NanoIP: The Zen of Embedded Networking

Zach Shelby
CWC, University of Oulu, Finland
e-mail: zach.shelby@ee.oulu.fi

P. Mähönen, J. Riihijärvi, O. Raivio
RWTH, Aachen University, Germany
e-mail: mahonen@comnets.rwth-aachen.de

Pertti Huuskonen
Nokia Research Center, Tampere, Finland
e-mail: pertti.huuskonen@nokia.com

***Abstract*— This paper reports on the development of an embedded networking protocol stack for pervasive embedded devices, taking into account requirements for sensors and low-price consumer devices. Instead of applying TCP/IP (designed to support routing and end-to-end connectivity) to embedded networking, our boundary conditions are quite different. The protocols presented here specifically provide embedded networking within local subnets. No routing is performed, instead a gateway can make the embedded devices visible to the global Internet. This configuration provides better scalability, RAM/ROM utilization, and lower power consumption. The developed single subnet architecture and additional protocols supporting pervasive applications are presented and shortly analyzed.**

I. Introduction

This paper introduces nanoIP, a minimal networking protocol for use with highly limited devices. Embedded networking applications often require only a network consisting of a single link or extended bridged subnet. It is argued that direct end-to-end compatibility with the Internet is not actually a desirable feature for embedded devices. In fact, direct connectivity may prove to be a huge scalability and security problem in the future with embedded Internet devices entering our lives. However, access to embedded devices, at least indirectly through a gateway, is often a necessity. In many cases the application itself will be peer-to-peer between proximity devices, e.g. between collaborative sensors, but there are cases where embedded devices need access to the Internet. This functionality can be provided through a gateway. The nanoIP protocol optimizes embedded networking over a subnet while moving complex functionality to Internet gateways. It is an embedded alternative to IP, and includes equivalent services such as unreliable datagrams (nanoUDP), connection oriented streams (nanoTCP), a socket compatible interface, and familiar application protocols. These features ease the transition to nanoIP, reduce gateway complexity, and speed up application development.

II. The Embedded Subnet

This research has studied and implemented solutions for resource-limited pervasive networking scenarios that could be suitable for many consumer devices. This includes cases where gateway functionality could be provided by more advanced equipment such as a mobile phone or PDA already equipped with TCP/IP. The most common constraints for extremely limited embedded devices are ROM and RAM size, implementation complexity, and power consumption. The goal of nanoIP is to provide a protocol suite optimized for these small devices and the networking environments they are used in.

Embedded short-range wireless networks provide some unique opportunities. First, embedded devices often only require networking over a single subnet (a subnet can consist of multiple bridged links). Secondly, if access to the external Internet is required one can use a gateway for indirect (and more secure) access. However, a gateway must not be a required node, as devices should be able to communicate peer-to-peer within a subnet or the range of their radio. Routing is not of interest in this research as it would add complexity and overhead to the protocol stack. Nodes acting as transparent bridges between links could be of more practical value. Figure 1 gives an example configuration of embedded subnets and an Internet gateway. In embedded short-range wireless networking most of the required communications is low bit-rate (1-30 kbps, easily implementable with small radios operating e.g. in the ISM bands [1], [2]) and in the case of some applications over 100 kbps, in which case Bluetooth or a similar technology could be attractive. The forthcoming IEEE 802.15.4 is another potential technology providing 20-250 kbps with a low power MAC [3].

NanoIP is applicable to sensor networks, "smart dust" scenarios [4], peer-to-peer pervasive networks, and more traditional home networking. The Mobile Mote project [5], which has had similar goals as this project, is an excellent start towards a highly scalable and minimized architecture especially in the sensor domain. Mobile Mote includes both link layer ARQ as well as a reliable transport layer. The architecture is dependent on access point nodes and works to solve mobility problems. NanoIP differs from Mote in being architecture and link independent, also applicable to peer-to-peer networking. Göthberg's microIP [6] work has influenced our direction towards embedded IP. In this work a small 8-bit address space and routing, along with protocols like ARP, ICMP, and DHCP are still included as part of the embedded IP solution. The simplified UDP and TCP protocols are similar to nanoUDP and nanoTCP along with the proxy solution, which is similar to our gateway. NanoIP provides an even more minimal protocol by removing the network layer altogether and providing embedded protocols at all layers. This is important as running standard HTTP over an embedded protocol stack will kill all gains made at lower layers. This research is in agreement that full TCP/IP is not needed in the smallest of embedded devices. In order to save complexity, RAM and ROM, and power consumption a proprietary optimized protocol is

0-7803-7802-4/03/$17.00 © 2003 IEEE

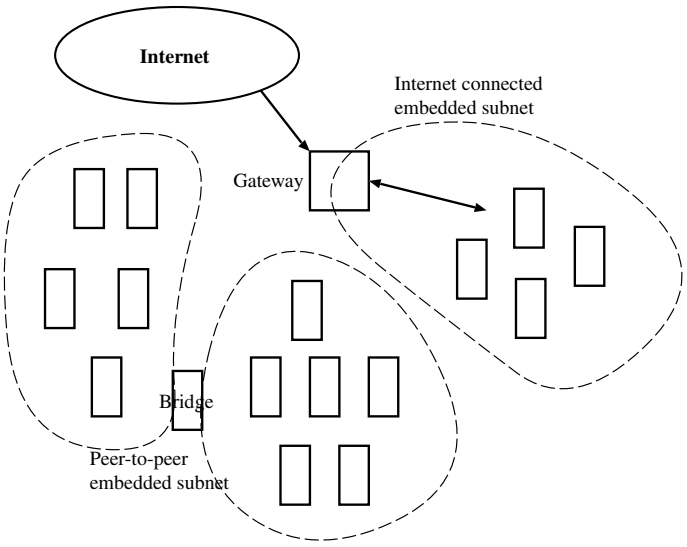

Fig. 1. Three embedded subnets, with one connected to the Internet through a gateway.

proposed, which is nevertheless interoperable with IP networks through a proxy mechanism. Our previous research to provide functionally minimized HTTP services on a chip has also influenced this work, especially by stressing the requirement of overall optimization and the importance of web-services [7].

A. TCP/IP for the embedded world?

TCP and IP were designed for use over fixed networks with a reasonable amount of bandwidth. They provide routing, flow control, and the interconnection of heterogeneous networks which have together become the Internet. In this context they are a great success. Along with UDP, practically all data exchanged on the Internet is carried over these protocols. However, in the context of wireless, low-power, embedded devices it has many unsuitable and unneeded features.

From the above criteria for embedded subnet networking, it can be seen that the features of the Internet Protocol layer are not needed. Routing is needed only when connecting multiple heterogeneous subnets together and this uses a large global address space. TCP's adaptive flow control becomes useful only over multiple routing hops where congestion can be a problem. IP address resolution is not needed without a separate global address space. These features result in extra complexity and wasted bandwidth.

IP version 4 has a variable-sized header with many options to be parsed, and even though IP version 6 no longer has ARP, it has optional headers and equivalent mechanisms. The address space of IPv6 is 128 bits, so each and every IPv6 packet carries 32 octets of address information. If the link MTU is less than 500 octets the overhead of IPv6 starts to be very large, see Figure 4. Most 433MHz transceivers, for example, recommend MTU sizes in the range of 25-50 bytes (in which case TCP/IP can not even be directly used). In addition, if one wants to have a node compatible with Internet

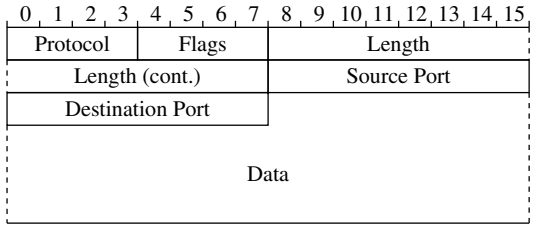

Fig. 2. Frame format of nanoUDP.

requirements, one must implement not only IP and TCP but also UDP, ICMP, and ARP as well.

The features, complexity, and overhead of the TCP/IP suite were not designed for, and are not usually suitable for low-power wireless embedded networking. There are however cases where higher power devices, the need for end-to-end Internet communication, or easy inter-operation with existing nodes make TCP/IP the best choice. This research introduces nanoIP as an alternative specifically suitable for embedded networking, which includes embedded application protocols.

B. The nanoUDP and nanoTCP protocols

The OSI network layer is responsible for the routing of datagrams between networks. As no routing is performed by nanoIP, it only implements transport layer functionality. There are no global IP addresses, instead the link-layer MAC address is re-used. This is possible because nanoIP is meant for use over a single subnet where MAC addresses must be unique. With this context, 48-bit IEEE MAC addresses are used much like an IP address. For example, with nanoIP a URL may look like

```
http://00:50:15:02:00:8F.80/temp
```

where `00:50:15:02:00:8F` is the 48-bit MAC address of the node's network interface and `80` is the port of the service. Broadcast is of course represented by `FF:FF:FF:FF:FF:FF`. Shorter MAC addresses can also be used and represented with this format.

The protocol is actually made up of two transport protocols, nanoUDP and nanoTCP. Figure 2 illustrates the frame structure of nanoUDP. The header begins with a protocol number and flag byte, followed by the payload length. To enable port multiplexing, 8-bit source and destination ports are included in the header. Using flags, data can be segmented into datagrams suitable for the MTU size of the link. The total length of the nanoUDP header is five octets. This connectionless transport is suitable for most embedded traffic where conversations are usually short-lived and retransmissions may be handled by applications. However, many situations demand a reliable transport protocol.

NanoTCP is a simple bit stream transport protocol with fixed flow control. It is terminated at the gateway, which is responsible for maintaining full TCP connections with Internet hosts. Figure 3 shows the nanoTCP frame structure. This offers a reliable connection-based transport. It is a very

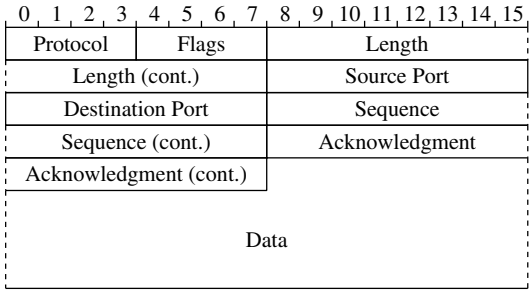

Fig. 3. Frame format of nanoTCP.

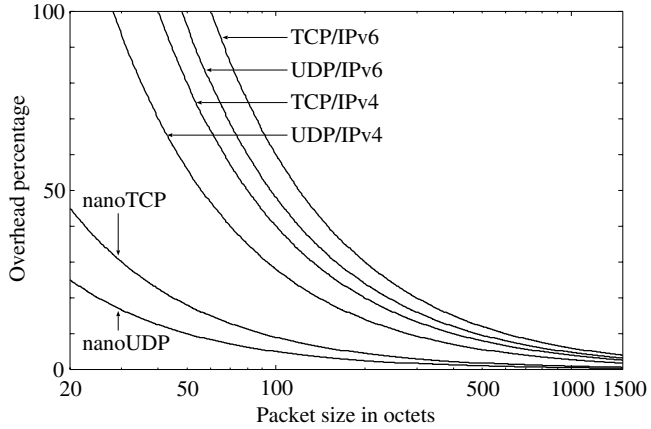

Fig. 4. Overhead comparison between different protocols against the typical MTU range.

simplified version of TCP. The first five octets are identical with nanoUDP. This is followed by 16-bit sequence and acknowledgement fields providing retransmission capabilities. The total length of the nanoTCP header is nine octets. Congestion control and out-of-band signaling are not included, as they are not needed across a single subnet. nanoTCP flow control works as in TCP except that the window size is fixed. The RTO is implementation specific, reference values can be set for different link technologies.

These proposed protocols provide a very efficient network and transport layer. Parsing is simple as the first five octets are identical for both transports. Segmentation and retransmission, which can be disabled for the tiniest of devices, consume the most memory and processing power. An overhead comparison between different protocols and packet sizes (see Figure 4) shows that the bandwidth savings are remarkable with nanoIP. One must also remember that because there is no ICMP or ARP, there are even fewer packets than with IP. This further reduces bandwidth usage, which is especially important in the case of wireless communications where the transceiver is often responsible for most of the power consumption.

III. Application Protocols

Pervasive peer-to-peer embedded computing involves not only network and transport protocols, but also service discovery, application protocols, and data formats. Service discovery is useful in the consumer embedded context to provide name resolution and distributed network services. For this reason this and other key protocols are incorporated into the nanoIP framework.

The nanoIP protocol does the job of providing a low overhead, low power, and simple transport with a standard socket interface. However, the gains of this would be lost if extremely inefficient protocols such as HTTP and SLPv2 combined with data formatted with HTML or XML were run over nanoIP. These protocols are complex and text based, which is both bandwidth inefficient and difficult to parse and implement on limited devices. For this reason, embedded versions of SLP, HTTP, and XML are used along with nanoIP. They are functionally equivalent to their IP counterparts which makes the Internet gateway simpler to implement and the protocols more familiar to developers. These are however only examples, as nanoIP uses a socket interface any lightweight protocol can be used.

A. NanoHTTP

Above the transport layer there is clearly a need for a common application-level protocol for accessing services. On the Internet this functionality is typically provided by the HyperText Transfer Protocol (HTTP) [8]. However, while HTTP can be used in the embedded world precisely as specified in the current standard, it is advantageous to place some restrictions on the form of the HTTP-messages in the name of efficiency and ease of parsing. Another important improvement would be to translate the text-based protocol into a binary compressed version.

One of the distinguishing characteristics of HTTP (compared to the lower-layer protocols discussed above) is that it is a text-based protocol, and in most cases a case-insensitive one. This, together with a non-fixed order and large number of headers makes parsing HTTP messages rather CPU and memory intensive, especially if one wants to implement HTTP support in hardware. Added difficulty is brought into the picture by the complex form of some field-values (for example, Accept-$*$) which allow different weighting-factors to be inserted, making the parsing of the message even more complex.

In the nanoIP stack a simplified version of HTTP is included, called nanoHTTP. The support for many HTTP headers is not necessary for subnet-level communications (such as those related to proxying), and the remaining headers are to be sent in predefined order. Support for complex field-values is also dropped. NanoHTTP goes even further to encode all headers using a binary version of HTTP. Using this binary form enables significant savings in terms of memory, processing power, and bandwidth.

B. nanoSLP

The nanoIP protocol does not make use of address resolution to resolve MAC addresses or domain name servers in order to resolve URLs. Since nanoIP addressing is based on MAC addresses, address resolution is not needed. However, these addresses are not convenient for finding nodes and are

typically unacceptable to human users. To fill this need, a minimal Service Location Protocol (SLP) has been developed named nanoSLP. It provides a sort of URL resolution and distributed resource discovery for nanoIP. Furthermore this must be done as efficiently as possible, both in terms of complexity and overhead in order not to cancel out the gains made with nanoIP.

This protocol is based on SLPv2, starting from the "minimal SLPv2" recommendation in [9], and following that to the extreme. NanoSLP is also similar to the RDP protocol developed for mobile IP networks [10]. It uses nanoUDP as a transport, simplifies the query/reply messages, and removes complex functionality. Functions of SLPv2 such as attribute requests, TCP usage, LDAP queries, multicast, and scopes are not included.

NanoSLP consists of simple Service Agent (SA) and User Agent (UA) capabilities. Resource queries can be broadcast by UAs based on service type as in SLP. A query reply is returned by SAs with matching service types, this contains the URL [11] of that service. Optionally, a more exact query can be included with the resource query. NanoSLP uses a simple XML-based query language roughly based on LDAPv3 [12]. The following shows an example query and reply.

```
--> query
service-type = service:http
query = (name='Temperature Sensor')

<-- reply
URL = http://00:4A:C4:01:0D:9C/temp
```

NanoSLP allows for embedded devices to be found using service types or XML queries rather than with MAC addresses, without the need for a naming infrastructure. This capability can also be included within the nanoHTTP implementation, providing name resolution capability. The following URL can for example be passed on to nanoHTTP.

```
http://(name='Temperature Sensor')
```

C. XML

So far we have discussed the minimization of the protocols used in subnet-level communication. The other important aspect is of course content compression, lowering the bandwidth requirements for moving around application data. For video, still pictures and audio, standard compression techniques used on the Internet work quite well, although schemes designed specifically for embedded platforms are also emerging. However, for text-based data the situation is not so good.

The standard method of representing structured text-based data is to use the Standard Generalized Markup Language (SGML), or the simpler eXtensible Markup Language (XML). From the embedded point of view XML shares many of the problems of HTTP discussed in the previous subsection. It is difficult to parse (especially on hardware) and is very space-consuming. Again, the preferred solution is to use a binary representation of XML in the transmissions between nanoIP hosts. Fortunately, a standard binary encoding exists for XML, namely the WAP binary XML content format (WBXML) [13] designed for the WAP forum by W3C. Using this along with nanoHTTP provides a fairly compact way to move ASCII data between nanoIP nodes. WBXML is also useful as it can be browsed directly with many existing WAP browsers.

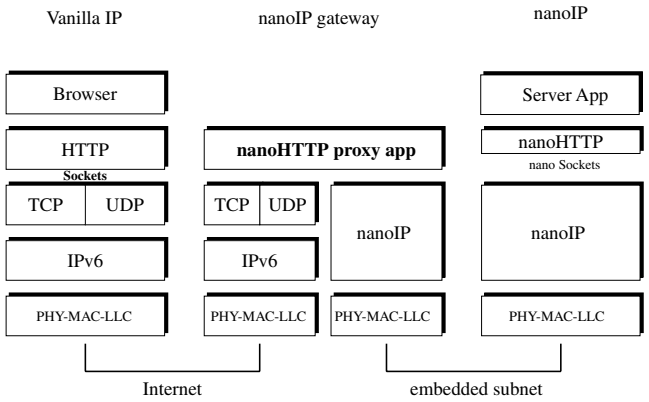

Fig. 5. The stack of a nanoIP gateway.

IV. Gateway Architecture

Instead of having direct Internet connectivity, nanoIP devices can be made accessible to TCP/IP networks with a nanoIP gateway. A nanoIP gateway has both TCP/IP and nanoIP stacks, and runs an HTTP proxy application, see Figure 5. Portable devices with built-in TCP/IP stacks such as PDAs and mobile phones make natural gateway platforms. To a node making an HTTP request to a nanoIP server through a gateway, the service seems to be located on the gateway. Mappings can be made for nanoIP devices, making them visible either on a specific URL of the gateway or on a port number of the gateway. For services that are largely static, caching can be used to completely shield nanoIP devices from the load of Internet requests. Not all request are directed to embedded devices from the outside, often devices need to communicate with the Internet. Mappings can just as easily be made so that a nanoIP device can access a resource on the Internet. The gateway can control which services are mapped from the embedded devices and which are available from the Internet. In addition, it can protect devices from a variety of security threats. As an example, the temperature service of a nanoIP device located at `http://00:50:15:02:00:8F/temp` can be mapped to a URL of the gateway `http://mygateway/temp` thus appearing to be part of the gateway itself.

In addition to performing an HTTP proxy function, the nanoIP gateway can also act as a service discovery proxy. Service descriptions for devices with HTTP to nanoHTTP mappings can be collected on the gateway. When a service query is sent to the gateway, it can check the descriptions of nanoIP devices in addition to its own. If a match is made for a nanoIP device's description, the gateway responds with the URL of the HTTP mapping. Thus the service looks as if it is part of the gateway.

TABLE I

THE RAM/ROM USAGE OF THE NANOIP IMPLEMENTATION WRITTEN IN C. THE MTU SIZE IS 254 BYTES IN THIS EXAMPLE. THE GATEWAY HERE ASSUMES 300 BYTES OF CACHED DATA, 5 ACTIVE CONNECTIONS, AND 10 MAPPINGS.

	RAM (Bytes)	Flash ROM (Bytes)
nanoUDP	282	∼500
nanoUDP + nanoTCP	300	∼1000
gateway proxy	750	1500

V. REFERENCE IMPLEMENTATION

A nanoIP reference implementation for small microcontrollers and linux kernels has been developed to confirm the protocol design. Low end μCs from the Atmel AVR and Texas Instruments MSP430 series are used as target platforms. This reference stack has been a proof-of-concept and design tool for research with nanoIP.

One overall goal of nanoIP is to minimize RAM and ROM usage through reduced complexity. This is important as many consumer embedded devices have not been designed specifically for networking, leaving minimal μC space for a networking stack. A simplified summary of RAM/ROM usage from this implementation can be found from the above table. The implementation is not of production quality, allowing for further reductions in size. However, it has given interesting insight into the complexity areas of the protocol and interfacing with various wireless links. NanoIP has been useful over everything from Ethernet and RS232, to Bluetooth and proprietary 433MHz equipment. The socket interface and port numbers, along with nanoHTTP and nanoSLP make it possible to implement quite advanced applications. This includes everything from discoverable smart environments to mini webservers. An extremely embedded OS, such as the tinyOS [14], would be a very good match for the nanoIP protocol. A public release of the nanoIP reference implementation is to be considered in the future, along with performance results from the implementation.

VI. CONCLUSIONS

This research work on nanoIP aims to provide a real alternative to TCP/IP for embedded pervasive networking. A compact design, with equivalent IP features (from the application point of view) allows for pervasive networking with embedded devices without the complexity and power consumption of larger protocols. Minimal application protocols complete the stack to give a fully functional application environment for embedded devices.

Future work on this area, in addition to implementation results, will definitely come in the future. Security for embedded devices, such as SPINS [15], should be applied to the nanoIP framework. The nanoIP gateway will be examined more deeply. We are also interested in studying the applicability of nanoIP to other fields such as automation and process engineering. Finally, scalability and overall architecture issues for embedded networking are quite open research areas.

VII. ACKNOWLEDGEMENT

We would like to thank all our colleagues in the nanoIP project team for useful discussions, collaboration, and a fun time. Our colleagues in WIRSU have helped nurture and integrate our ideas and work. This work is in part funded by Nokia through TEKES and the Academy of Finland (grant 50624).

REFERENCES

[1] F. Bennett et al., "Piconet: Embedded Mobile Networking," *IEEE Personal Comm.*, vol. Oct., pp. 8–15, 1997.

[2] J. Weatherall and A. Jones, "Ubiquitous Networks and Their Applications," *IEEE Wireless Comm.*, vol. Feb., pp. 18–29, 2002.

[3] Ed Callaway et al., "Home Netorking with IEEE 802.15.4: A Developing Standard for Low-Rate Wireless Personal Area Networks," *IEEE Communications Magazine*, vol. Aug., pp. 70–77, 2002.

[4] J. Kahn, R. H. Katz, and K. S. J. Pister, "Next Century Challenges: Mobile Networking for "Smart Dust"," *ACM Mobicomm*, 1999.

[5] Z. M. Mao, H. W. So, F. Wong, and S. Zhuang, "Mobile Mote," *http://www.cs.berkeley.edu/ zmao/cs268/project/MobileMote.pdf*, 2000.

[6] D. Göthberg, "MicroIP," *http://www.micro-ip.org/*, 2000.

[7] J. Riihijärvi, P. Mähönen, M. Saaranen, J. Roivainen, and J.-P. Soininen, "Providing network connectivity for small appliances: a functionally minimized embedded web server," *IEEE Communications Magazine*, vol. vol. 39, no. 10, pp. 74–79, oct 2001.

[8] R. Fielding et al., "Hypertext Transfer Protocol – HTTP/1.1," *IETF RFC 2616*, June 1999.

[9] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," *IETF RFC 2608*, June 1999.

[10] C. Perkins and H. Harjono, "Resource discovery protocol for mobile computing," *ACM Mobile Networks and Applications*, vol. vol. 1, no. 4, 1996.

[11] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)," *IETF RFC 1738*, dec 1994.

[12] M. Wahl, T. Howes, and S. Kille, "Lightweight Directory Access Protocol (v3)," *IETF RFC 2251*, December 1997.

[13] B. Martin and B. Jano, "WAP Binary XML Content Format," *http://www.w3.org/TR/wbxml*, June 1999.

[14] D. Culler et al., "A Network-Centric Approach to Embedded Software for Tiny Devices," *Intel Research IRB-TR-01-001*, 2001.

[15] A. Perrig et al., "SPINS: Security Protocols for Sensor Networks," *ACM Mobicom*, 2001.