TIPC
Transparent Inter Process Communication
by Jon Maloy
PRESENTATION OVERVIEW

- Introduction to TIPC
- Demo
- Current status
- Roadmap 2015-2016
TIPC FEATURES

“All-in-one” L2 based messaging service

→ Reliable datagram unicast, anycast and multicast
→ Connections with stream or message transport
→ Location transparent service addressing
→ Multi-binding of addresses
→ Immediate auto-adaptation/failover after network changes

Service and Topology tracking function

→ Nodes, processes, sockets, addresses, connections
→ Immediate feedback about service availability or topology changes
→ Subscription/event function for service addresses
→ Fully automatic neighbor discovery
TIPC == SIMPLICITY

No need to configure or lookup (IP) addresses
   → Addresses are always valid - can be hard-coded
   → Addresses refer to services - not locations
   → Address space unique for distributed containers/name spaces

No need to configure L3 networks
   → But VLANS may be useful...

No need to supervise processes or nodes
   → No more heart-beating
   → You will learn about changes - if you want to know

Easy synchronization during start-up
   → First, bind to own service address(es)
   → Second, subscribe for wanted service addresses
   → Third, start communicating when service becomes available
THE DOMAIN CONCEPT

32-bit domain identifier

- Assigned to node, cluster and zone
- Structure <Z.C.N> where zero in a position means wildcard (~anywhere/anycast)

typedef uint32_t tipc_domain_t;
tipc_domain_t tipc_domain(unsigned int zone,
                         unsigned int cluster,
                         unsigned int node);
SERVICE ADDRESSING

“Well known port number” assigned by developer

→ 32-bit service type number – typically hard-coded
→ 32-bit service instance – typically calculated
→ 32-bit domain identity
  → Indicating visibility scope on the binding side
  → Indicating lookup scope on the calling side

struct tipc_addr{
    uint32_t type;
    uint32_t instance;
    tipc_domain_t domain;
};
No (almost) restrictions on how to bind service addresses

- Different service addresses can bind to the same socket
- Same service address can bind to different sockets
- Ranges of service instances can bind to a socket

```
struct tipc_addr {
    uint32_t type;
    uint32_t instance;
    tipc_domain_t domain;
};
```
MULTICAST

All servers bound to the given service address receive a copy

→ Delivery and sequentiality guaranteed socket-to-socket

```c
struct tipc_addr{
    uint32_t type;
    uint32_t instance;
    tipc_domain_t domain;
};
```

Client Process

```c
sendto(service = 42,
       instance = 2,
       domain = 0)
```

Server Process

```c
bind(service = 42,
      lower = 1,
      domain = 0)
bind(service = 42,
      instance = 2,
      domain = 0)
bind(service = 4711,
      lower = 0,
      upper = 100,
      domain = 0)
```
Location Transparency

Location of server not known by client

→ Translation service address to physical destination performed on-the-fly at source node
→ Replica of global binding table on each node
→ Very efficient hash lookup
RELIABLE DATAGRAM SERVICE

Reliable socket to socket
→ Receive buffer overload protection
→ No real flow control, messages may still be rejected

Rejected messages may be dropped or returned
→ Configurable in sending socket
→ Truncated message returned with error code

Multicast is just a special case
→ But messages can not be made returnable
Established by using service address
- Two-way setup using data-carrying messages
- Traditional TCP-style setup/shutdown also available

Stream- or message oriented
- End-to-end flow control for buffer overflow protection
- No sequence numbers, acks or retransmissions, - the link layer takes care of that

Breaks immediately if peer becomes unavailable
- Irrespective of reason
“L2.5” reliable link layer, node to node
  - Guarantees delivery and sequentiality for all messaging
  - Acts as “trunk” for multiple connections, and keeps track of those
  - Keeps track of peer node’s address bindings in local replica of the binding table

**Supervised by probing at low traffic**
  - “Lost service address” events issued for bindings from peer node if no link left
  - Breaks all connections to peer node if no link left

**Several links per node pair**
  - Load sharing or active-standby, - but maximum two active
  - Loss-free failover to remaining link if any
NEIGHBOR DISCOVERY

- **L2 connectivity** determines network
  - Neighbor discovery by L2 broadcast, qualified by a lookup domain identity
  - All qualifying nodes in the same L2 broadcast domain establish mutual links
  - One link per interface, maximum two active links per node pair
  - Each node has its own view of its environment
Users can subscribe for contents of the global address binding table

→ Receives events at each change matching the subscription

There is a match when

→ Bound/unbound instance or range overlaps with subscribed range

**SERVICE SUBSCRIPTION**

**Server Process**

- \( \text{bind}(\text{service} = 42, \text{instance} = 2, \text{domain} = 0) \)

- \( \text{bind}(\text{service} = 42, \text{instance} = 1, \text{domain} = 0) \)

**Client Process**

- \( \text{subscribe}(\text{service} = 42, \text{lower} = 0, \text{upper} = 10) \)

- Event: \( 42, 1, <1.1.7>, \text{up} \)

- Event: \( 42, 2, <1.1.8>, \text{up} \)

**Node <1.1.7>**

**Node <1.1.8>**
Special case of service subscription

- Using same mechanism, - based on service table contents
- Represented by the built-in service type zero (0 ~ “node availability”)

Client Process

```
subscribe(service = 0,
          lower = 0,
          upper = ~0)
```

Node <1.1.7>

Node <1.1.8>
WHEN TO USE TIPC

TIPC does not replace IP based transport protocols
- It is a complement to be used under certain conditions
- It is an IPC!

TIPC may be a good option if you
- Want startup synchronization for free
- Have application components that need to keep continuous watch on each other
- Need short latency times
- Traffic is heavily intra node
- Don’t want to bother with configuration
- One L2 hop is enough between your components
- Are inside a security perimeter
WHAT TIPC WILL NOT DO FOR YOU

No user-to-user acknowledging of messages

→ Only socket-to-socket delivery guaranteed
→ What if the user doesn’t process the message?
→ On the other hand, which protocol does?

No datagram transmission flow control

→ For unicast, anycast and multicast
→ Must currently be solved by user
→ We are working on the problem...

No routing

→ Only nodes on same L2 network can communicate
→ But a node may attach to several L2 networks
**DEMO SETUP**

**Messaging Client**
- A simple "Hello World" reliable datagram message exchange
- A "Hello World" message exchange used for a two-way set up a SOCK_STREAM connection
- A regular TCP-style "connect/accept" to set up a SOCK_SEQPACKET connection

**Service Topology Subscriber**
- Subscribing and receiving up/down events for server process availability
- Subscribing and receiving events about <1.1.1>’s neighbor nodes
- Remotely subscribing and receiving events about <1.1.2>’s neighbor nodes
- Subscribing and receiving events for <1.1.1>’s links
FUNCTIONALITY

Only <1.1.N> domains available
→ In reality easy to fix
→ What is the need?

Service binding table still updated by “replicast”
→ Relatively easy to fix, but has not been prioritized
→ It works fine with current cluster sizes

Dropped ambition to have TIPC-level routing between domains
→ Only direct L2 hops is supported
→ IP level routing only option, but still no official L3 bearer
→ UDP based “bearer” implementation soon ready for upstream

Container/Name Space support
→ New as from January 2015
API

Only a low-level socket C API available
   → Hard to learn and use
   → Prototype of a new, higher-level C API available

API for Python, Perl, Ruby, D
   → But not for Java

Support for TIPC in ZeroMQ
   → Not yet with full features
AVAILABILITY

Installation package available only in SLES

Earlier supported package in Debian/Ubuntu broken
  → Volunteers wanted

No package yet in Fedora/RHEL
  → We are working on this
ARCHITECTURE

tipc_sock

tipc_port

tipc_node

tipc_link

tipc_bearer

tipc_media
IMPLEMENTATION

Significant effort to improve quality and maintainability the last 2-3 years

→ Eliminated the redundant “native API” and related “port layer”
  → Only sockets are supported now
→ Reduced code bloat
→ Reduced structure interdependencies
→ Improved locking policies
  → Fewer locks, RCU locks instead of RW locks...
  → Eliminated all known risks of deadlock
→ Buffer handling
  → Much more use of sk_buff lists and other features
  → Improved and simplified fragmentation/reassembly

Support for name spaces

→ Will be very useful in the cloud
→ Enables “distributed containers”

Linuxification of code and coding style

→ Still too visible that TIPC comes from a different world
→ Adapting to kernel naming conventions
TRAFFIC CONTROL

Connection flow control is still message based
   → May potentially consume enormous amounts of memory
   → skb_truesize() in combination with out no-drop requirement is a problem
   → We think we have a solution

Link flow control still uses a fix window
   → Too simplistic
      → We need a congestion avoidance algorithm
   → Lots of unnecessary retransmits

Datagram flow control missing
   → Probably impossible to get this hundred percent safe
      → But we can make it much better than now
SCALABILITY

Largest known cluster we have seen is 72 nodes
  → Works flawlessly
  → We need to get up to hundreds of nodes
  → The link supervision scheme may become a problem

Limited domain support
  → We need support for <Z.C.N>, not only <1.1.N>
  → Makes it possible to segment TIPC networks

Name space support
  → DONE!!!
**PERFORMANCE**

Latency times better than on TCP

→ 10-20% inter-node
→ 2 to 7 times faster intra-node messaging (depends on message size)
  → We don’t use the loopback interface

Throughput still poorer than TCP

→ 55-100% of max TCP throughput inter-node
  → Seems to be very environment dependent
→ But 25-30% better than TCP intra-node
**MANAGEMENT**

**New netlink based API introduced**
- Replaces old ascii-based commands (also via netlink)
- Uses more standard features such as socket buffers, attribute nesting, sanity checks etc.
- Scales much better when clusters grow

**New user space tool “tipc”**
- Syntax inspired by “ip” tool
- Modular design inspired by git
- Uses libnl
- Replaces old “tipc-config” tool
- Part of tipc-utils package
ROADMAP 2015-2016
FUNCTIONALITY

Allowing overlapping address ranges for same type
  → Currently only limitation to service binding
  → Causes race problems sometimes
  → Proposal exists

Updating binding table by broadcast instead of replicast
  → We know how to do this
  → Compatibility biggest challenge
LIBTIPC WITH C API

Addressing in TIPC socket API

```
struct tipc_portid {
    __u32 ref;
    __u32 node;
};
struct tipc_name {
    __u32 type;
    __u32 instance;
};
struct tipc_name_seq {
    __u32 type;
    __u32 lower;
    __u32 upper;
};
#define TIPC_ADDR_NAMESEQ       1
#define TIPC_ADDR_MCAST         1
#define TIPC_ADDR_NAME          2
#define TIPC_ADDR_ID            3
struct sockaddr_tipc {
    unsigned short family;
    unsigned char   addrtype;
    signed   char    scope;
    union {
        struct tipc_portid id;
        struct tipc_name_seq nameseq;
        struct {
            struct tipc_name name;
            __u32 domain; /* 0: own zone */
        } name;
    } addr;
};
```

Addressing in TIPC C API

```
typedef uint32_t tipc_domain_t;
struct tipc_addr {
    uint32_t      type;
    uint32_t      instance;
    tipc_domain_t domain;
};
```

Service/topology subscriptions in C API

```
int tipc_topsrv_conn(tipc_domain_t topsrv_node);
int tipc_srv_subscr(int sd, uint32_t type, uint32_t lower, uint32_t upper,
        bool all, int expire);
int tipc_srv_evt(int sd, struct tipc_addr *srv,
        bool *available, bool expired);
bool tipc_srv_wait(const struct tipc_addr *srv, int expire);
int tipc_neigh_subscr(tipc_domain_t topsrv_node);
int tipc_neigh_evt(int sd, tipc_domain_t *neigh_node, bool *available);
```

http://sourceforge.net/p/tipc/tipcutils/ci/master/tree/demos/c_api_demo/tipcc.h
TRAFFIC CONTROL

Improved connection level flow control
  → Packet based instead of message based
  → Byte based does not seem feasible

Improved link level flow control
  → Adaptable window size
  → Congestion avoidance
  → SACK, FRTO ...?

Datagram and multicast congestion feedback
  → Sender socket selects least loaded destination
  → Sender socket locks or returns –EAGAIN if all destinations congested
  → Academic work ongoing to find best algorithm
SCALABILITY

Full network address space

→ Node identity \(<Z.C.N>\) instead of \(<I.1.N>\)
→ Can group nodes by discovery rules instead of VLANS

Hierarchical neighbor supervision and failure detection

→ “Biased Gossip” type algorithm?

Ring: Scales \(\sim 2*N\)

TIPC: Scales \(\sim N*(N-1)\)

TIPC/Gossip: Scales \(\sim M*N\)
PERFORMANCE

Improved link level flow control
→ Already mentioned

Separate spinlock for each parallel link to same node
→ Currently jointly covered by a “node_lock”, serializing access
→ Loss-free transitions 1-2 and 2-1 (failover) links will be a challenge

Reducing and fragmenting code sequences covered by node_lock (link_lock)
→ Gives better parallelization
→ Big potential for improvements

Dualpath connections
→ 20 Gb/s per connection?

General code optimization
→ Based on profiling
MULTICAST/BROADCAST

- **Code overhaul of broadcast link**
  - Leveraging recent changes to unicast link

- **Multicast groups**
  - Explicit membership handling

- **Transactions**
  - Ensure “all-or-nothing” delivery

- **Virtual Synchronism**
  - Ensure virtual in-order delivery
  - From different source nodes
MORE INFORMATION

TIPC project page
http://tipc.sourceforge.net/

TIPC protocol specification
http://tipc.sourceforge.net/doc/draft-spec-tipc-10.html

TIPC programmer’s guide
http://tipc.sourceforge.net/doc/tipc_2.0_prog_guide.html

TIPC C API
http://sourceforge.net/p/tipc/tipcutils/ci/master/tree/demos/c_api_demo/tipcc.h
THANK YOU
INTER-NODE THROUGHPUT (NETPERF)

Intel(R) Xeon(R) CPU E5-2658 v2 @ 2.40GHz 48G ECC ram
3.19 RC4+ kernel + busybox. No tuning done.
Netperf stream test, ixgbe NIC’s, TCP using cubic, TIPC link window=400.

TCP:
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 11.0.0.3 () port 0 AF_INET
Recv  Send  Send
Socket Socket Message Elapsed
Size  Size  Size  Time  Throughput
bytes bytes bytes secs. 10^6bits/sec

87380  16384  16384  60.00  8338.82

TIPC:
TIPC STREAM TEST to <1.1.3:3089135711>
Recv  Send  Send
Socket Socket Message Elapsed
Size  Size  Size  Time  Throughput
bytes bytes bytes secs. 10^6bits/sec

34120200  212992  212992  60.00  4351.60
### TCP Inter Node Thruoput

```
blade3 ~ # ./client_tipc -p tcp -i eth4
****** TIPC Benchmark Client Started ******
Transferring 64000 messages in TCP Throughput Benchmark
+--------------------------------------------------------------------------------------------+
|     Msg | #     |  #  Msgs/  |  Elapsed |                    Throughput                  |
| [octets] | Conns |    Conn   | [ms]     | Total [Msg/s] | Total [Mb/s] | Per Conn [Mb/s] |
|---------|------|----------|---------|--------------|-------------|----------------|----------------|
| 64      | 1    | 64000    | 115     | 552409       | 282         | 282            |
+--------------------------------------------------------------------------------------------+
| 256     | 1    | 32000    | 60      | 526099       | 1077        | 1077           |
+--------------------------------------------------------------------------------------------+
| 1024    | 1    | 16000    | 115     | 552409       | 282         | 282            |
+--------------------------------------------------------------------------------------------+
| 4096    | 1    | 8000     | 85      | 93857        | 3075        | 3075           |
+--------------------------------------------------------------------------------------------+
| 16384   | 1    | 4000     | 209     | 19134        | 2507        | 2507           |
+--------------------------------------------------------------------------------------------+
| 65536   | 1    | 2000     | 248     | 8032         | 4211        | 4211           |
+--------------------------------------------------------------------------------------------+
Completed Throughput Benchmark
****** TIPC Benchmark Client Finished ******
```

### TIPC Inter Node Thruoput

```
blade3 ~ # ./client_tipc
****** TIPC Benchmark Client Started
Transferring 64000 messages in TIPC Throughput Benchmark
+--------------------------------------------------------------------------------------------+
|     Msg | #     |  #  Msgs/  |  Elapsed |                    Throughput                  |
| [octets] | Conns |    Conn   | [ms]     | Total [Msg/s] | Total [Mb/s] | Per Conn [Mb/s] |
|---------|------|----------|---------|--------------|-------------|----------------|----------------|
| 64      | 1    | 64000    | 304     | 209847       | 107         | 107            |
+--------------------------------------------------------------------------------------------+
| 256     | 1    | 32000    | 164     | 194584       | 398         | 398            |
+--------------------------------------------------------------------------------------------+
| 1024    | 1    | 16000    | 104     | 153283       | 1255        | 1255           |
+--------------------------------------------------------------------------------------------+
| 4096    | 1    | 8000     | 86      | 92803        | 3040        | 3040           |
+--------------------------------------------------------------------------------------------+
| 16384   | 1    | 4000     | 147     | 27196        | 3564        | 3564           |
+--------------------------------------------------------------------------------------------+
| 65536   | 1    | 2000     | 249     | 8027         | 4208        | 4208           |
+--------------------------------------------------------------------------------------------+
Completed Throughput Benchmark
****** TIPC Benchmark Client Finished ******
```
LATENCY (TIPC TOOL)

TCP Inter Node

blade3 ~ # ./client_tipc -p tcp -i eth4

****** TIPC Benchmark Client Started ******

<table>
<thead>
<tr>
<th>Msg Size [octets]</th>
<th># Mgs</th>
<th>Elapsed [ms]</th>
<th>Avg round-trip [us]</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>8000</td>
<td>341</td>
<td>42.2</td>
</tr>
<tr>
<td>256</td>
<td>4000</td>
<td>232</td>
<td>58.0</td>
</tr>
<tr>
<td>1024</td>
<td>2666</td>
<td>145</td>
<td>54.39</td>
</tr>
<tr>
<td>4096</td>
<td>2000</td>
<td>497</td>
<td>248.86</td>
</tr>
<tr>
<td>16384</td>
<td>1600</td>
<td>399</td>
<td>249.15</td>
</tr>
<tr>
<td>65536</td>
<td>1333</td>
<td>665</td>
<td>499.13</td>
</tr>
</tbody>
</table>

Completed Latency Benchmark

Transferring 64000 messages in TCP Throughput Benchmark

root@tipc1:~# bmc

****** TIPC Benchmark Client Finished ******

TCP Intra Node

blade3 ~ # ./client_tipc

****** TIPC Benchmark Client Started ******

<table>
<thead>
<tr>
<th>Msg Size [octets]</th>
<th># Mgs</th>
<th>Elapsed [ms]</th>
<th>Avg round-trip [us]</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>8000</td>
<td>311</td>
<td>38.65</td>
</tr>
<tr>
<td>256</td>
<td>4000</td>
<td>209</td>
<td>52.78</td>
</tr>
<tr>
<td>1024</td>
<td>2666</td>
<td>228</td>
<td>85.51</td>
</tr>
<tr>
<td>4096</td>
<td>2000</td>
<td>249</td>
<td>124.84</td>
</tr>
<tr>
<td>16384</td>
<td>1600</td>
<td>258</td>
<td>161.65</td>
</tr>
<tr>
<td>65536</td>
<td>1333</td>
<td>652</td>
<td>489.58</td>
</tr>
</tbody>
</table>

Completed Latency Benchmark

Using server address 127.0.0.1:4711

Transferring 80000 messages in TCP Latency Benchmark

Completed Latency Benchmark

Transferring 80000 messages in TIPC Latency Benchmark

TIPC Inter Node

blade3 ~ # ./client_tipc -p tcp -i eth4

****** TIPC Benchmark Client Started ******

<table>
<thead>
<tr>
<th>Msg Size [octets]</th>
<th># Mgs</th>
<th>Elapsed [ms]</th>
<th>Avg round-trip [us]</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>8000</td>
<td>311</td>
<td>38.65</td>
</tr>
<tr>
<td>256</td>
<td>4000</td>
<td>209</td>
<td>52.78</td>
</tr>
<tr>
<td>1024</td>
<td>2666</td>
<td>228</td>
<td>85.51</td>
</tr>
<tr>
<td>4096</td>
<td>2000</td>
<td>249</td>
<td>124.84</td>
</tr>
<tr>
<td>16384</td>
<td>1600</td>
<td>258</td>
<td>161.65</td>
</tr>
<tr>
<td>65536</td>
<td>1333</td>
<td>652</td>
<td>489.58</td>
</tr>
</tbody>
</table>

Completed Latency Benchmark

Transferring 80000 messages in TIPC Throughput Benchmark

root@tipc1:~# bmc

****** TIPC Benchmark Client Finished ******

TIPC Intra Node

blade3 ~ # ./client_tipc

****** TIPC Benchmark Client Started ******

<table>
<thead>
<tr>
<th>Msg Size [octets]</th>
<th># Mgs</th>
<th>Elapsed [ms]</th>
<th>Avg round-trip [us]</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>8000</td>
<td>426</td>
<td>5.34</td>
</tr>
<tr>
<td>256</td>
<td>4000</td>
<td>219</td>
<td>5.90</td>
</tr>
<tr>
<td>1024</td>
<td>2666</td>
<td>141</td>
<td>5.12</td>
</tr>
<tr>
<td>4096</td>
<td>2000</td>
<td>121</td>
<td>6.81</td>
</tr>
<tr>
<td>16384</td>
<td>1600</td>
<td>186</td>
<td>11.36</td>
</tr>
<tr>
<td>65536</td>
<td>1333</td>
<td>328</td>
<td>24.50</td>
</tr>
</tbody>
</table>

Completed Latency Benchmark

Using server address 127.0.0.1:4711

Transferring 80000 messages in TIPC Latency Benchmark

Completed Latency Benchmark

Transferring 80000 messages in TIPC Throughput Benchmark

root@tipc1:~# bmc

****** TIPC Benchmark Client Finished ******