Fast-Stacking: Providing High-Performance Stack Compositions in C++

Motivation

- Internet traffic is increasing tremendously
  - IPTV, Peer-to-Peer, Web 2.0
  - High Bandwidth End-Systems
- Co-relation to physical networks
  - Ethernet Bandwidth: 10→100→1000 MBit/sec→ ... →100 GET
  - ISDN→DSL2000→DSL 16000→DSL32000→ ... →FTTH


→ CPU processing power and memory performance becomes a bottleneck

- Evolutionary approaches (e.g., the SpoVNet ariba-underlay)
  - Use existing network stacks to build overlay networks
- New architectures (e.g., the 4WARD Node Architecture)
  - Use modular compositions of new network stacks

→ Both are implemented in Software for use in experimentation platforms!

→ A efficient interface design to compose protocols and components is required

Design considerations

High-performance constraints

- Use of virtual classes only when necessary
  - Reduces overhead, allows inline optimization
- Couple components using C++ templates
  - Reduces overhead due to glue logic
  - Highly increases Performance
- Event driven, asynchronous design
  - Allows integration with simulation environments
  - No threads → Less synchronization issues

Semantic/Design constraints

- Asynchronous connection free / connection oriented I/O
  - Considered to achieve highest performance
- Flow-Control/Management
  - Support existing protocols
- Scatter/Gather Semantics
  - Reduce memory accesses dramatically
- Timers
  - Used to implement timers or special events

→ A simple, yet highly extensible interface for high-performance stack compositions

The *connector* interface

Start / Stop Module

```
void      start( );
void      stop( );
```

Information

```
endpoint_type get_endpoint();
void      set_endpoint( endpoint );
```

Input / Output

```
eventid_t send( message, endpoint | flowid );
bool      is_send_ready( msg_size, endpoint | flowid );
eventid_t send_ready( msg_size, endpoint | flowid );

eventid_t receive( message, endpoint | flowid );
bool      is_receive_ready( endpoint | flowid );
eventid_t receive_ready( endpoint | flowid );
```

Flow Management

```
void      accept();
flowid_t  setup( endpoint, parameters );
void      drop( flowid );
```

Timers / Events

```
eventid_t timer( time_duration, info );
void      cancel( eventid );
```

Asynchronous Event Listener

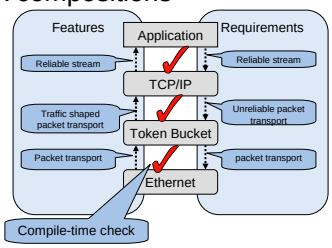
```
void      bind( listener )
void      unbind( listener )
```

Events emitted from the interface

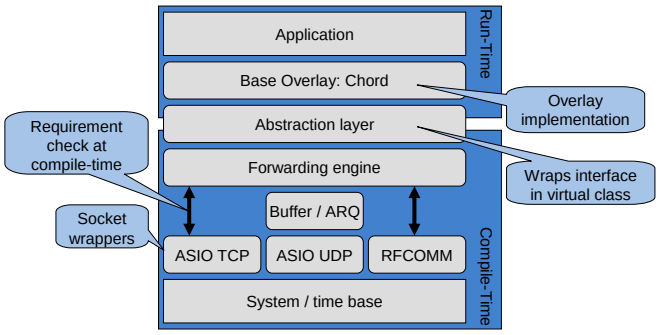
```
connector_up,  connector_down
receive_ready, receive_done, receive_fail
send_ready,    send_done,    send_fail
flow_up, flow_down, flow_request, flow_failed
timer_expired
```

Application Examples

- Pure compile-time stack compositions
  - Reuse of modules in different layers
  - Check constraints at compile-time



- Application-layer overlay application (SpoVNet)



Summary

- Reminder: Think about software design when implementing for experimentation platforms
- Presented here:
  A simple, yet efficient bundle of well-known semantics in a easy-to-use interface for compositions in C++

R.Bless, C.Hübsch, S.Mies, and O.Waldhorst. The Underlay Abstraction in the Spontaneous Virtual Networks (SpoVNet) Architecture. In Proc. *4th EuroNGI Conf. On Next Generation Internet Networks (NGI 2008)*, Apr. 2008

Ariba Virtual Network Substrate. http://www.ariba-underlay.org

Lars Völker, Denis Martin, Ibtissam El Khayat, Christoph Werle, Martina Zitterbart, : A Node Architecture for 1000 Future Networks, Proceedings of the International Workshop on the Network of the Future 2009, IEEE, Dresden, Germany, Jun 2009.

Universität Karlsruhe (TH)
Research University · founded 1825

Sebastian Mies {mies@tm.uka.de}
Institute of Telematics – Universität Karlsruhe (TH) – Germany