SpartanRPC
Secure WSN Middleware for Cooperating Domains

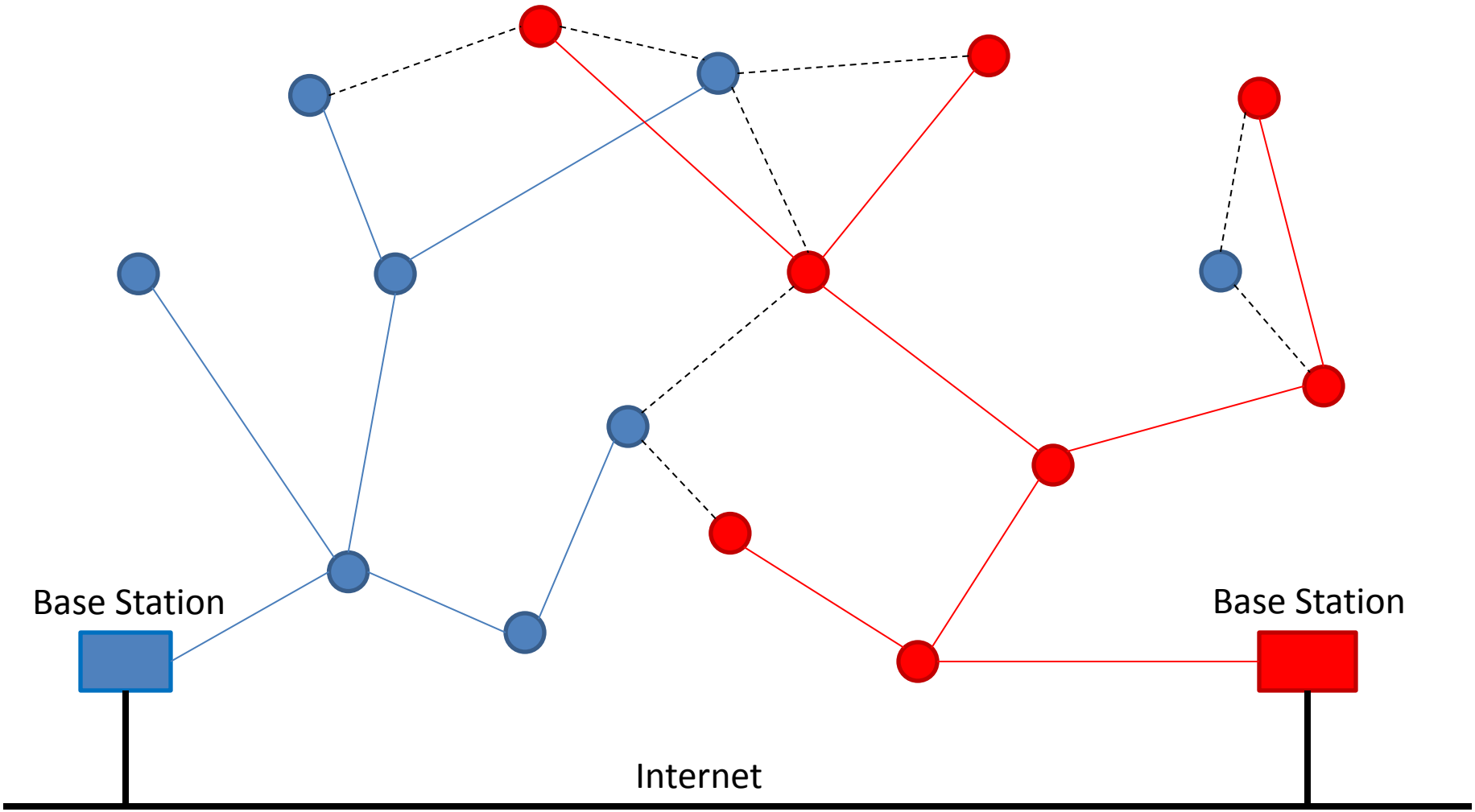
Peter Chapin and Christian Skalka
University of Vermont

MASS-2010; San Francisco; November 8-12, 2010

Outline

- What Problem are we Solving?
- SpartanRPC
- Language Design and Implementation
- Performance Evaluation
- Future Work

What Problem are we Solving?



Example: Emergency Responders

- Fire fighter's sensor network
  - Senses fire conditions
- Medical sensor network
  - Tracks state of injured individuals*
- Police with hand held devices
  - Communicates with other networks to alert officers about fire conditions

* Gao et. al., "Wireless medical sensor networks in emergency response: Implementation and pilot results" in 2008 IEEE Conference on Technologies for Homeland Security

SpartanRPC

- Extension of the nesC programming language
- *Link level* RPC discipline
  - Hides details of radio communication
  - Extends nesC “wiring” over the air
  - Allows dynamic reconfiguration of internode wiring
- Provides access control feature
  - Hides authorization computations
  - Allows multiple roles in the same network
- Intended to support higher level protocols
  - A light weight infrastructure

SpartanRPC Example

Interface

```
interface LEDControl {
  duty void setLeds(uint8_t ctl);
}
```

Client

```
module LoggerC {
  uses interface LEDControl;
}
implementation {
  …
  post LEDControl.setLeds(42);
  …
}
```

Server

```
module LEDControllerC {
  provides remote
    interface LEDControl
      requires "LEDMASTER";
}
implementation {
  …
  duty void LEDControl.setLeds
    (uint8_t ctl) {
     …
   }
   …
}
```

Dynamic Wires

```
configuration AppC { }
implementation {
  components ClientC, RemoteSelectorC;
  …
  auth "LEDMASTER"
    ClientC.LEDControl -> [RemoteSelectorC].LEDControl;
  …
}
```

SpartanRPC Addressing

(N, C, I)

- `N` : TinyOS node ID (0xFFFF for broadcast)
- `C` : Component ID
- `I` : Interface ID

(`C,I`) must be agreed upon; “well known” service ID

Sprocket

- Rewrites SpartanRPC programs to plain nesC
- Roles associated with symmetric AES keys
  - (role, key) associations known statically
- Authorization via simple MAC
  - 32 bit CBC-MAC computed with AES encryption
- Alternate authorization algorithms possible
  - Authorization independent of SpartanRPC features

Performance Evaluation (Memory)

	ROM Bytes	ROM %	RAM Bytes	RAM %
Baseline Client	13096	-	378	-
Baseline Server	12576	-	306	-
Duties Client	13568	3.6	398	5.3
Duties Server	12624	0.4	308	0.6
Security Client	22662	73	608	61
Security Server	21978	75	534	74

Tmote Sky nodes with TI's MSP430F1611 controller
48 KiB ROM, 10 KiB RAM

Performance Evaluation (Energy)

	Compute Burst (ms)	Compute Burst (µJoule)
Baseline Client	3	17
Duties Client	3	17
Security Client	4	22

Transmitter pulse lasted 15 to 16 ms => 780 µJ
MSP430F1611 draws max 500 µA => 6µJ in security case

Future Work

- Confidentiality support
- Access control using trust management concepts
- General staged programming in embedded systems

Peter Chapin (pchapin@cems.uvm.edu)
Christian Skalka (skalka@cems.uvm.edu)

Sprocket home page: http://www.cs.uvm.edu/~pchapin/Sprocket

Overflow Slides

Component Managers

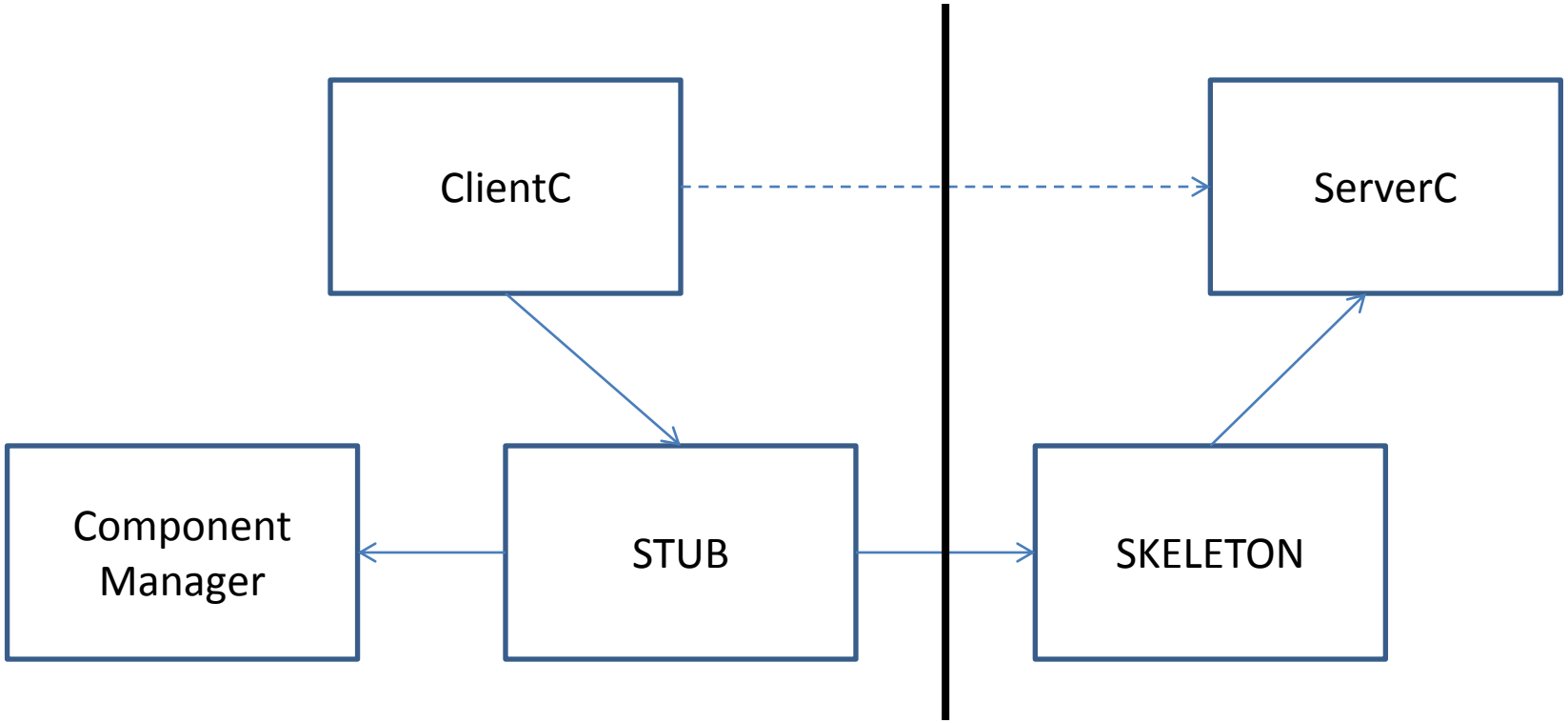
```
typedef struct {
  uint16_t node_id;   // N
  uint8_t  local_id;  // C
} component_id;

typedef struct {
  int count;
  component_id *ids;
} component_set;

interface ComponentManager {
  command component_set elements();
}
```

Components used to specify dynamic wire endpoints provide ComponentManager

RPC calls are multicast

Sprocket

Transforms SpartanRPC programs into ordinary nesC programs



Stub generated for each dynamic wire. Skeleton for each remote interface

Dynamic Wire Conversion

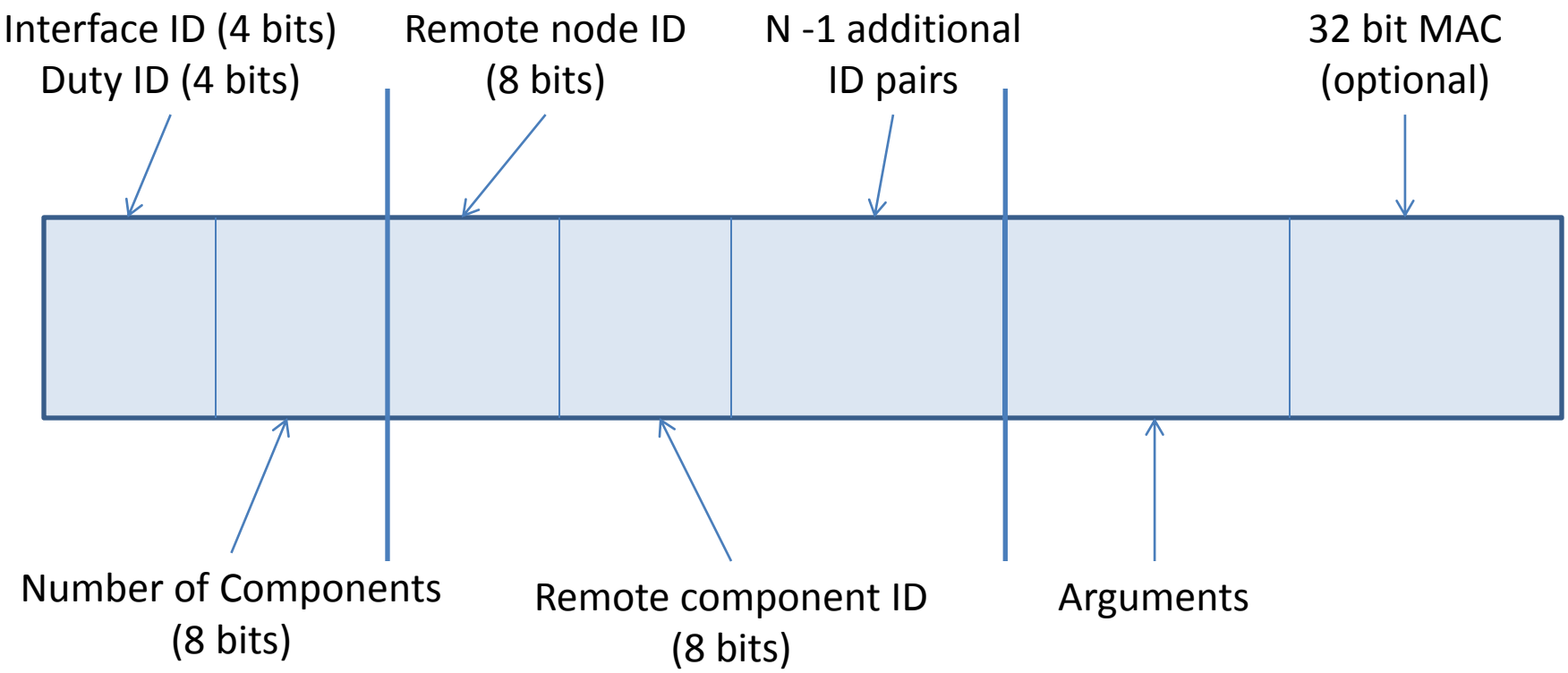
```
ClientC.LEDControl -> [RemoteSelectorC].LEDControl
```

The above dynamic wire is converted by Sprocket into the configuration

```
components Spkt__1;
ClientC.LEDControl -> Spkt__1;
Spkt__1.ComponentManager -> RemoteSelectorC;
Spkt__1.Packet    -> AMSenderC;
Spkt__1.AMPacket  -> AMSenderC;
Spkt__1.AMControl -> ActiveMessageC;
Spkt__1.AMSend    -> AMSenderC;
```

Where Spkt__1 is the stub that prepares the data packet and marshals arguments. Duty post operations in ClientC are converted into command invocations.

Message Format



Overhead = 2N + 2 bytes (or 2N + 6 when MAC is used)

Directed Diffusion Example

```
interface InterestManagement {

    duty void set_interest(
        uint16_t sender_node,
        int      temp_threshold,
        int      interval,
        int      duration);
}
```

```
interface DataManagement {

    duty void set_data(
        int      sender_node,
        int      originator_node,
        int      temp_value);

}
```