The Implementation of a SAHN Protocol on an Open Source Platform

Mike Tyson

Supervisors: Dr. Carlo Kopp, Dr. Bin Qiu
Overview

Introduction: Ad-hoc networks
- Monash Suburban Ad-Hoc Network introduction
- Features
- Other ad-hoc networks

System definition
- Requirements
- Architecture and interfaces

Components
- Routing
- Flow control

Discussion and further work

Conclusion
Physical infrastructure

- Communications based on physical infrastructure
  - Telephone system: Dialup, ADSL, ISDN...

- Expensive
  - Ongoing maintenance costs

- Inaccessible or of low quality in low population density regions
A Solution: Wireless Ad-Hoc Networks

- Self-forming, self-healing
- Nodes cooperate to route traffic
- Low recurring cost
- No dependency on fixed infrastructure
- Affordable broadband access where other techniques not viable
Monash Suburban Ad-Hoc Network

- Designed for suburban setting
  - Use by local residents, small businesses
  - Nodes rarely move or disappear: Quasi-static
  - Provides opportunities for optimisation over traditional ad-hoc network mechanisms

- High security
- Fine-grained resource access control
- Quality of Service (QoS) support
SAHN Features

- Encryption at the network layer
  - Guaranteed security
  - Protection from intrusion

- Distributed secure node authentication
  - Distributed certificate verification

- Optimised hybrid routing

- QoS support
  - IPv6 and ATM compatibility
Other implementations

- **LUNAR: Light-weight Underlay Network Ad-hoc Routing**
  - Aimed at small, spontaneous environments

- **MIT Roofnet**
  - Experimental network at MIT

- **Microsoft Mesh**
  - Intended for suburban, community setting

- A viable, easily available, ready-to-use implementation is not yet in existence
Research Objectives

- Develop Ad-Hoc network implementation
  - Linux 2.6 kernel module & daemon
  - Virtual network device
- Portable
- Fast
- Transparent and interoperable with existing software
- Modular

Diagram:

- Virtual SAHN device
- Real network device
- TCP/IP, etc.
Architecture

- Pseudo device driver that emulates real device.
- Character device interfaces with daemon
- User-space daemon runs SAHN.

Portability
- Decoupling from OS

Performance
- Minimal extra copying of data
Interfaces

- Shared data structure
- Packets stored in shared area
- References passed between kernel and daemon
- Internal queues maintained by kernel
- Synchronisation through `ioctl()` calls
Outgoing packets enter SAHN module from TCP/IP
SAHN module copies packet into a *sahn_pktbuf* structure in shared area, and enqueues packet
SAHN daemon reads from queue, and processes packet
On completion of processing, SAHN daemon passes reference to packet to SAHN module.

SAHN module creates `sk_buff` for packet, and passes to physical layer.
SAHN Daemon

Six primary SAHN daemon components:
**Implemented modules**

- **SAHN Routing**: Based on Dynamic Source Routing (*Johnson and Maltz, '96*), with some elements from Ad-Hoc On-Demand Distance Vector Routing (*Perkins and Royer, '99*)
  - Optimised for SAHN environment
  - Neighbour discovery
  - Route discovery
  - Route maintenance

- **Flow control**: Sliding-window, with out-of-order packet reception
Discussion

- Working architecture designed and implemented which reflects requirements:
  - Portability
  - Speed
  - Modularity
  - Transparency
- Implementation passes packets successfully, over multiple hops
  - Ping tests
  - File transfer tests with 'scp'
Further development

- Formal verification of existing system
  - More extensive testing under different conditions

- Implementation of remaining modules
  - Quality of Service
  - Security (SAHN Security Protocol)
  - Access control/management
Conclusion

- This Honours project has produced the beginnings of a working SAHN implementation.

- Within a few years, implementation will be production-grade, and ready for public use.

- Answers a need for a broadband alternative.
http://www.csse.monash.edu.au/~mtysn

References:

Backup slides follow
Backup slide 1: Architecture

- **Applications**
- **Socket layer**
- **TCP/IP**
- **Link (802.11)**
- **Physical**

Functions:
- `netif_rx()`
- `dev_queue_xmit()`
- `ioctl()`
- `hard_start_xmit()`
- `recvmsg()`

Spaces:
- **User space**
- **Kernel space**

Relationships:
- **SAHN module**
- **Char dev**

Connections:
- Shared Memory
- SAHN proc
Backup slide 2: Architecture
Backup slide 3: Data flow

RX path
1. sahn_netif_recv
2. SAHNWAIT
3. SAHNGETRXQUEUE
4. SAHNENQUEUERX
5. netif_rx

Module

TX path
1. hard_start_xmit
2. SAHNWAIT
3. SAHNGETTXQUEUE
4. SAHNENQUEUEUTX
5. dev_queue_xmit

Upper Layers

Daemon

Physical Layer
Backup slide 4: Ad-hoc scenario