IPC in 1-2-3

Everything you always wanted from a network

( but were afraid to ask )

Dimitri Staessens
Sander Vrijders
Internet reliability?

DDoS attack that disrupted internet was largest of its kind in history, experts say

Dyn, the victim of last week’s denial of service attack, said it was orchestrated using a weapon called the Mirai botnet as the ‘primary source of malicious attack’

Major cyber attack disrupts internet service across Europe and US

Source: The Guardian

Internet boffins take aim at BGP route leaks

Routers should know their place

By Richard Chirgwin 19 Jun 2017 at 03:57

Source: Newsweek

BGP Routing Table Size Limit Blamed for Tuesday’s Website Outages

Many service providers experience outages, causing downtime for hosting firms and customer websites.

Source: DataCenter Knowledge

The internet is broken. You can blame sharks. And Netflix

Don’t panic, the situation isn’t as awful as it sounds, but online space is running out and undersea creatures are nibbling at some very important cables.

Source: The Register

RUSSIAN SHIPS COULD CAUSE ‘CATASTROPHE’ FOR WEST BY CUTTING TRANSATLANTIC INTERNET CABLES

Source: The Guardian

News and useful stuff

LOLcat picture

What’s eating the Internet? Photograph / Guardian

Bad news: The internet ran out of space on Tuesday. Worse news: Sharks are eating what is left.

Source: Newsweek
Internet security?

Hacker makes $84k hijacking Bitcoin mining pool

Researchers investigated after their own Bitcoin mining pool was tapped, though how hackers accessed ISP infrastructure is still not known.

The Guardian view on internet security: a huge and growing problem

The power of smartphones is too easily turned against their users. Governments, companies and users must all work together to keep themselves safe.

The Internet Is Mostly Bots

More than half of web traffic comes from automated programs—many of them malicious.

Heartbleed bug: What you need to know

This week it has emerged that a major security flaw at the heart of the internet may have been exposing users' personal information and passwords to hackers for the past two years.

Security expert Bruce Schneier described it as "catastrophic". He said: "On the scale of one to 10, this is an 11."

The BBC has attempted to round up everything you need to know about Heartbleed.
Internet privacy?

Edward Snowden: Leaks that exposed US spy programme

Edward Snowden, a former contractor for the CIA, left the US in late May after leaking to the media details of extensive internet and phone surveillance by American intelligence. Mr Snowden, who has been granted temporary asylum in Russia, faces espionage charges over his actions.

As the scandal widens, BBC News looks at the activities to light.

Source: BBC

Encryption Won’t Stop Your Internet Provider FromSpying on You

Data patterns alone can be enough to give away what video you’re watching on YouTube.

Source: The Atlantic

Verizon and AT&T accused of selling your phone number and location to almost anyone

Source: Android Authority

They Have, Right Now, Another You

Sue Halpern

Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy

by Cathy O’Neil
Crown, 259 pp., $26.00

Virtual Competition: The Promise and Perils of the Algorithm-Driven Economy

by Ariel Ezrachi and Maurice E. Stucke
Harvard University Press, 356 pp., $29.95

Source: New York review of books

PGP creator Phil Zimmermann: ‘Intelligence agencies have never had it so good’

Encryption expert says Sony Pictures hack shows why companies should value privacy as well as security, too, and is unimpressed by David Cameron’s encryption stance

Source: The Guardian
Our research methodology

**Evolutionary Prototyping**

1. Initial Concept
2. Design and implement initial prototype
3. Refine prototype until acceptable
4. Complete and release prototype

**Immediate Feedback**

*KEEP CALM AND WRITE YOUR RESEARCH PAPER*
Network Architectures

OSI model

Application
Presentation
Session
Transport
Network
Data Link
Physical

TCP model

Different layers defined by function
Network Architectures

OSI model

TCP model

Different layers defined by function?

Encryption

Technology crossover (e.g. SPB (Data Link) based on IS-IS (Network))
Network Architectures

OSI model

- Application
- Presentation
- Session
- Transport
- Presentation

Different layers defined by function?

- Encryption

Technology crossover (e.g. SPB (Data Link) based on IS-IS (Network))
Network Architectures

OSI model

Application
Presentation
Session
Transport
Network
Data Link
Physical

Different layers defined by function?

Encryption

Technology crossover (e.g. SPB (Data Link) based on IS-IS (Network))

Recursive Model

Application

Physical
Network Architectures

**OSI model**

- Application
- Presentation
- Session
- Transport
- Network
- Data Link
- Physical

Different layers defined by function?

**TCP model**

- Application
- Presentation
- Session
- Transport
- Network
- Data Link
- Physical

Encryption

Technology crossover (e.g. SPB (Data Link) based on IS-IS (Network))

**Recursive Model**

- Application
- Layer
- Layer
- Layer
- Physical
Network APIs

OSI model

- Application
- Presentation
- Session
- Transport
- Network
- Data Link
- Physical

TCP model

- Application
- Layer
- Layer
- Layer
- Physical

Recursive Model

- Application
- Layer
- Layer
- Physical

Identical API required

POSIX SOCKETS

AF_INET
SOCK_STREAM
AF_INET
SOCK_RAW
AF_PACKET
SOCK_RAW

AF_INET
SOCK_STREAM

AF_INET
SOCK_RAW
What is Ouroboros?

- A decentralised packet switched network
- Redesigned from the ground up, following a recursive model
- That blurs the differences between LANs, MANs, WANs and VPNs
- And provides you better services than you are used to from TCP and UDP
- With increased privacy, security and anonymity
- Using the simplest network API known to (this) man
Ouroboros API

Server

Client

Client

Server

<pid>

<pid>

layer
Ouroboros API

Server

Client

Client

Server

<pid>

<pid>

layer

"flow"

read()
write()

write()
read()
Ouroboros API

Server

Client

“flow descriptor”

fd = flow_accept()

Client

Server

<pid>

layer

“flow”

read()

write()

read()

write()
Ouroboros API

Server

Client

```
fds = flow_accept()
fds = flow_alloc(pid)
```

Client

Server

```
<pid>

layer

“flow”

read()
write()
write()
read()
```
Ouroboros API

Server

Client

fd = flow_accept()

fd = flow_alloc(pid)

Client

Server

<pid>

layer

"flow"

read()

write()

write()

read()
Ouroboros API

Server

fd = flow_accept()

Client

fd = flow_alloc(pid)

Client

<pid>

Layer

Server

<pid>

bind()

<name>

"flow"

read()

write()

write()

read()
**Ouroboros API**

**Server**
- `fd = flow_accept()`

**Client**
- `fd = flow_alloc(name)`

**Diagram:***
- Client
  - `<pid>`
  - `<name>`
  - `register()`
- Server
  - `<pid>`
  - `bind()`
- Layer
  - `flow`
  - `read()`, `write()`
- Flow
  - `read()`, `write()`
Ouroboros API

Server

Client

< pid >
< name >
< pid >

Client

Server

1 bind()

layer

< name >

register()

“flow”

read()
write()

write()
read()

fd = flow_accept()

fd = flow_alloc(name)
Ouroboros API

Server

1. bind()
2. register()
3. fd = flow_accept()
   - fd = flow_alloc(name)
   - flow_read(fd, buf, len)
   - flow_write(fd, buf, len)

Client

- flow_read(fd, buf, len)
- flow_write(fd, buf, len)

Layer

- <name>
- <pid>

"flow"
Ouroboros API

Server

- `fd = flow_accept()`
- `flow_read(fd, buf, len)`
- `flow_write(fd, buf, len)`
- `flow_dealloc(fd)`

Client

- `fd = flow_alloc(name)`
- `flow_read(fd, buf, len)`
- `flow_write(fd, buf, len)`
- `flow_dealloc(fd)`

```
write()
```

```
read()
```

```
flow
```

Layer

```
bind()
```

```
register()
```

```
<name>
```

```
<pid>
```
Ouroboros API

Server

- fd = flow_accept()
- flow_write(fd, buf, len)
- flow_dealloc(fd)

Client

- fd = flow_alloc(name)
- flow_read(fd, buf, len)
- flow_write(fd, buf, len)
- flow_dealloc(fd)

The OS/kernel doesn't know these calls...

1. bind()
2. register()
3. ???

1. <pid> <name>
2. <name> <pid>
The Ouroboros API consists of several functions for communication between clients and servers. Here are the key functions:

- **Server**
  - `fd = flow_accept()`: Accepts a new connection.
  - `flow_read(fd, buf, len)`: Reads data from the client.
  - `flow_write(fd, buf, len)`: Writes data to the client.
  - `flow_dealloc(fd)`: Deallocates a flow.

- **Client**
  - `fd = flow_alloc(name)`: Allocates a new flow.
  - `flow_read(fd, buf, len)`: Reads data from the server.
  - `flow_write(fd, buf, len)`: Writes data to the server.
  - `flow_dealloc(fd)`: Deallocates a flow.

The diagram also indicates that the OS/kernel doesn't know these calls, which might be a design choice to provide additional flexibility or control over network communication. The API is compatible with various environments:

- **User Space**
- **C89**
- **POSIX 2001/2008**

The diagram includes logos for various operating systems such as GNU/Linux, FreeBSD, macOS Sierra, Windows 10, and Android.
Ouroboros API

Server

Client

Server

Client

<pid>

<name>

layer

register()

bind()
int server_main(void) {
    int fd = 0;
    char buf[BUF_SIZE];
    ssize_t count = 0;

    printf("Starting the server.\n");

    while (true) {
        fd = flow_accept(NULL, NULL);
        if (fd < 0) {
            printf("Failed to accept flow.\n");
            break;
        }
        printf("New flow.\n");
        count = flow_read(fd, &buf, BUF_SIZE);
        if (count < 0) {
            printf("Failed to read SDU.\n");
            flow_dealloc(fd);
            continue;
        }
        printf("Message from client is %.*s.\n", (int) count, buf);
        if (flow_write(fd, buf, count) == -1) {
            printf("Failed to write SDU.\n");
            flow_dealloc(fd);
            continue;
        }
        flow_dealloc(fd);
    }
    return 0;
}

int client_main(void) {
    int fd = 0;
    char buf[BUF_SIZE];
    char * message = "Client says hi!";
    ssize_t count = 0;

    fd = flow_alloc("echo", NULL, NULL);
    if (fd < 0) {
        printf("Failed to allocate flow.\n");
        return -1;
    }
    if (flow_write(fd, message, strlen(message) + 1) < 0) {
        printf("Failed to write SDU.\n");
        flow_dealloc(fd);
        return -1;
    }
    count = flow_read(fd, buf, BUF_SIZE);
    if (count < 0) {
        printf("Failed to read SDU.\n");
        flow_dealloc(fd);
        return -1;
    }
    printf("Server replied with %.*s\n", (int) count, buf);
    flow_dealloc(fd);
    return 0;
}

$ echo-app -l
Starting the server.
New flow.
Message from client is Client says hi!

$ echo-app
Server replied with Client says hi!
Functions of a layer

Server

- `fd = flow_accept()`
- `flow_read(fd, buf, len)`
- `flow_write(fd, buf, len)`
- `flow_dealloc(fd)`

Client

- `fd = flow_alloc(name)`
- `flow_read(fd, buf, len)`
- `flow_write(fd, buf, len)`
- `flow_dealloc(fd)`

Ouroboros Subsystem (IRMd)

```
bind()
register()
```
NB1) The bind operation is local to the IRMd
Functions of a layer

- Keep track and figure out where there are endpoints for a certain name: directory
- Figure out how to get packets from one point to another: routing
- Forward the packets from one point to another: forwarding
- Allocate and release resources: flow allocation/deallocation

NB1) The bind operation is local to the IRMd
NB2) This is not be an exhaustive list
How do two processes on a PC communicate?

“Loopback interface”

127.0.0.1
How do two processes on a PC communicate?

- **Application**
- **Transport**
- **Network**
- **Data Link**
- **Physical**

There is a layer of communication described as the "Loopback layer" between the Client and Server.
Ouroboros Local Inter-Process Communication

```
$ sudo irmd --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started…
```

- **Ouroboros Subsystem (IRMd)**
Ouroboros Local Inter-Process Communication

$ sudo irmd --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started…

$ oping -l &
[1] 6417
Ouroboros ping server started.

irmd(DB): New instance (6417) of oping added.

Ouroboros Subsystem (IRMd)
Ouroboros Local Inter-Process Communication

$ sudo irmd --stdout

irmd(II): Ouroboros IPC Resource Manager daemon started…

irmd(II): Bound process 6417 to name server.

$ irm bind process 6417 name server

$ oping -l &
[1] 6417
Ouroboros ping server started.

Ouroboros Local Inter-Process Communication

$ sudo irmd --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started...
irmd(II): Bound process 6417 to name server.
irmd(II): Created IPCP 6532.
irmd(II): Bootstrapped IPCP 6532 in layer local.

$ irm bind process 6417 name server
$ irm ipcp bootstrap type local name local layer local

$ oping -l &
[1] 6417
Ouroboros ping server started.

ipcpd-local(II): Bootstrapped local IPCP with pid 6532.

Ouroboros
Subsystem (IRMd)

Server

Local layer

$ sudo irmd --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started...
irmd(II): Bound process 6417 to name server.
irmd(II): Created IPCP 6532.
irmd(II): Bootstrapped IPCP 6532 in layer local.
Ouroboros Local Inter-Process Communication

$ sudo irmd --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started…
irmd(II): Bound process 6417 to name server.
irmd(II): Created IPCP 6532.
irmd(II): Bootstrapped IPCP 6532 in layer local.
irmd(II): Registered server in local as 081afc84.

$ irm bind process 6417 name server
$ irm ipcp bootstrap type local name local layer local
$ irm register name server layer local

Oping -l &
[1] 6417
Ouroboros ping server started.

ipcpd-local(II): Bootstrapped local IPCP with pid 6532.
ipcpd-local(II): Registered 081afc84.

$ sudo irmd --stdout
irm(II): Ouroboros IPC Resource Manager daemon started…
irm(II): Bound process 6417 to name server.
irm(II): Created IPCP 6532.
irm(II): Bootstrapped IPCP 6532 in layer local.
irm(II): Registered server in local as 081afc84.
**Ouroboros Local Inter-Process Communication**

1. `$ irm bind process 6417 name server`
   - Pinging server with 64 bytes of data:
     - 64 bytes from server: seq=0 time=0.480 ms
     - 64 bytes from server: seq=1 time=0.268 ms
     - 64 bytes from server: seq=2 time=0.239 ms

2. `$ irm ipcp bootstrap type local name local layer local`
   - Ouroboros ping server started.
   - New flow 64.

3. `$ oping -l &`
   - `Client`
   - 6417 - server
   - `Server`
   - `Local layer`

   ipcpd-local(II): Bootstrapped local IPCP with pid 6532.
   ipcpd-local(II): Registered 081afc84.
   ipcpd-local(II): Pending local allocation request on fd 64.
   ipcpd-local(II): Flow allocation completed, fds (64, 65).

4. `$ sudo irmd --stdout`
   - irmd(II): Ouroboros IPC Resource Manager daemon started…
   - irmd(II): Bound process 6417 to name server.
   - irmd(II): Created IPCP 6532.
   - irmd(II): Bootstrapped IPCP 6532 in layer local.
   - irmd(II): Registered server in local as 081afc84.
   - irmd(II): Flow request arrived for server.
   - irmd(II): Flow on port_id 0 | 1 allocated.
$ sudo irmd --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started...
irmd(II): Bound process 6417 to name server.
irmd(II): Created IPCP 6532.
irmd(II): Bootstrapped IPCP 6532 in layer local.
irmd(II): Registered server in local as 081afc84.
irmd(II): Flow request arrived for server.
irmd(II): Flow on port_id 0 allocated.
irmd(II): Completed deallocation of port_id 0 by process 6532.

$ oping -n server
Ping server with 64 bytes of data:
64 bytes from server: seq=0 time=0.480 ms
64 bytes from server: seq=1 time=0.268 ms
64 bytes from server: seq=2 time=0.239 ms
^C64 bytes from server: seq=3 time=0.263 ms

$ oping -l &
[1] 6417
Ouroboros ping server started.
New flow 64.
Flow 64 timed out.

$ irm bind process 6417 name server
$ irm ipcp bootstrap type local name local layer local
$ irm register name server layer local

--- server ping statistics ---
4 SDUs transmitted, 4 received, 0% packet loss, time: 4001.325 ms
rtt min/avg/max/mdev = 0.239/0.312/0.480/0.112 ms

ipcpd-local(II): Bootstrapped local IPCP with pid 6532.
ipcpd-local(II): Registered 081afc84.
ipcpd-local(II): Pending local allocation request on fd 64.
ipcpd-local(II): Flow allocation completed, fds (64, 65).
ipcpd-local(II): Flow with fd 64 deallocated.
ipcpd-local(II): Flow with fd 65 deallocated.

$ sudo irmd --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started...
irmd(II): Bound process 6417 to name server.
irmd(II): Created IPCP 6532.
irmd(II): Bootstrapped IPCP 6532 in layer local.
irmd(II): Registered server in local as 081afc84.
irmd(II): Flow request arrived for server.
irmd(II): Flow on port_id 0 | 1 allocated.
irmd(II): Completed deallocation of port_id 0 | 1 by process 6532.
Ouroboros over layer X
Networking Ouroboros

Client
Layer X
Server

Ouroboros Subsystem (IRMd)
Networking Ouroboros

Client

Server

Layer X

Ouroboros API

Ouroboros API
Ouroboros over Ethernet

```
$ sudo irmd --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started…
```

Ouroboros
Subsystem
(IRMd)
Ouroboros over Ethernet

```
$ sudo irmd --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started...
irmd(II): Created IPCP 19591.
ipcpd/eth-llc(II): Using raw socket device.
irmd(II): Bootstrapped IPCP 19591 in layer ethernet.
```

```
$ irm b t eth-llc l ethernet n eth if wlp2s0
```

Ouroboros Subsystem (IRMd)

Eth IPCP
Ouroboros over Ethernet

Ouroboros Subsystem (IRMd)

$ irm -b t eth-llc l ethernet n eth if wlp2s0
$ irm reg n ioq3 l ethernet

$ sudo irmd --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started…
irmd(II): Created IPCP 19591.
ipcpd/eth-llc(II): Using raw socket device.
irmd(II): Bootstrapped IPCP 19591 in layer ethernet.
irmd(II): Registered ioq3 in ethernet as c8a3f205.

Eth IPCP

c8a3f205
Ouroboros over Ethernet

$ sudo irmd --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started…
irmd(II): Created IPCP 19591.
ipcpd/eth-llc(II): Using raw socket device.
irmd(II): Bootstrapped IPCP 19591 in layer ethernet.
irmd(II): Registered ioq3 in ethernet as c8a3f205.
irmd(II): Bound program <path>/ioq3ded.x86_64 to name ioq3.

$ irm i b t eth-llc l ethernet n eth if wlp2s0
$ irm reg n ioq3 l ethernet
$ irm b prog ./ioq3ded.x86_64 n ioq3
Ouroboros over Ethernet

Server

Eth IPCP

c8a3f205

$ sudo irmd --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started…
irmd(II): Created IPCP 19591.
ipcpd/eth-llc(II): Using raw socket device.
irmd(II): Bootstrapped IPCP 19591 in layer ethernet.
irmd(II): Registered ioq3 in ethernet as c8a3f205.
irmd(II): Bound program <path>/ioq3ded.x86_64 to name ioq3.
irmd(DB): Process 8976 inherits name ioq3 from program ioq3ded.x86_64.

Ouroboros Subsystem (IRMd)

$ ./ioq3ded.x86_64 <params>
ioq3 1.36_GIT_71bd8d10-2018-02-02 linux-x86_64 Feb 2 2018 ...

$ irm b t eth-llc l ethernet n eth if wlp2s0
$ irm reg n ioq3 l ethernet
$ irm b prog ./ioq3ded.x86_64 n ioq3
Ouroboros over Ethernet

- `$ sudo irmd --stdout`
  - irmd(II): Ouroboros IPC Resource Manager daemon started...

- `$ ./ioq3ded.x86_64 <params>`
  - ioq3 1.36_GIT_71bd8d10-2018-02-02 linux-x86_64 Feb 2 2018...

- `$ sudo irmd --stdout`
  - irmd(II): Ouroboros IPC Resource Manager daemon started...
    - irmd(II): Created IPCP 19591.
    - irmd(II): Bootstrapped IPCP 19591 in layer ethernet.
    - irmd(II): Registered ioq3 in ethernet as c8a3f205.
    - irmd(II): Bound program <path>/ioq3ded.x86_64 to name ioq3.
    - irmd(DB): Process 8976 inherits name ioq3 from program ioq3ded.x86_64.

- `$ irm i b t eth-llc l ethernet n eth if wlp2s0`
  - $ irm reg n ioq3 l ethernet
  - $ irm b prog ./ioq3ded.x86_64 n ioq3
Ouroboros over Ethernet

$ sudo irm --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started...
irmd(II): Created IPCP 6268.
ipcpd/eth-llc(II): Using raw socket device.
irmd(II): Bootstrapped IPCP 6268 in layer ethernet.

$ sudo irmd --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started...
irmd(II): Created IPCP 19591.
ipcpd/eth-llc(II): Using raw socket device.
irmd(II): Bootstrapped IPCP 19591 in layer ethernet.
irmd(II): Registered ioq3 in ethernet as c8a3f205.
irmd(II): Bound program <path>/ioq3ded.x86_64 to name ioq3.
irmd(DB): Process 8976 inherits name ioq3 from program ioq3ded.x86_64.

$ irm i b t eth-llc l ethernet n eth if wlp2s0

$ irm i b t eth-llc l ethernet n eth if wlp2s0

$ ./ioq3ded.x86_64 <params>
ioq3 1.36_GIT_71bd8d10-2018-02-02 linux-x86_64 Feb 2 2018
...

$irm i b t eth-llc l ethernet n eth if wlp2s0
$ irm reg n ioq3 l ethernet
$ irm b prog ./ioq3ded.x86_64 n ioq3
**Ouroboros over Ethernet**

**Client**

```bash
$ sudo irmd --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started...
irmd(II): Created IPCP 6268.
ipcpd/eth-llc(II): Using raw socket device.
irmd(II): Bootstrapped IPCP 6268 in layer ethernet.

$ ./ioquake3.x86_64 +set com basegame baseoa
] connect -O ioq3
```

**Server**

```bash
$ sudo irmd --stdout
irmd(II): Ouroboros IPC Resource Manager daemon started...
irmd(II): Created IPCP 19591.
ipcpd/eth-llc(II): Using raw socket device.
irmd(II): Bootstrapped IPCP 19591 in layer ethernet.
irmd(II): Registered ioq3 in ethernet as c8a3f205.
irmd(II): Bound program <path>/ioq3ded.x86_64 to name ioq3.
irmd(DB): Process 8976 inherits name ioq3 from program ioq3ded.x86_64.
irmd(II): Flow request arrived for ioq3.
```

**Ouroboros Subsystem (IRMd)**
Reliability

Client

Server

Eth IPCP Ethernet Layer Eth IPCP

Packet loss

Jitter

...
Reliability

NB3) connection management and fragmentation/encryption/checksumming is available in every process and thus not a distinct function of a layer
### Ouroboros over X

<table>
<thead>
<tr>
<th>Application Layer</th>
<th>L2</th>
<th>L3/4</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW SOCKETS</td>
<td>BPF</td>
<td>POSIX SOCKETS</td>
</tr>
<tr>
<td>BPF</td>
<td>NETMAP</td>
<td>IPv4 / UDP / DDNS</td>
</tr>
<tr>
<td>NETMAP</td>
<td>Physical</td>
<td></td>
</tr>
</tbody>
</table>

**Ouroboros over X Diagram**

- **L1**
  - **Application Layer**
  - **Layer**
  - **Layer**
  - **PHY IPCP**

- **L2**
  - **Application Layer**
  - **Layer**
  - **Layer**
  - **RAW SOCKETS**
  - **BPF**
  - **NETMAP**

- **L3/4**
  - **Application Layer**
  - **Layer**
  - **Layer**
  - **POSIX SOCKETS**
  - **IPv4 / UDP / DDNS**

**Table: Flow Allocation, Routing, Forwarding, and Directory**

<table>
<thead>
<tr>
<th>Flow Allocation</th>
<th>Routing</th>
<th>Forwarding</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>raptor</td>
<td>ouroboros</td>
<td>N/A</td>
<td>ouroboros</td>
</tr>
<tr>
<td>eth-llc</td>
<td>ouroboros</td>
<td>RSTP</td>
<td>Ethernet</td>
</tr>
<tr>
<td>udp</td>
<td>ouroboros</td>
<td>OSPF</td>
<td>IP</td>
</tr>
</tbody>
</table>

**Diagram Elements**

- **RAPTOR NetFPGA 10G**

**Key**

- **Application**
- **Layer**
- **UDP IPCP**
- **Network**
- **Data Link**
- **Physical**
Recursive model

System 1

System 2

Application

Ethernet Layer
Recursive model

System 1          System 2
Normal IPCP       Normal IPCP
Ethernet Layer
Recursive model

Extend the scope with a normal layer!

System 1  System 2  System 3
Recursive model

System 1
Ethernet Layer
Normal Layer
System 2
Ethernet Layer
System 3
Recursive model

System 1

System 2

System 3
Recursive model

Application

Ethernet Layer

Normal Layer

Ethernet Layer

System 1

System 2

System 3
Let’s keep recursing!
Let’s keep recursing!
Within each normal layer...
Within each normal layer...

All equal, completely decentralized
The main objective of a normal layer
Complexity reduced through synergy
Complexity reduced through synergy
Complexity reduced through synergy
Bootstrapping the normal layer
Bootstrapping the normal layer

Usage: irm ipcp bootstrap name <ipcp name>
layer <layer name>
type [TYPE]
where TYPE = {normal local udp eth-llc raptor},

if TYPE == normal
[addr <address size> (default: 4)]
[fd <fd size> (default: 2)]
[ttl (add time to live value in the PCI)]
[addr_auth <ADDRESS_POLICY> (default: flat)]
[routing <ROUTING_POLICY> (default: link_state)]
[pff [PFF_POLICY] (default: simple)]
[hash [ALGORITHM] (default: SHA3_256)]
[autobind]
where ADDRESS_POLICY = {flat}
ROUTING_POLICY = {link_state lfa}
PFF_POLICY = {simple alternate}
ALGORITHM = {SHA3_224 SHA3_256 SHA3_384 SHA3_512}
...

$ irm ipcp bootstrap
Bootstrapping the normal layer

**Ethernet IPCP “e1”**

**Ouroboros Subsystem (IRMd)**

$ irm ipcp bootstrap name e1 type eth-llc if eth0 layer e

$ sudo irmd --stdout

irmd(II): Created IPCP 16357.

ipcpd/eth-llc(DB): Bootstrapped IPCP over Ethernet with LLC with pid 16357.

$ sudo irmd --stdout

irmd(II): Created IPCP 16357.

ipcpd/eth-llc(DB): Bootstrapped IPCP over Ethernet with LLC with pid 16357.
Bootstrapping the normal layer

Normal IPCP “n1”

Ethernet IPCP “e1”

Ouroboros Subsystem (IRMd)

16373 - n1
16373 - n

sudo irmd --stdout
irmd(II): Created IPCP 16357.
directory(II): Directory bootstrapped.

ipcpd/eth-llc(II): Using raw socket device.
ipcpd/eth-llc(DB): Bootstrapped IPCP over Ethernet with LLC with pid 16357.

$ sudo irmd --stdout
irmd(II): Created IPCP 16357.
irmd(II): Bootstrapped IPCP 16357 in layer e.
irmd(II): Created IPCP 16373.
irmd(II): Bound process 16373 to name n1.
irmd(II): Bound process 16373 to name n.
irmd(II): Bootstrapped IPCP 16373 in layer n.

irmd(DB): New instance (16373) of ipcpd-normal added.
irmd(DB): This process accepts flows for:

$ irm ipcp bootstrap name e1 type eth-llc if eth0 layer e
$ irm ipcp bootstrap name n1 type normal layer n autobind
Bootstrapping the normal layer

$ sudo irmd --stdout
...
irmd(II): Registered n1 in e as 7976a340.
irmd(II): Registered n in e as 3250c7b0.

Ouroboros Subsystem (IRMd)

Normal IPCP “n1”

Ethernet IPCP “e1”

normal-ipcp(DB): IPCP got address 1860022337.
directory(DB): Bootstrapping directory.
directory(II): Directory bootstrapped.

ipcpd/eth-llc(II): Using raw socket device.
ipcpd/eth-llc(DB): Bootstrapped IPCP over Ethernet with LLC with pid 16357.

$ irm ipcp bootstrap name e1 type eth-llc if eth0 layer e
$ irm ipcp bootstrap name n1 type normal layer n autobind
$ irm register name n1 layer e
$ irm register name n layer e
$ sudo irmd --stdout
...
irmd(II): Registered n1 in e as 7976a340.
irmd(II): Registered n in e as 3250c7b0.
Enrolling into a layer
Enrolling the normal layer

Authenticate, obtain configuration, obtain address, ...

System 1  System 2
Enrollment normal layer

Normal IPCP “n1”

Ethernet IPCP “e1”

Ouroboros Subsystem (IRMd)

7976a340

3250c7b0

Ethernet IPCP “e2”

Ouroboros Subsystem (IRMd)

16373 - n1

16373 - n

System 1

System 2

$ irm ipcp bootstrap name e1 type eth-llc if eth0 layer e

$ irm ipcp bootstrap name n1 type normal layer n autobind

$ irm register name n1 layer e

$ irm register name n layer e

$ irm ipcp bootstrap name e2 type eth-llc if eth0 layer e
Enrollment normal layer

enrollment(DB): Enrolling a new neighbor.
enrollment(DB): Sending enrollment info (38 bytes).
enrollment(DB): Neighbor enrollment successful.

Normal IPCP “n1”

Ethernet IPCP “e1”

7976a340
3250c7b0

Ouroboros Subsystem (IRMd)

16373 - n1
16373 - n

System 1

$ irm ipcp bootstrap name e1 type eth-llc if eth0 layer e
$ irm ipcp bootstrap name n1 type normal layer n autobind
$ irm register name n1 layer e
$ irm register name n layer e

$ irm ipcp bootstrap name e2 type eth-llc if eth0 layer e

Normal IPCP “n2”

Ethernet IPCP “e2”

654 - n2
654 - n

Ouroboros Subsystem (IRMd)

654 - n

normal-ipcp(DB): IPCP got address 65925404.

enrollment(DB): Getting boot information.
enrollment(DB): Received enrollment info (38 bytes).

System 2
enrollment(DB): Enrolling a new neighbor.
enrollment(DB): Sending enrollment info (38 bytes).
enrollment(DB): Neighbor enrollment successful.

System 1

$ irm ipcp bootstrap name e1 type eth-llc if eth0 layer e
$ irm ipcp bootstrap name n1 type normal layer n autobind
$ irm register name n1 layer e
$ irm register name n layer e

System 2

$ irm ipcp bootstrap name e2 type eth-llc if eth0 layer e
$ irm ipcp enroll name n2 layer n autobind
$ irm register name n2 layer e
$ irm register name n layer e

enrollment(DB): Getting boot information.
enrollment(DB): Received enrollment info (38 bytes).
normal-ipcp(DB): IPCP got address 65925404.

Normal IPCP “n1”
Ethernet IPCP “e1” 3250c7b0
Ouroboros Subsystem (IRMd) 16373 - n1 16373 - n

Normal IPCP “n2”
Ethernet IPCP “e2” 3250c7b0
Ouroboros Subsystem (IRMd) 654 - n2 654 - n
Data transfer connectivity graph
Data transfer connectivity graph
Data transfer connectivity graph

Flow endpoint

1. IPCP 125456
2. IPCP 5985436
3. IPCP 9516851
4. IPCP 65925404
5. IPCP 5644615
6. IPCP 1860022337
7. IPCP 65925404
8. IPCP 25615196
9. IPCP 9951515
10. IPCP 9516851
11. IPCP 25615196
12. IPCP 5985436

Flow endpoint
Data transfer connectivity graph

- Destination address
- TTL
- Endpoint identifier

1. IPCP
   - 125456

2. IPCP
   - 5985436

3. IPCP
   - 5644615

4. IPCP
   - 1860022337

5. IPCP
   - 65925404

6. IPCP
   - 5644615

7. IPCP
   - 125456

8. IPCP
   - 5985436

9. IPCP
   - 5644615

10. IPCP
    - 1860022337

11. IPCP
    - 65925404

12. IPCP
    - 25615196

13. IPCP
    - 6959551

14. IPCP
    - 9951515

15. IPCP
    - 9516851

16. IPCP
    - 65925404

17. IPCP
    - 25615196

18. IPCP
    - 6959551

19. IPCP
    - 9951515

20. IPCP
    - 9516851
Data transfer connection

connection-manager(DB): Sending cacep info for protocol dtp to fd 65.
dt-ae(DB): Added fd 65 to SDU scheduler.
link-state-routing(DB): Type dt neighbor 142626484 added.

Normal IPCP “n1”

Ethernet IPCP “e1”
7976a340 3250c7b0

Ouroboros Subsystem (IRMd)
16373 - n1
16373 - n

Normal IPCP “n2”

Ethernet IPCP “e2”
0808f8bd 3250c7b0

Ouroboros Subsystem (IRMd)
654 - n2
654 - n

$ irm ipcp connect name n1 component dt dst n2

System 1

System 2
dt-ae(DB): Added fd 65 to SDU scheduler.
link-state-routing(DB): Type dt neighbor 559955924 added.
dt-ae(DB): Could not get nhop for addr 559955924.
dt-ae(DB): Could not get nhop for addr 559955924.
dht(DB): Enrollment of DHT completed.
Management connectivity graph

IPCP

IPCP

IPCP

IPCP

IPCP

IPCP

IPCP
Management connectivity graph

- IPCP
- IPCP
- IPCP
- IPCP
- IPCP
- IPCP
- IPCP
- IPCP
- IPCP
- IPCP
- Management info
Management connection

link-state-routing(DB): Type mgmt neighbor 142626484 added.

Normal IPCP “n1”

Ethernet IPCP “e1” 3250c7b0

Ouroboros Subsystem (IRMd) 16373 - n1 16373 - n

System 1

connection-manager(DB): Sending cacep info for protocol LSP to fd 64.
link-state-routing(DB): Type mgmt neighbor 559955924 added.

Normal IPCP “n2”

Ethernet IPCP “e2” 3250c7b0

Ouroboros Subsystem (IRMd) 654 - n2 654 - n

System 2

$ irm ipcp connect name n2 component mgmt dst n1
# Ouroboros over Ouroboros

<table>
<thead>
<tr>
<th>Layer</th>
<th>flow allocation</th>
<th>routing</th>
<th>forwarding</th>
<th>directory</th>
<th>enrollment</th>
</tr>
</thead>
<tbody>
<tr>
<td>raptor</td>
<td>ouroboros</td>
<td>N/A</td>
<td>N/A</td>
<td>ouroboros</td>
<td>N/A</td>
</tr>
<tr>
<td>eth-llc</td>
<td>ouroboros</td>
<td>RSTP</td>
<td>Ethernet</td>
<td>ouroboros</td>
<td>N/A or WiFi</td>
</tr>
<tr>
<td>udp</td>
<td>ouroboros</td>
<td>OSPF</td>
<td>IP</td>
<td>DDNS</td>
<td>N/A</td>
</tr>
<tr>
<td>ouroboros</td>
<td>ouroboros</td>
<td>IS-IS</td>
<td>ouroboros</td>
<td>DHT</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Reliability (revisited)

NB3) connection management is available in every process and thus not a distinct function of a layer.
Asynchronous I/O API

while (true) {
    fd = flow_accept(&qs, NULL);
    if (fd < 0) {
        printf("Failed to accept flow.\n");
        break;
    }
    printf("New flow %d.\n", fd);
    clock_gettime(CLOCK_REALTIME, &now);
    pthread_mutex_lock(&server.lock);
    fset_add(server.flows, fd);
    server.times[fd] = now;
    pthread_mutex_unlock(&server.lock);
    fcntl(fd, FLOWSFLAGS, FLOWFNONBLOCK | FLOWFRDWR);
}

while (true) {
    if (fevent(server.flows, server.fq, &timeout) == -ETIMEDOUT)
        continue;
    while ((fd = fqueue_next(server.fq)) >= 0) {
        msg_len = flow_read(fd, buf, OPING_BUF_SIZE);
        if (msg_len < 0)
            continue;
        if (ntohl(msg->type) != ECHO_REQUEST) {
            printf("Invalid message on fd %d.\n", fd);
            continue;
        }
        clock_gettime(CLOCK_REALTIME, &now);
        pthread_mutex_lock(&server.lock);
        server.times[fd] = now;
        pthread_mutex_unlock(&server.lock);
        msg->type = htonl(ECHO_REPLY);
        if (flow_write(fd, buf, msg_len) < 0)
            printf("Error writing to flow (fd %d).\n", fd);
    }
    clock_gettime(CLOCK_REALTIME, &now);
    pthread_mutex_lock(&server.lock);
    server.times[fd] = now;
    pthread_mutex_unlock(&server.lock);
    msg->type = htonl(ECHO_REPLY);
    if (flow_write(fd, buf, msg_len) < 0)
        printf("Error writing to flow (fd %d).\n", fd);
Wrapping Up
Ouroboros

- Provides a single layer abstraction
- Simplifies writing distributed applications
- Simplifies managing distributed applications
- Provides a secure and trustworthy network design
- Hides complexity
Ouroboros is more anonymous and secure

- Encrypted data
- Does not contain source address in header
- Completely decentralized
- Self-contained
- Registered as a hash
The future…

- Research
  - Distributed address assignment
  - Efficient layer designs (routing, resource allocation, interactions)
  - Efficient congestion control
  - IoT devices
  - ...

- Implementation
  - Bug fixing
  - Optimization
  - Lockless data structures
  - Encryption (ECDH - AES)
  - ...

- Deployment
  - Porting applications
  - Sockets emulator
  - ....
Join us...

#ouroboros
freenode

ouroboros@freelists.org

https://ouroboros.ilabt.imec.be
Acknowledgements

The development of Ouroboros was partly funded by the Flemish Government under grant no G045315N.

We would like to thank our colleagues for their feedback to improve this presentation.

We would like to thank our European and US project partners for all the valuable discussions on recursive network architectures.

We would like to thank our past and current master thesis students for their work on the prototypes.

We would like to thank our supervisors for the opportunity for us to work on this ambitious project.
That’s all we could cram into 50 minutes folks!!