Towards Transport-Agnostic Middleware

Martin Sústrik
sustrik@250bpm.com
www.250bpm.com
Messaging Middleware
A layer in the network stack to manage communication between more than two endpoints.
ZeroMQ/nanomsg
1 minute overview
As a layer in the network stack it implements multiple protocols, a.k.a. “messaging patterns”.

- Request/Reply
- Publish/Subscribe
- Pipeline
- Survey
- Et c.
Publish/Subscribe
Distributing data to all interested endpoints
Request/Reply
Load-balancing tasks among stateless workers
What about the transport layer?
It's heterogenous!
int main()
{
    int s = nn_socket (AF_SP, NN_PUB);

    nn_bind (s, “tcp://eth0:5555”);
    nn_bind (s, “pgm://eth0;241.0.0.1:5555”);

    while (1) {
        nn_send (s, “ABC”, 3, 0);
        sleep (1);
    }
}

Publisher

int main()
{
    int s = nn_socket (AF_SP, NN_SUB);

    nn_connect (s, “tcp://myserver:5555”);

    while (1) {
        char buf [100];
        nn_recv (s, buf, sizeof (buf), 0);
    }
}

Subscriber
Why should this group care?
Because it's hard for the application developer to make informed decision about transport protocol to use:

- Reliable or unreliable?
- Unicast of multicast?
- Ordered or unordered?
- Pushback or no pushback?
- Widely used (TCP, UDP) or niche (SCTP)?
- Et c.
Often, informed decision can't even be made at the development time:

- Developer has little understanding of customer's deployment environment...
- Application is sold to different customers, each having different network...
- Environment is going to change in the future...
Yet, by choosing a “messaging pattern”, developer provides enough information to make an informed decision about transport protocols to use!
Example

Publish/Subcribe pattern requires transport layer not to be reliable. Reliability would mean that a single slow or dead subscriber can stop the entire distribution tree.

Preferred transport protocol is **UDP** or **DCCP**.
Different example

Request/Reply pattern requires transport layer to exercise pushback. That way the tasks can be redirected from overloaded workers to underutilised workers.

Preferred transport protocol is TCP or SCTP.
What are the implications?
Back to the heterogenous example:
No need to specify the transport protocol:

Publisher

```c
int main()
{
    int s = nn_socket (AF_SP, NN_PUB);
    nn_bind (s, "eth0:NYSE-stock-quotes");

    while (1) {
        nn_send (s, "ABC", 3, 0);
        sleep (1);
    }
}
```

Subscriber

```c
int main()
{
    int s = nn_socket (AF_SP, NN_SUB);
    nn_connect (s, "myserver:NYSE-stock-quotes");

    while (1) {
        char buf [100];
        nn_recv (s, buf, sizeof (buf), 0);
    }
}
```
What we get is clean mechanism vs. policy separation!
Developer specifies the mechanism: “NYSE stock quote feed is to use the Publish/Subscribe pattern.”

```c
int main()
{
    int s = nn_socket (AF_SP, NN_PUB);
    nn_bind (s, "eth0:NYSE-stock-quotes");
    while (1) {
        nn_send (s, "ABC", 3, 0);
        sleep (1);
    }
}
```

Mechanism is specified via transport-agnostic API.
Administrator specifies the policy:

“NYSE stock quote feed is to use PGM on the LAN and TCP over the WAN.”

NYSE-stock-quotes:
LAN: pgm
WAN: tcp

Policy is specified via transport-aware network configuration tools.
Questions?

Email: sustrik@250bpm.com