BKK19-504: XDP offload for OPC UA

Maxim Uvarov maxim.uvarov@linaro.org
Introduction: OPC UA

**OPC Unified Architecture (OPC UA)** is a machine to machine communication protocol for industrial automation developed by the OPC Foundation. [https://opcfoundation.org](https://opcfoundation.org)

OPC UA goals:

- **Functional equivalence**: all COM OPC Classic specifications are mapped to UA
- **Platform independence**: from an embedded micro-controller to cloud-based infrastructure
- **Secure**: encryption, authentication, and auditing
- **Extensible**: ability to add new features without affecting existing applications
- **Comprehensive information modeling**: for defining complex information
Introduction: open62541

Open62541 is implementation of OPC UA following IEC 62541 standard.

https://open62541.org/

https://github.com/open62541/open62541/

- Open source license (Mozilla Public License v2.0)
- Portable (Linux, Windows, Microcontrollers).
  - arch/freertosLWIP
  - arch/posix
  - arch/vxworks
  - arch/win32
Goal of the work:

- Lower latency for OPC UA messages (network packets).
- Use ‘standard’ linux environment
Introduction: AF_XDP sockets

- Exist in Linux since 4.19
- Skb, non Skb and zero copy modes
- Remove Linux kernel network stack from process chain

```
fd = socket(PF_XDP, SOCK_RAW, 0);
```
Initial idea:

- Open62541 example app.
- libopen62541
- Posix + LWIP
- AF_XDP raw sockets
- CPU0
- CPUx isolated

Possible LD_PRELOAD AF_XDP functions here
IPXDP repo:

https://github.com/muvarov/ipxdp

- Libopen62541 with hybrid Posix LWIP arch.
- AF_XDP LWIP port.
- LWIP netperf3 to validate that code works.
- Open6251demos and server for all available sensors (lmsensors). Demos work over TCP protocol.
- Patches to measure packet delivery times.
IPXDP results numbers:

```
./iperf3 -s
```

Native UDP 100Mbit

```
Server listening on 5201
```

```
Accepted connection from 192.168.1.100, port 51112
[  5] local 192.168.1.200 port 5201 connected to 192.168.1.100 port 51114
[ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00 sec  10.7 MBytes  89.4 Mbits/sec
[  5]   1.00-2.00 sec  11.1 MBytes  93.4 Mbits/sec
[  5]   2.00-3.00 sec  11.1 MBytes  93.4 Mbits/sec
[  5]   3.00-4.00 sec  11.1 MBytes  93.4 Mbits/sec
[  5]   4.00-5.00 sec  11.1 MBytes  93.4 Mbits/sec
[  5]   5.00-6.00 sec  11.1 MBytes  93.4 Mbits/sec
[  5]   6.00-7.00 sec  11.1 MBytes  93.4 Mbits/sec
[  5]   7.00-8.00 sec  11.1 MBytes  93.4 Mbits/sec
[  5]   8.00-9.00 sec  11.1 MBytes  93.4 Mbits/sec
[  5]   9.00-10.00 sec  11.1 MBytes  93.4 Mbits/sec
[  5]  10.00-10.04 sec  491 KBytes  93.1 Mbits/sec
```

AF_XDP 100Mbit (XDP_SKB)

```
iperf_run_server() 427
Accepted connection from ::d446:9da6:ff7f:0, port 51282
fcntl(F_SETFL): Function not implemented
[  2] local ::1c0a:c854:0:0 port 5201 connected to ::ece0:18c5:f07f:0 port 51284
[ID] Interval           Transfer     Bitrate
[  2]   0.00-1.01 sec  11.4 MBytes  94.1 Mbits/sec
[  2]   1.01-2.02 sec  11.2 MBytes  94.1 Mbits/sec
[  2]   2.02-3.02 sec  11.2 MBytes  94.2 Mbits/sec
[  2]   3.02-4.02 sec  11.2 MBytes  94.1 Mbits/sec
[  2]   4.02-5.01 sec  11.1 MBytes  94.2 Mbits/sec
[  2]   5.01-6.02 sec  11.2 MBytes  93.8 Mbits/sec
[  2]   6.02-7.02 sec  11.2 MBytes  94.2 Mbits/sec
[  2]   7.02-8.01 sec  11.1 MBytes  94.2 Mbits/sec
[  2]   8.01-9.01 sec  11.2 MBytes  94.1 Mbits/sec
[  2]   9.01-10.02 sec  11.2 MBytes  94.2 Mbits/sec
[  2]  10.02-11.02 sec  11.2 MBytes  94.2 Mbits/sec
[  2]  11.02-12.02 sec  11.2 MBytes  94.2 Mbits/sec
[  2]  12.02-13.01 sec  11.1 MBytes  94.1 Mbits/sec
[  2]  13.01-14.02 sec  11.2 MBytes  94.2 Mbits/sec
[  2]  14.02-15.02 sec  11.2 MBytes  94.2 Mbits/sec
```
## Strace AF_XDP IWIP netperf

### strace output

```plaintext
strace: Process 1143 detached
% time   seconds  usecs/call  calls  errors syscall
---------- ----------- ----------- --------- --------- ----------------
80.39    1.179556           6    193135     11529 futex
19.28    0.282908           3     99857         1 sendto
0.14    0.002005         167        12           read
0.07    0.001025           2     440           clock_gettime
0.04    0.000610          14        11           mprotect
0.02    0.000312           9        33           open
0.02    0.000221          11        21           write
0.01    0.000154           7        12           mmap
0.01    0.000140          13        11           write
0.01    0.000094           8        12           fstat
0.01    0.000082           7        12           close
0.00    0.000032           4         8           access
0.00    0.000028           9         3           brk
0.00    0.000021          21         1           munmap
```
IWIP summary:

- LWIP stack itself needs optimizations
- Implementation itself has a lot of configuration options needed the best of them

But:
- Can be used for latency and network footprint optimizations.
Idea 2: remove LWIP and hack OPC code

This experiment is to use server for UDP multicast messages crafter from raw packet. I.e. no any IP stack.

https://github.com/muvarov/opc-ua-measurements

libpbpf
AF_XDP socket
libover.so

LD_PRELOAD=libover.so ./pub_sub_app
libover.so

recvfrom() -> af_xdp_recvfrom()
select() -> empty()
accept4() -> empty()
__attribute__((constructor)) -> af_xdp_init();

3 system calls per packet goes away! 3 user / kernel context switches goes away!
examples/pubsub/tutorial_pubsub_subscribe.c

UA_Server_addRepeatedCallback(server,
(UA_ServerCallback)subscriptionPollingCallback,
    connection, 100, &subscriptionCallbackId);

Falls down to OPC UA scheduler implemented with linked list of tasks in specific time execution.

Needs to be changed to:
while(1)
    subscriptionPollingCallback()
Perf results:

- Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GH
- isolcpus=$cpu
- nohz_full=$cpu
- rcu_nocbs=$cpu
Perf random spikes

Tested on laptop. No NUMA machine. Looks like some periodic fight for shared resources.
Change perf to rdtsc() on x86

- Remove perf overhead
- Hack code to use time stamp counter and measure polling time
Callback polling times with OPC US scheduler (x86)
cb poll lat 5926 ns -> 5 us
cb poll lat 5820 ns -> 5 us
cb poll lat 5928 ns -> 5 us
cb poll lat 6135 ns -> 6 us
cb poll lat 6130 ns -> 6 us
cb poll lat 5846 ns -> 5 us
cb poll lat 5858 ns -> 5 us
cb poll lat 5888 ns -> 5 us
cb poll lat 7092 ns -> 7 us
cb poll lat 5838 ns -> 5 us
cb poll lat 5927 ns -> 5 us
Remove OPC callback and do direct calls (x86)

Average polling time for RX callback is from 95 to 97 nanoseconds!

Ticks per second 2.494165
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Max ts diff is 96 ns, 0 us
Socionext ARM64: AF_XDP Driver

Max: 3470 ns
Avg: 2507 ns
Socionext ARM64: application

Max: 8700 ns
Avg: 7184 ns
Summary

AF_XDP sockets as part of mainline kernel can be used to rapidly acceleration generic networking applications.
Core isolation and latency stabilization require very careful writing of your application. No locked function, no context switches or TLB flushes.
Work in microseconds intervals is possible without current kernel modification (assuming that XDP driver exist).

But:
More speed up can be done with using data plane frameworks like DPDK or OpenDataPlane.
Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contactus@linaro.org