nprintf() Goals

- Lightweight visibility into network state.

- Easy-to-use tool:

  ```c
  nprintf(“Node %d changed parent.\n”, TOS_LOCAL_ADDR);
  ```

- Can be wired to radio, serial, and/or flash
nprintf() Requirements

- Easy-to-use - familiar API
- Lightweight - nprintf() will never run alone
- Reliability - no lost messages
- In-order delivery
- Synchronized - correlate distributed events
- Latency is not critical
Drawbacks of current approaches

- Simulation (TOSSIM) - not real
- Breakpoints (GDB/JTAG) - doesn’t scale
- Replay debugging (TOSHILT) - not lightweight
- Interactive (Nucleus, Marionette) - user-driven
Easy-to-use

- include <nprintf.h>

- components NPrintfM;

  ...

  NPrintfM.Output -> RadioOutC;

- nprintf(char *fmt, ...);
Lightweight

NPrintf:
1906 bytes in ROM
468 bytes in RAM

Serial:
1890 bytes in ROM
142 bytes in RAM
Reliability

- End-to-end acknowledgments
- Link-layer retransmissions
- Failures (i.e. queue overflow) indicated to user/application
In-order Delivery/Synchronized

- Packet sequence numbers/timestamps
- Lamport clocks (happens before)
- Reordering at PC-side
Evaluation

- Message generation rate vs. delivery rate
- Impact on application performance
- Latency for each output interface
- Response to topology changes
- Resilience to node failure
- Queue sizes
- Compare memory usage to Nucleus/Marionette
Design Choices

- Routing - Collection/Dissemination or Point-to-point?
- Transport - Is Custody Transfer better to achieve guarantees?
- Dataflow - Packet-switched or locked Virtual Circuits?
- Queue sizes - What are the call rates?
Drawbacks of Approach

- User Experience
  - Requires too much knowledge of app code?
  - Iterative approach too time-consuming?

- Limited Scope
  - Enough information gathered to diagnose faults?
Future Directions

- Design for large storage - query-based access to data
- System logger component - collects system status periodically