?
  - Need of standard benchmarks

Linux to the rescue - OpenWRT

- A Linux distribution for embedded devices
- Flash the router!
- Bleeding edge version - Chaos Calmer

Now the benchmarking ..

- File Transfer over TCP
  - **Wired:** 89 Mpbs **Wireless:** 44 Mpbs **With I/O**: 7Mbps
- OpenSSL benchmarks

Test (for 3 seconds) block size: 16	Cloud2 Bps	NEXX Bps
SHA1	31787.25K	2805.40K
AES 256 CBC	54005.82K	6222.59K
2048 bit private RSA for 10s	1384 signs/s 49705 verify/s	8.1 signs/s 257.9 verify/s

- Definitely not for Asymmetric Encryption!

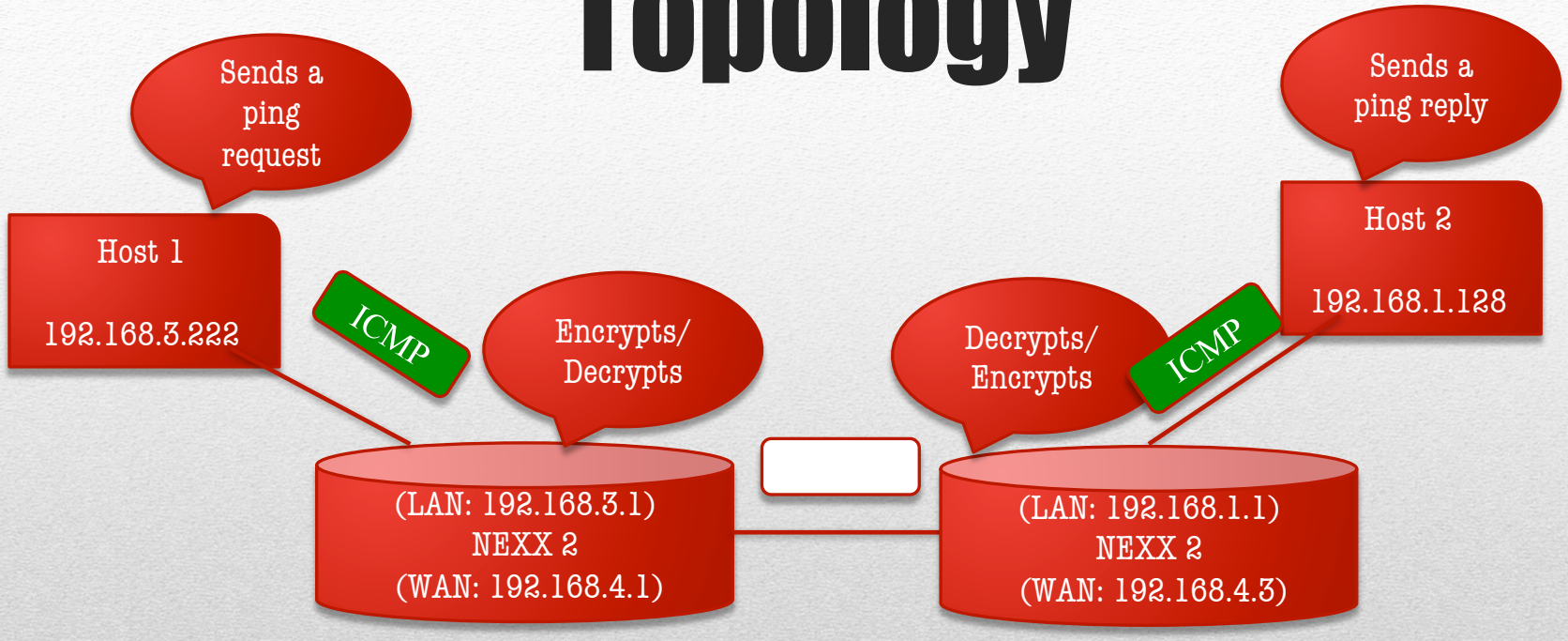
Let's write some code…

- Cross compilation – **Sourcery codebench Lite**, **OpenWRT SDK**
- *mipslinux-gnu-gcc -msoft-float -EL -static <helloworld.c> -o <helloworld>*
- Get USB support - **opkg** package manager
- Building a simple package using the SDK

What do we do with it?

- What about a Bump-In-The-Wire?
- Set it up as two intermediary hops between two hosts trying to communicate
- Encryption/Decryption on the fly

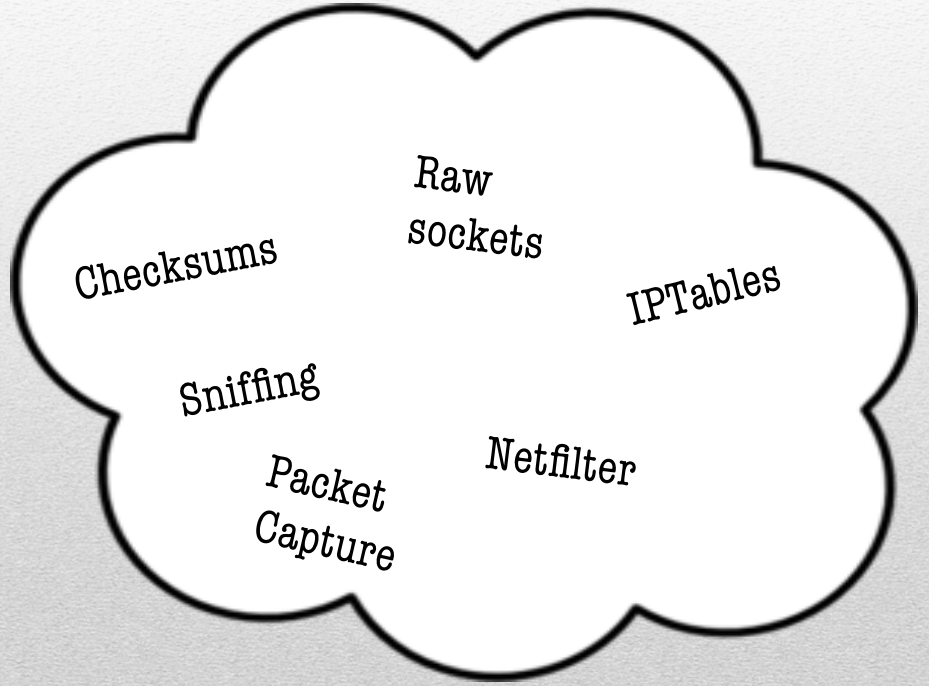
Topology


Sends a ping request
Host 1
192.168.3.222
ICMP
Encrypts/ Decrypts
(LAN: 192.168.3.1)
NEXX 2
(WAN: 192.168.4.1)
Decrypts/ Encrypts
ICMP
Sends a ping reply
Host 2
192.168.1.128
(LAN: 192.168.1.1)
NEXX 2
(WAN: 192.168.4.3)

Step 1

- Understanding Journey of an IP packet through the network

All that jargon



Raw Sockets

- A user level application can open a raw socket to get packets exactly as they would arrive on the network
- Not suitable for this application - creates a '**clone**' of each incoming packet for **every** application that has opened a raw socket, to listen to that type of packet. It didn't really 'bypass' the kernel processing

Divert Sockets

- They fit the bill, their very use case was to filter specific packets and get them to user space, giving the process total control of the packet. It could pass the packet as is, or choose to mangle it
- IPPROTO_DIVERT – instruct the firewall to send packets to a certain port, to which this socket is bound
- Different kernel needed

Netfilter/Iptables

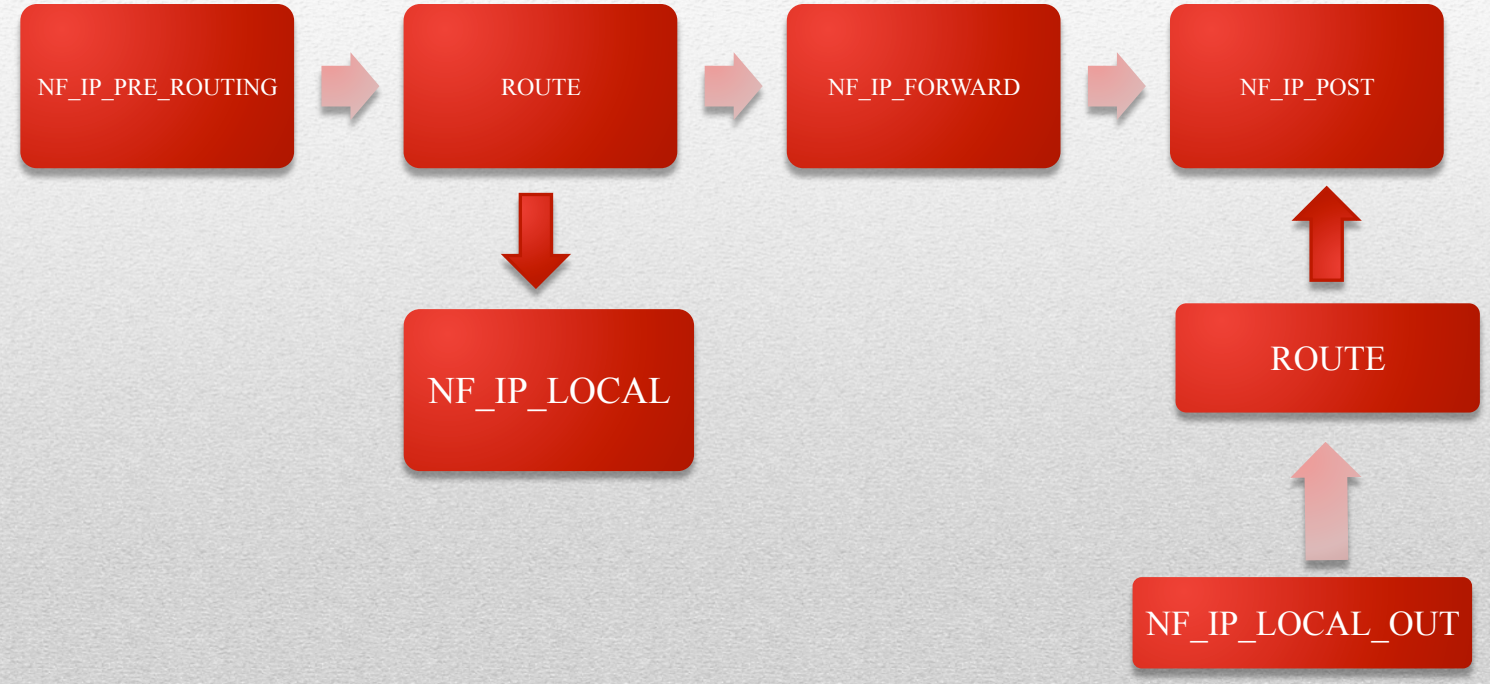
- A **framework** for packet filtering, a kernel subsystem in all modern linux kernels, all incoming packets traverse the netfilter subsystem

- Iptables - a **user level application** to interact with netfilter modules

Digging into Netfilter

- Each protocol (IPV4/IPV6/DECnet)defines '**hooks**'. These are well defined points in a packet's traversal of that protocol stack
- Kernel modules can register to listen on different hooks of that stack with priority. Packets are passed to them in order of priority on arrival
- These modules, can decide the '**fate**' of the packet
  - **NF_DROP/NF_ACCEPT/NF_STOLEN/NF_QUEUE**
- Packets that are queued, are **handled in user space**

Packet Traversing Netfilter system



Iptables

- A packet selection system that is built over Netfilter
- The ‘tables’ are modules that are registered at various ‘hooks’ in this framework, these ‘hooks’ are referred to as ‘chains’ when handling incoming packets
- `Iptables -t mangle -I FORWARD -p tcp -j ACCEPT`
- `Iptables -t mangle -I FORWARD -p tcp -j NF_QUEUE`

Libnetfilter queue

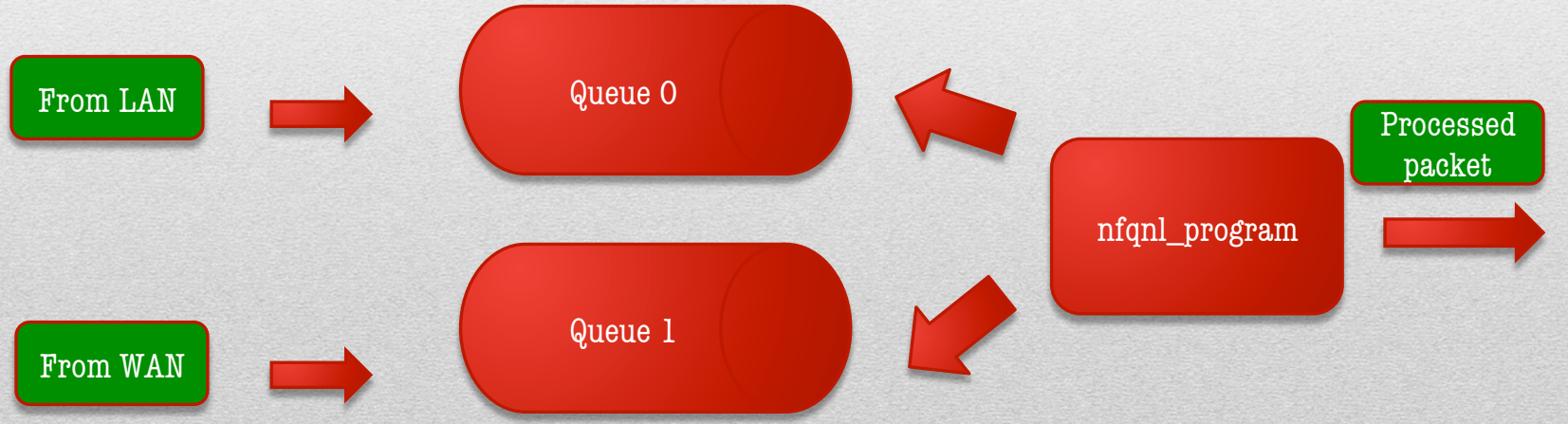
- Userspace library providing an API to packets that have been queued by the kernel packet filter

- **Three step process**:

1. Library setup – nfq_open(), nfq_unbind_pf(), nfq_bind_pf()
2. Message receiving – callback function for each received packet
3. Exit phase – nfq_close()

Our system

- Packet received – call back function – check type – encrypt/decrypt – re-calculate checksums – issue verdict



Encryption / Decryption

- Using simple XOR
- AES Encryption
- OpenSSL library

Performance (all wired)

Tool / Method	XOR	AES	No queue
PING packets	2 ms	2.2 ms	1 ms
Iperf (TCP)	15.7 Mbps	11 Mbps	94 Mbps
File Transfer (TCP) (20 MB)	13.54 Mbps	8.8 Mbps	90 Mbps

Conclusions

- For performance improvement, write a netfilter hook (loadable kernel module) instead of mangling packets in user space – will need to see how encryption can be done here
- How does this perform encryption with respect to other tools
- Is this value for money?

References

- http://sock-raw.org/papers/sock_raw
- https://home.regit.org/netfilter-en/using-nfqueue-and-libnetfilter_queue/
- http://www.netfilter.org/projects/iptables/index.html

Questions ?

- Thank you!