

Speaker Notes: Mobility Management in Sensor Networks (DCOSS 2006)

Muneeb Ali
muneeb@sics.se

June 15, 2006

This is an informal document meant to compliment the slides of the “Mobility Management in Sensor Networks” talk.

Slide 2: Outline

The important point to make here is that there are two recent research trends that motivate our work:

1. Towards a Sensor Network Architecture
2. Mobility in Sensor Networks

So first we would talk about the recent advances in moving towards a sensor network architecture. Briefly talk about the differences between the Internet and sensor networks and how it leads to different standards (IP for the Internet and SP for the sensor networks). Then we would talk about the Sensor-Protocol (SP) as our mobility-management framework is build on top of SP.

We summarize the recent advances in mobility for sensor networks (hardware that enable mobility, applications that require mobile sensors etc.). And then we present our Mobility Management framework.

Slide 3: Internet vs Sensor-Nets

It is safe to assume that the workshop audience consists of people who have been working in sensor networks for quite some years so they *should* be familiar with these differences. My suggestion would be to go over this slide quickly because explaining each and every point would take a lot of time and could be boring for the audience.

To be on the safe side here are the differences:

1. The Internet end host nodes are independent from each other (no way to predict which node requests what information) but sensor nodes have specific tasks and they usually collaborate with (physically co-located) nodes to perform those tasks.
2. A similar thing to the first point is that in the Internet there is end-to-end flow of communication (information from one host is sent to another and the internet core does not change that information) whereas in sensor networks the data might start as huge raw data but there is “in network processing” to reduce the data size and generally only aggregated information reaches the sink node. Similarly there are many-to-one flows and no two nodes within the sensor network would generally ever need to talk to each other - they normally just send data to the sink.
3. two-tier (fixed) architecture of the internet (i.e. we have routers etc) but in sensor networks the devices are more homogeneous (the sensor nodes act both as routers and as nodes)
4. Internet is generally wired, sensor networks are generally wireless
5. In the Internet it is important that how fast the information reaches the other node (latency) but in sensor nodes we do not care about latency but in fact we trade-off latency by putting nodes to sleep so that we can save battery.
6. In the Internet throughput is important - we want to use the maximum available link capacity. In sensor networks the data rates are generally very low and we care want to keep the sensors sleeping for as long as possible (so data rates turn out to be low and available link capacity is not used but we don't care about that)
7. bandwidth is cheap in the Internet but expensive in sensor networks (in the sense that the radio links do not give us a lot of bandwidth in the sensor world)

Slide 4: Internet vs Sensor-Nets

This slide talks about the fact that there if we compare the literature/experience of the Internet research community and the sensor-net research community:

1. What they (Internet researchers) have learnt and we (sensor-net community) have forgotten is BIG
2. What we have learnt and they have ignored is SMALL

There is this trend of sensor-net research of “making the area look more different from Internet research than it actually is!”. Ofcourse, Internet solutions do not directly apply to sensor networks BUT we can learn from their experiences and apply, change, and adapt their solutions to the peculiarities of sensor networks.

Slide 5: Towards a Sensor Network Architecture

Continuing from the last slide in the starting years of sensor networks research the “conceptual” sensor-net network stack looked something like the figure. It is a simplified form of the OSI model with some things like power management, mobility management cutting across different layers and presented as cross-layer entities. This traditional view is taken from the (famous) Ian Akyildiz (Georgia Tech) award winning sensor network survey paper (2002) [1].

However, even this simplified OSI model like network stack was never really strickly enforced in sensor-networks and researchers generally provided and implemented cross-layer solutions that cut-across the traditional boundaries. e.g. the MAC layer is supposed to decide when the radio is ON or OFF but in sensor-networks every protocol tries to play with the radio! (to save energy) e.g. TinyDB [2] is an application-layer program but it directly calls and manipulates the radio (MAC layer).

Basically this network stack was never enforced and we have monolithic solutions from different research groups that cut-across these boundaries blurring the traditional lines of what procotol lies within what layer and which layer provides what service to the one on top.

Irrelevant information: [TinyDB was Sam Madden’s PhD thesis at UC Berkeley. Sam is at MIT now. TinyDB basically implements a small database system on top of TinyOS and the main focus is how to provide in-network processing and use data aggregation techniques to collect data from sensor nodes]

Slide 6: Towards a Sensor Network Architecture

Currently, in sensor networks research you can change anything and everything. There are no standards that the protocols and solutions need to conform to - which is great for research but bad for interoperability and greatly reduces synergy between different research efforts.

Widely differing assumptions about the rest of the system: e.g. The PSFQ transport protocol [?] assumes that the sensor nodes could overhear each other. Now there are a LOT of MAC protocols (basically all TDMA-based MACs and any other MACs that put the radio to sleep) for which this assumption is NOT valid! Another example the (famous) LEACH [4] assumes a

TDMA-style MAC. If you try to run LEACH over a CDMA-style MAC it goes out of synch!

Vertically integrated designs: meaning that most sensor network research groups propose solutions which are vertically integrated with their own solutions. These solutions seem to work with their own set of components but cannot inter-operate with works of other groups. e.g. TinyDB assumes TinyOS as the underlying component which assumes B-MAC as the MAC protocol. I want to run TinyDB on the Contiki OS with TMAC. How do i do that? Its non-trivial. If you look at the codes there is little distinction that this is the MAC code or this is the application code. The overall “monolithic” system works but what use is it for other people?

No standards: Not having standards might be good for research (gives researchers more space to experiment) but bad for interoperability. Not having a sensor-net standard is considered as the SINGLE-MOST important road block by some researchers [David Culler and Joe Polastre (UC Berkeley) et. al HotOS paper [5]]

Irrelevant information: [PSFQ is a work by Andrew T. Campbell’s group from Columbia published at SenSys’03. Andrew recently moved to Dartmouth. PSFQ is basically a link-layer protocol aimed at improving congestion for sensor networks using different techniques like Pump-Slowly-Fetch-Quickly. LEACH is the PhD thesis of Wendi Heinzelman at MIT. Wendi joined Rochester. LEACH is a cluster protocol. Talks about how clusters should be formed. How cluster heads should be elected and more importantly how the responsibility of being a cluster head is rotated amongst nodes. Contiki OS is the operating system for sensor networks written by Adam Dunkels at SICS, Sweden. TMAC the famous contention-based traffic-adaptive MAC protocol by Koen Langendoen at TU Delft, The netherlands.]

Slide 7: Sensor-Net (SP) Protocol

(As a response to the problems mentioned in the last slide) SP [?] (Joe Polastre’s PhD thesis at UC Berkeley) is one of the early encouraging steps towards defining a sensor-net architecture. Basically SP is to the sensor-networks what IP is to the Internet. IP cuts the Internet network stack into two; layers above IP and layers below IP. The applications and protocols written on top of IP and the link-layer and physical technologies below IP can evolve independent of each other and still continue to operate with each other.

However, unlike IP the “waist-line” is NOT at the network layer but instead SP sits between the network and the data link layer. Why do we lower the “waist-line” in sensor-networks? It is because processing potentially occurs at each hops and not just at the end points i.e. in the Internet the data part of a packet is not changed by the forwarding nodes but in sensor-networks because

of data aggregation and in-network processing even the data-part of a packet could potentially change at each hop. So the standardization point becomes the one-hop communication point and NOT the end-to-end communication like the Internet and IP.

Basically, SP would allow a network protocol, say A, to run over different MAC layers without any change. Similarly it would allow this network protocol A to run over different physical technologies e.g. CC1000 radio or IEEE 802.15.4 packet radio.

As SP is an abstraction it could be implemented in any operating system for sensor networks. So far, SP is not even publicly available for TinyOS (there is an unstable version of SP available from joe polastre though) and there is work going on at SICS, Sweden to implement SP on the Contiki operating system. SP basically performs three main operations. A) Send data B) Receive data C) does neighbor management - so that different network protocols do not have to do their own neighbor management they can just make use of the SP neighbor table.

Main differences of SP from IP would be:

1. unlike IP, SP allows some sort of “feedback” mechanism which could be used e.g. to send congestion bits up the network stack to indicate that there is congestion
2. another flexible behavior different from IP is that different protocols could request urgent or reliable service i.e. if some sensor data is more important they can request SP to send their packets first (SP could choose to ignore these bits) OR as there is little concept of reliable data delivery in sensor-nets if some application wants reliable data-delivery was sent they can request so.
3. as SP maintains a neighbor table it can allow different protocols (network protocols or MAC protocols) to use this link-layer information e.g. which nodes are my first-hop neighbors, which nodes are my 2-hop neighbors, what is the link-quality to some neighbor say B etc. etc.

Slide 8: Sensor-Net (SP) Protocol

This slide presents the diagram representation of SP. The main points to explain in the figure would be:

1. SP is NOT at the network layer and SP is NOT at the data-link layer it sits BETWEEN the two layers.
2. SP cuts the sensor-net network into two.

3. Below the SP “narrow waist” are the data link-layer technologies and the physical layer technologies.
4. the most important thing at the link-layer is ofcourse the medium access control protocols. However, this layer also performs other functions like time stamping, acknowledgements etc.
5. At the physical layer we have physical sensors (sensing), carrier sense, and physical transmit and receive (apart from other things).
6. Now if we go ABOVE the SP layer, there are basically two layers above (A) network layer (B) Application layer.
7. The network layer could have “Address free protocols”, “Name-based protocols” which could be used by other network layer protocols like in-network storage.
8. The important thing to notice about applications is that the applications at the application layer could talk to SP directly, or via some network layer protocol or via some network layer protocol that builds on top of some other network layer protocol.
9. finally there are some things in sensor networks that cannot fit into any specific layer - these are represented as “cross-layer” services in the SP-architecture e.g. power management has to deal with the physical layer, it also has to deal with the MAC layer (to control radio power), it also needs to deal with network protocols (to optimize network protocols for power).

Our proposal is that just as SP maintains a “network table” to save different protocols from doing neighbor mangement themselves and this helps taking one step towards defining a sensor network architecture. For mobile sensor networks we need to implement mobility management as a cross-layer service in SP so that different applications or network protocols or MAC protocols don’t have to do mobility management themselves! (more on mobility management after a few slides).

Slide 9: Sensor-Net (SP) Protocol

This slide is another diagram representation of SP. The important point to make in this diagram would be:

1. SP simply maintains a neighbor table and a message pool
2. different network protocols (network protocol 1 or network protocol 2) could use the SEND or RECEIVE interface to send or receive messages.

3. different network protocols can use the SP neighbor table to get information about neighbors.
4. by using “SP Adapters” SP enables different link-layer technologies to co-exist e.g. assume data Link A is SMAC (the famous SMAC Infocom’02 by Deborah Estrin’s group) running on CC1000 radio and data link B is TMAC (traffic-adaptive MAC SenSys’03 by Koen Langendoen TU Delft, The Netherlands) running on a IEEE 802.15.4 packet radio like CC2420. By writing a simple “SP Adaptor” for each data-link (A and B) the two different MAC protocols (SMAC and TMAC) can run with same higher level network protocols. Similarly the higher level network protocols can run on different physical radios like CC1000 or 802.15.4 standard packet radios.

Slide 10: SP vs ZigBee

This slide is a NOTE that SP is NOT the only proposed sensor network architecture there are other proposals as well e.g. ZigBee. Defining a sensor-network architecture would be an organic evolving process that would take some time. However, SP is an encouraging step towards the right direction.

Whats the difference between ZigBee and SP? In ZigBee each layer assumes a specific instance of the layer below. ZigBee stack is divided into a simple Application layer, Network Layer, Data-Link Layer, and Physical Layer categories but each layer assumes very specific instances of the layer below e.g. the link layers and physical layers in ZigBee are ALWAYS IEEE 802.15.4 standard compliant and the routing layers are build on the assumption that the underlying layers are IEEE 802.15.4 compliant. There is a direct quote of joe’s SP SenSys’05 paper on the slide where he makes very aggressive comments about ZigBee “An architecture build on static technologies is destined for obsolescence”! So would SP prevail as the architecture or ZigBee or some other standard? We do not know. Only time would tell - however, what we do know is that we need SOME standard architecture ANY architecture!

Slide 11: Mobility in Sensor Networks

This brings us to the second research trend that motivates our work - mobility in sensor networks. So on this slide we give a brief overview of the recent trends in mobility in sensor networks.

Research community in sensor-nets has largely “ignored” mobility so far. Almost always protocols and sub-systems assume that the sensor nodes are static. However, recent works like RoboMote [?] and Parasitic Mobility [?] **have enabled mobile sensor networks.** (Note: RoboMote the work from USC is

a pure hardware advancement. Its a Mote that has hardware abilities to move around. Parasitic mobility the work from MIT media lab is a hardware plus software thing - basically they use the concept of parasites and say that sensor nodes could selectively attach or detach themselves from mobile objects and do simulations on how a mobile node would improve on network coverage, node discovery, uniform node deployments, context aware deployments etc. They build some hardware prototypes for their idea of parasitic mobility and do some real implementation and testing as well)

Recently there has been an increased interest in the area of using sensor networks for **medical care** (also sometimes called as “body sensor networks”) or for using sensor networks in **disaster relief**. Such new applications challenge the assumption of static sensor nodes as in these environments the sensor nodes could be mobile e.g. sensors attached to doctors or patients or sensors attached to disaster first-responders who come to rescue people.

The protocols designed for static sensor networks perform poorly in mobile scenarios e.g. the study done in the MMAC IPCCC’05 paper shows that TDMA-like scheduling MAC protocols can go loose correctness in mobile scenarios because each node is supposed to sleep and wakeup at a very specific time and only one node is supposed to transmit during a specific frame - a mobile node not aware of the schedule of some 2-hop neighborhood could disrupt communication leading to incorrect behavior of the TDMA-like protocol. Further, for contention-based MAC protocols the collision rate goes up with increase in mobility and their performance also degrades. So there is a need to have mobility-adaptive protocols like the mobility adaptive MAC - MMAC.

Further, mobility is not always a “bad” thing. Its true that mobility makes normal operations of network or MAC protocols more complex but mobility could also be applied in a useful manner to improve certain things like e.g. localization accuracy or network coverage. (references are mentioned on the slide)

Slide 12: Mobility in Sensor Networks

This slide is just a picture of RoboMotes (the work from USC) and it shows the audience an example of Hardware mobility. The picture shows a group of RoboMotes that have movement hardware (wheels) attached to them and these motes can move around in the environment.

Slide 13: Mobility in Sensor Networks

This slide is a picture giving an overview of medical care or disaster relief applications of sensor networks (these environments make use of mobile sensor

nodes). The image is courtesy of CodeBlue project at Harvard (lead by Matt Welsh who finished his PhD at UC Berkeley with David Culler and is at Harvard now) This picture shows that motes could be attached to patients monitoring vital signs of the patients like heart rate etc. PDAs carried by doctors could see these vital signs. Also incase of a disaster like fire etc. the first-responders can make use of sensors attached to for various purposes e.g. sensors attached to firefighters could help the firefighting team to locate where other firefighters are and this could help bringing firefighters back alive as the team could know if some firefighter is stuck somewhere in the building.

The important point to make here would be that sensors attached to patients or carried by doctors or attached to firefighters would all be mobile as the individuals are mobile themselves.

Slide 14: Mobility-Management

Now lets discuss the mobility-management problem in the light of the two previously discussed trends (towards a sensor network architecture and mobility management).

1. We need a sensor network architecture!
2. Sensor nodes are no longer static! so we need mobility-management!
3. we need mobility-management in a way that complies with and helps us move further in defining a sensor network architecture!

First of all, mobility information could be required by applications and protocols running at DIFFERENT conceptual layers e.g. if there is an application that monitors the physical movement of depression patients then that application-layer entity needs mobility information. if there is a mobility-adaptive network protocol then it needs mobility information for neighbor discovery or for route maintenance. a mobility adaptive MAC protocol e.g. MMAC would require mobility information so that it can dynamically adjust its frame size inversely proportional to mobility. Basically, applications and protocols at ALL layers could potentially need to gather, store and manage mobility information.

Now, either these applications and protocols could individually gather, store and manage mobility information (which is the current practice) OR they could make use of a mobility-management SERVICE that “takes-care” of all mobility information needs of these applications and protocols. (which is our proposal). It is easy to see that in the current practice of every application or protocol gathering, storing, and managing mobility information there is a lot of redundant processes involved. This could lead to redundant network communication (bad for energy-efficiency), more data storage (the nodes do not have that much

space RAM is a limiting factor) and possibly conflicting views on mobility information by different protocols and sub-systems (MAC layer thinks there is a lot of mobility in the network but the network layer does not think so possibly because MAC layer gets updated more frequently about mobility information than the network layer protocol).

So it is better if we have one mobility management framework to provide a unified view on mobility and (more importantly) to decrease on redundant data communication (every protocol does not need to fetch information about mobility itself - they could all share the same mobility information). Further, we can make this single mobility-storage database visible across all layers (by implementing it as a shared buffer) so that applications and protocols at each layer can make use of the mobility information.

So we FETCH mobility information FROM the network stack and store it in the cross-layer database and any application or protocol that needs mobility information could get it from the cross-layer mobility-management database. As this database is updated frequently all applications and protocols can have a unified updated view of mobility.

Standardizing what goes INTO the database enables the network protocols and management applications to evolve independent of each other. Lets take an example from the Internet. Most of us would be familiar with HTTP LOG files e.g. ERROR-LOG generated by HTTP. This is one standard way how HTTP writes the ERROR-LOG and it is a very simple way and it generating a simple text file of ERRORS. Now depending on the needs of the “customer” of the log file someone could do a simple thing like search for a specific thing in the log file manually OR you can build sophisticated web site statistics applications using the same log file. As long as there is a **“standard way of writing the log file”** the applications that use the information from the file can evolve independent of the applications which WRITE those files. So back in 1990s there was little concept of sophisticated web statistics like Google Query Stats. The same HTTP protocol is still used today to build these sophisticated applications. Learning from that Internet experience, there is a need to de-couple the applications and protocols which PROVIDE information from the applications and protocols which USE that information. So that these protocols and applications can evolve independent of each other over time. One simple way to de-couple them in our case is to define what goes INTO the database. Defining a standard on what goes INTO the mobility management database would help us take another step towards defining a sensor-network architecture. Also, once we have defined what is contained in the database - this would serve as a common interface for two set of protocols and applications - the **“customers of mobility information”** and the **“sellers of mobility information”**. Customers and Sellers of mobility information would be able to *talk to each other* using the **“common language”** - our standard mobility-management database.

Slide 15: Mobility-Management

This slide is a diagram representation of the things we talked about in the last slide. This is the "Bow-Tie" mobility management architecture proposed by us. The important points to make here would be:

1. On the left side of the figure we have the sensor-net network stack as defined by SP (and talked about earlier on).
2. SP sits between the network layer and data-link layer and defines the "standardization point".
3. Mobility management is implemented as a cross-layer service (left side of the figure) in the sensor-net network stack.
4. On the right side of the figure are the management applications (the "customers of mobility information").
5. its important to note that (elements on the right side) - the customers are from DIFFERENT network layers e.g. patient monitoring is from application layer, power management is cross-layer, neighbor management is at SP (between network and data-link layer), MMAC is at MAC layer.
6. in the center of the figure we have the mobility-management database which is implemented as a SHARED-BUFFER visible across all layers.
7. the policy box defines the policy of updating the database i.e. system developers can define here how frequently to update the database, which elements to update when etc etc. (basically any policy related matter goes here).
8. so we FETCH the mobility information from the LEFT SIDE of the figure (i.e. the network stack). We can fetch different parts of mobility information from different layers e.g. if we need to store the mobility information of neighbors of a node. we would need the node's ID (from the MAC layer), we would need information about neighbors (from SP), maybe we would require information about congestion experienced by the nodes (the congestion control protocol at the network layer would know about that). Basically information from different layers could be stored in the mobility management database.
9. what is important is that we need to DEFINE what goes INTO the database, so that the database becomes a "common language" that the sellers of mobility information (left side) and "customers of mobility information" (right side - management applications) could use to talk to each other. (more on defining what goes into the database later on - its an open question)

10. Now the original MMAC (mobility-adaptive MAC proposal) (right side) includes specification on how to obtain mobility information, how to store it and how to use it. With our mobility management framework MMAC does not need to do mobility management itself - it could simply use the mobility-management cross-layer service to satisfy all its mobility information needs. Further, the mobility information that the mobility management cross layer service would fetch for MMAC could also be used by some other component e.g. neighbor management - and we do NOT need to fetch the same information again.
11. the difference between the mobility management cross-layer service (left side of the figure) and the mobility management database (center of the figure) is simple - the database actually stores the information as a shared buffer and the cross-layer mobility management service provides a cross-layer interface to both sellers and customers of mobility-information i.e. anyone who has got anything to do with mobility talks to the mobility-management cross-layer service and the mobility-management service talks to the cross-layer database. the mobility management cross layer service is just an interface to the database they are essentially the same thing.

Slide 16: Mobility-Management

Our mobility management framework does NOT take any stance on naming - it could work with any underlying naming scheme however it assumes that UNIQUE names are available. Our framework also does NOT take any stance on time-synchronization and could work with any time-synchronization mechanism.

point two in the slide is extensively covered in the last slide.

for mobility estimation we propose to use the AR-1 model. AR-1 model is the autoregressive-model these models are used to estimate almost anything and Zainab Zaidi et al., used AR-1 model to estimate mobility in wireless networks. As in, if we know the (X,Y) of a node at a given point say t , and we know the (X,Y) of a node at $t+1$, and so on till lets say $t+9$ then we can ESTIMATE or predict the (X,Y) of the node at time $t + 10$ (when we are actually at time $t + 9$).

we do NOT propose to use more complex estimation models e.g. AR-3 (AR-3 takes in more input parameters than AR-1 and is more accurate) because we believe that using more complex models like AR-3 would be too computationally intensive for the small sensor nodes.

when using estimation models like AR-1 - the accuracy of the mobility es-

timization depends on the underlying localization accuracy. Localization is too far off the focus of this talk - so the line just serves as a warning that yes! the accuracy of the underlying localization mechanism effects the accuracy of mobility estimation obtained by the AR-1 method.

Now it is easy for a node to predict its own mobility using AR-1, but for the mobility information to be useful a node needs to know about the predicted mobility of OTHER nodes as well so obviously there is some communication overhead to tell all nodes of mobility of their neighbor nodes (MMAC does that by doing a broadcast from the head node at the end of a schedule-based frame like all nodes send their mobility information to the head and then head just sends it to all members of the cluster). Anyway, so there is SOME communication costs of gathering mobility information. Are the benefits of that worth the cost? The MMAC study says yes they are - atleast for the MAC layer. But more real experimentation is needed to answer this question (and that is a part of our on-going work).

Slide 17: On-going Work

There is work going at SICS, Sweden on implementing SP on the Contiki operating system. (Its a master student project). In parallel to that there is work going on to implement the mobility-management framework in contiki but as the framework requires SP - we need the SP implementation before we can proceed further. (As discussed earlier there are no publicly available implementations of SP yet - the SP protocol is fairly new and joe's implementation of SP on TinyOS is not exactly stable). Why we use Contiki and not TinyOS? Simple answer: Contiki was build by researchers at SICS and we use it for everything. Longer answer: TinyOS has NesC which is a variant of C while Contiki can be programmed in normal C so there is no need to learn the extra semantics of NesC. Learning NesC has been a complain of many TinyOS developers. Further, TinyOS might be the most famous OS for sensor networks but it is not necessarily the best one - there have been lots of complains about using tinyOS and currently they are completely writing the TinyOS from scratch and changing a lot of things in it. The new release of TinyOS called T2 or TinyOS 2.0 is not available yet.

Also, we have libraries like Protothreads on Contiki. Protothreads are stack-less threads. They could be thought of as something in-between threads and event-driven programming. They have benefits of both and draw-backs of none. Using protothreads help in writing smaller code and makes the program logic more visible as the programmer can code in a thread-like fashion.

For simulations, we are using the COOJA simulator. COOJA is the sim-

ulator for Contiki and for doing simulations in COOJA we write C code for contiki - so it makes it very easy to map the code written for simulations to real deployments using Contiki. Infact, in most cases the same simulation code can be used without any modifications at all!

It is important to note that for any mobility evaluations it is important to consider realistic mobility models e.g. CodeBlue (healthcare architecture work from harvard mentioned before) just considers the mobility of a doctor in the hallway for their mobility evaluations (such simple mobility model might not be enough for evaluations!). There have been studies on realistic mobility models and we plan to implement some of these models for our mobility evaluation (reference on slide).

Also for simulations it might be better to use REAL mobility traces collected by some research efforts (reference on slide). However, it is important to note that these mobility traces were recorded for mobile ad-hoc networks and might not be directly applicable to sensor networks. In the best of our knowledge no mobility traces have yet been recorded for sensor networks.

Slide 18: Open Issues: Standard Database?

By now we have hopefully convinced the audience that moving towards a sensor network architecture is a good thing and that our proposed mobility management framework is a nice way to handle mobility in the over-all sensor network architecture. But we talked about standardizing what goes INTO the database and tried to convince that it is IMPORTANT to standardize what goes into the database so that we can have a "common language" that different (possibly future) protocols can speak.

The standard database is an open issue - we do not attempt at proposing a standard for the database at this point. Maybe in this workshop we can have a healthy discussion on what possible things could go into such a cross-layer database. Let's look at one example: assuming we are using the AR-1 model for mobility prediction an example database could look something like the figure in the slide:

1. we can store node ID in the database to uniquely identify a node (thats why we have the requirement that the naming scheme should provide unique names)
2. we can store predicted (X,Y) of the node.
3. this prediction was made at lets say time T1 (we need to store that information so that we know when the information is no longer useful)

4. also we need to store that for WHAT time the prediction was made lets say it was made at time T1 FOR time (T1 + i)

This is just one very simple proposal for the standard database. This proposal is tied to the AR-1 model. This is an open issue and we are open to suggestions.

Slide 19: Open Issues: IP over SP?

A second open issue not directly related to mobility but related to moving towards a sensor network architecture is that. IP has been discarded for sensor networks due to various reasons (e.g. processing occurs at each hop and not just at the end points). However, the Contiki operating system is build around the micro-TCP/IP stack. Micro-TCP/IP is a small implementation of TCP/IP for 8-bit architectures written by Adam Dunkels at SICS, Sweden and published at MobiSys 2003.

Now unlike tinyOS contiki has a working FULL TCP/IP stack for 8-bit architectures. So can IP and SP co-exist? Do we even need SP if we have IP? The figure shows one way in which IP could be a part of the SP-sensor network architecture. In this figure we just treat IP as any other network layer protocol and if some application wants to use it - it can use it. But this IP vs SP is an open issue and we are open to suggestions.

Slide 20: Conclusions

Conclusions should be short i guess, so those are straight from the slides.

References

- [1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communications Magazine*, vol. 40, pp. 102–116, Aug. 2002.
- [2] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: an acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, 2005.
- [3] C.-Y. Wan, S. B. Eisenman, and A. T. Campbell, "Coda: congestion detection and avoidance in sensor networks," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 266–279, 2003.

- [4] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on Wireless Communications*, vol. 01, pp. 660–670, Oct. 2002.
- [5] D. Culler, P. Dutta, C. T. Ee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, and J. Zhao, "Towards a sensor network architecture: Lowering the waistline," in *Proc. HotOS-X*, June 2005.