FLIP: a Flexible Protocol for Efficient Communication Between Heterogeneous Devices

Ignacio Solis, Katia Obraczka and Julio Marcos
Information Sciences Institute, University of Southern California
{isolis,katia}@isi.edu, jmarcos@usc.edu*

Abstract

Interconnecting heterogeneous devices, that is, devices with varying capabilities, has raised new challenges in the design of network protocols. This paper describes the design of the Flexible Interconnecting Protocol, or FLIP*, whose goal is to interconnect heterogeneous devices. FLIP is a flexible protocol that addresses the needs of heterogeneous networks: it incurs little overhead when run by simple devices, while still providing a range of functions that can be performed by more sophisticated devices.*

We describe a simplified implementation of FLIP under Linux. We also conducted a preliminary evaluation of FLIP's overhead and functionality in the context of IP (IPv4 and IPv6) and sensor network environments. FLIP incurs reasonably low overhead when providing IPv4 and IPv6 functionality (1 and 3 bytes respectively), yet it does particularly well in the case of small payloads. When compared to a sensor-specific protocol, FLIP incurs a small overhead increase while still providing full protocol functionality.

1. Introduction

One of the implications of "being connected anywhere, anytime" is that networks will become more heterogeneous. Network heterogeneity will manifest itself in terms of increased diversity in communication medium technology (such as wired, wireless, satellite, and optical links), as well as in the types of devices networks will interconnect. We envision that the Internet of the future will interconnect not only traditional desktop and laptop computers, but also unconventional devices such as sensors, actuators, and home appliances. These unconventional devices, whose power, processing, and communication capabilities differ widely, will form network clouds, which will be interconnected among themselves and with the existing IP infrastructure.

While many of the Internet protocols have proven successful and long-lived in traditional networks, they were not designed to accommodate the degree of device heterogeneity that will characterize future internets. Consider IP (and both its instances, i.e., IPv4 [7] and IPv6 [13]). We argue that it adds unnecessary and sometimes prohibitive amount of complexity and overhead, especially in the case of limited-capability devices. More recently, motivated by research programs and projects on sensor networks, sensor manufacturers have implemented protocols specifically tailored for sensing devices. Because they are so specialized, these protocols will not be able to accommodate more sophisticated and powerful devices.

In this paper, we describe the design and implementation of a network-layer protocol whose main goal is to accommodate devices with varying power, processing, and communication capabilities. The proposed protocol, Flexible Interconnecting Protocol, or *FLIP*, will operate among devices in the farthest branches/leaves of an intranet while providing inter-network connectivity with other clouds and with the existing IP-based Internet infrastructure. To achieve its main design goals of flexibility and efficiency, FLIP's overhead (both in terms of per-packet overhead and protocol complexity) is dependent on the capabilities of the particular device running FLIP and the functionality needed by the application. For anemic devices, FLIP's close to optimal overhead not only saves bandwidth, but, more importantly, energy.

Figure 1 shows FLIP's position in the protocol stack. FLIP is designed to run atop a data link layer protocol (such as IEEE 802.11 [17]) and provide functionality all the way up to the application layer, replacing the functionality of network and transport protocols. The FLIP layer can be very "thin", which means that FLIP provides minimum functionality; this is the case of the version of FLIP that very simple devices like sensors would run. On the other hand, FLIP could provide functionality (or a subset thereof) of a "heavy-duty" transport protocol like TCP. It is the application designer's choice what services are required and should be included in FLIP. FLIP can also run underneath

*The authors have since moved and can now be reached at {isolis,katia}@cse.ucsc.edu and julio@donjulio.net

any transport- or even network-layer protocol.

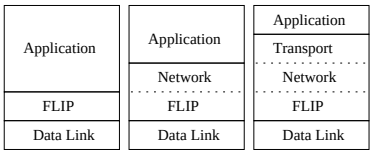

Figure 1. FLIP in the protocol stack.

The remainder of this paper is organized as follows. In the next section we highlight the main principles that guided the design of FLIP. Section 3 describes FLIP's protocol specification, including FLIP's header fields. In Sections 4 and 5, we present our implementation of FLIP and results of a preliminary evaluation of the protocol, respectively. Section 6 describes related work and Section 7 presents concluding remarks and our future work plans.

2. FLIP Design Principles

The overhead and complexity of a protocol is directly related to the functionality the protocol provides. Recall that FLIP's main goal is to accommodate a range of devices with different capabilities and yet provide the functionality applications need. Thus FLIP allows application programmers to select just the functionality they need, without incurring the overhead associated with functions they do not need. Furthermore, the ability to select a subset of protocol functions allows FLIP to accommodate a range of devices from very simple sensors to desktop computers. For instance, if the application needs packets to age, then the application programmer can "turn on" FLIP's Time-To-Live (TTL) field (we describe FLIP's TTL and other fields in Section 3 below) and assign its maximum value. At each hop, the packet's TTL value will be examined and in the case it is greater than the maximum value, the packet will be discarded. If not, the node increments the TTL and forwards the packet. Users can also "turn on" the length field, whose value will be calculated as part of composing a packet.

In its simplest form, FLIP does not even provide routing as in some scenarios FLIP can be used, routing is done by the application which uses special information such as geographic positioning or power conservation. Some of these scenarios may use small, very simple devices which would only be encumbered with routing: these are just end devices and do not have the required capabilities to perform the routing function effectively. End-to-end reliability and ordering are not included in FLIP's simplest form.

A more complete version of FLIP could provide similar functionality to TCP/IP. Applications could, however, disable requirements such as ordered delivery while keeping reliable delivery.

3. Protocol Description

Recall that FLIP's main design goals are flexibility and efficiency. To achieve these goals, FLIP makes use of *extensible* headers, which allows customization of the header fields as a function of the underlying device and the target application. Thus FLIP's extensible headers allow near optimum power, bandwidth, and processing overhead by excluding unneeded information.

FLIP packets are composed of a meta-header, the header fields and the payload. The meta-header indicates which header fields are included in the packet and consists of an array of bits, or a bitmap. If a header field is included in the packet, then the corresponding bit in the meta-header is one, otherwise, it is set to zero.

In order to minimize the bitmap's overhead, we split it into one-byte pieces. Each byte contains a continuation bit that indicates if more bitmap pieces follow. Figure 2 shows an example of the FLIP meta header. Note that the continuation bit is the first bit of each byte. This ensures that, in the 2-byte version of FLIP's extra simple packet (ESP) (FLIP's ESP mode will be described below), the payload occupies contiguous bits.

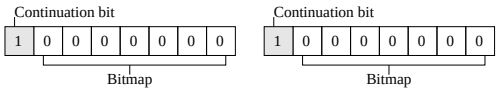

Figure 2. FLIP Meta Headers

Consider a scenario that only requires the *length* field, whose presence bit lies in the first byte of the meta-header. Then, the packet will only have to carry the first byte of the meta-header, which will have the bit corresponding to the length field on, and the others, including the continuation bit, off.

In some cases, the target application need only send small amounts of data every time with no header. Sensor network environments are a good example of such a scenario: sensors simply broadcast data related to what they are sensing. In these cases, even a 1-byte meta-header to indicate that no header is needed is a lot. For instance, if sensors broadcast 1-byte data, then 1-byte headers results in 50% overhead, which is often too high.

To address these scenarios, FLIP offers the *extra simple packet*, or ESP. The second bit of the meta-header, that is, the one following the continuation bit in the first byte, is the ESP bit. If this bit is set, it indicates an ESP. The use of the continuation bit in the ESP allows for 1- and 2-bytes ESPs. While a 1-byte ESP, that is, one with the continuation bit off, contains 6 data bits, a 2-byte ESP allows for 14 bits of data (all the 8 bits of the second byte will be counted as data). Figure 3 depicts both ESP cases.

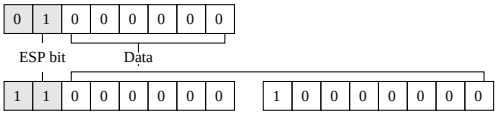

Figure 3. Extra Simple Packet (ESP)

FLIP's ESP addresses the need for a "barebone" protocol, which will be used by applications that need to send small pieces of data with no overhead. FLIP's regular meta-header bitmap covers the more general cases where some fields are required and some are not, thus optimizing average use.

As shown in Figure 4, FLIP's current meta-header bitmap spans 3 bytes, including three continuation bits and the ESP bit. The last portion is still undefined since not all uses have been considered. Defining the missing fields is an item we will address as part of our future work on FLIP.

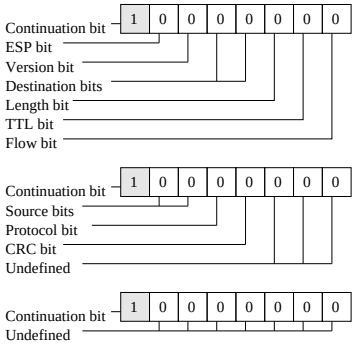

Figure 4. The FLIP meta header bitmap

A sample of a complete FLIP packet can be seen in Figure 5. The shaded area is the header and rest is the payload. The definitions of the FLIP header fields are given below. The ordering of the fields was determined so as to optimize packet overhead for very simple applications and devices. More complex devices and applications can normally amortize the cost of having more meta-header bytes.

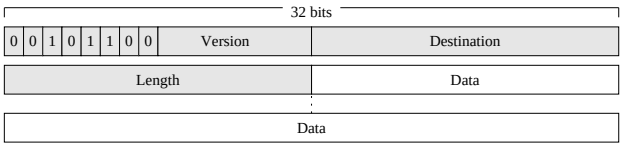

Figure 5. FLIP sample packet

- **Version** is 1 byte in length, The 4 higher order bits are considered the version field. The current version is version 0. The 4 lower order bits are considered the priority field. If a packet lacks the version field, it's considered version 0 and priority 0.
- **Destination** is a variable-length field. The corresponding bitmap field is composed of 2 bits, whose value determines the size of the field. If the bitmap bits are set to:
  - 00 indicates the destination field is not present.
  - 01 indicates we have a destination field of 2 bytes in length carrying a FLIP *address*.
  - 10 indicates it is a 4-byte field
  - 11 indicates the field is 16 bytes in length.

  It is no coincidence that this matches IPv4 [7] and IPv6 [13] addresses.
- **Length** is two bytes in length, which means that the maximum packet size is limited to 64 KBytes.
- **Time to Live (TTL)** is one byte in length and is interpreted by the application. It can be used to limit the scope of a packet.
- **Flow** is four bytes in length. As the name implies, this field is intended for flow identification.
- **Source** is a variable-length field and its length is determined by 2 bits in the meta-header exactly the same way as the destination field.
- **Protocol** is one byte in length and indicates the protocol type. This refers to the same field in IPv4 or next header in IPv6.
- **Checksum** is two bytes in length and checks the packet payload. It is calculated similarly to the IP Checksum.

For increased flexibility, FLIP also allows for *user-defined* header fields. If the continuation bit of the third meta-header byte is set, it indicates that user-defined header fields are included in the packet. Each user header field definition is one byte in length. The first bit, as usual, is the continuation bit, the remaining 7 bits are defined and interpreted by the application.

An example of a user-defined field would be geographic positioning, latitude and longitude. A host might use this information for packet processing or routing, but it might not be directly related to application level. Another example might be a minimum power field. In a sensor network environment it might be useful to determine the power (lifetime) of a certain route, and this might be done by using such a field.

One could argue that these user-defined header fields stretch the line between what should be in the header versus the payload. In other words, FLIP's user-header fields could be seen as protocol layering violation. However, we claim that, when networking in heterogeneous environments, the device layer need to be more exposed to the application developer.

4. Implementation

As proof of concept, we implemented a barebone version of FLIP in the Linux 2.3 kernel. We generated a patch for the kernel which allows the inclusion of FLIP at compile time or as a loadable module. Linux is making its way into devices of various kinds and capabilities; having a Linux implementation of FLIP will allow us to conduct live experiments in heterogeneous environments.

Below the FLIP code lies the device code, specifically the device output/input queues, through which FLIP sends/receives data. When data is received, the receiving device passes it to the FLIP layer which queues the data on the corresponding socket.

FLIP uses the BSD socket abstraction to interface with applications. In order to send and receive data using FLIP, application programmers will use the same set of socket system calls they would use to handle TCP/IP communication endpoints. For instance, to create a FLIP socket, all they have to do is request a socket of family `AF_FLIP`.

```
char* buf = "Hello World";
__u16 addr;

s = socket(AF_FLIP,SOCK_RAW,FLIP_NO_ESP);
addr = htons(1000);
setsockopt(s,SOL_FLIP,FLIPO_DESTINATION,
           &addr,sizeof(addr));
write(s, buf, strlen(buf));
```

Figure 6. Sample application code

Figure 6 illustrates the FLIP API. In this example, a FLIP socket of type `SOCK_RAW` (allowing the programmer to modify most of the fields) is defined. The `CAP_NET_RAW` capability is required to use the socket if capabilities are being used. FLIP sockets will eventually be able to support datagram and stream once transport layer functionality is implemented. `Socket`'s last parameter is used to select ESP or non-ESP mode. With the current implementation, the programmer cannot change the ESP mode once the socket is created.

The programmer can then use `setsockopt()` to set the necessary header fields. For instance, address fields (source and destination) identify a given communication end-point. If a socket is assigned an address, that socket will only receive packets with that address in the destination field. The address of a FLIP traffic source is set through the `FLIPO_SOURCE` option. If no address is assigned to a socket, FLIP will not set the source address on outgoing packets from that socket.

Since ESP packets have no headers, and thus no destination or source addresses specified, ESP sockets always get all packets. In this example, the destination field is defined as a 16-bit FLIP destination and is set with the `FLIPO_DESTINATION` option.

The `getsockopt()` call is used to read header definitions for a certain socket, as well as to read the header fields of incoming packets on that socket. As previously pointed out, to achieve flexibility and efficiency, our design exposes the network layer to the application programmer.

In order to speed up packet header construction, we cache header information for every socket that has been defined. Dynamic header fields, which change from packet to packet, are computed on the fly before the packet is sent. *Packet length* and *checksum* are examples of dynamic header fields.

In the current implementation, we use Ethernet as the MAC layer protocol. Like RF wireless access, Ethernet assumes a shared broadcast medium. A unique protocol number was selected as Ethernet's *next protocol* field. Using Ethernet as the underlying MAC protocol means that FLIP packets must be at least the size of the minimum Ethernet payload. Consequently, in this implementation, we cannot take full advantage of FLIP's ESP mode.

Furthermore, accurate evaluation of FLIP's power requirements cannot be conducted with the current wired implementation. A comparison between the power requirements of FLIP and other protocols is an item in our future work list.

5. Evaluation

In this section we present a preliminary evaluation of FLIP. We compare its functionality and overhead with a more "traditional" protocol, namely IP. We then evaluate how FLIP integrates into an IP environment. We also evaluate FLIP in the context of a sensor network environment.

5.1. FLIP and IP

We should point out that both FLIP and IP were designed to address different goals and target environments. Thus comparing them is not really fair to either. While the IP layer provides a fixed set of functions, FLIP's functionality and overhead are application-dependent. In other words, the application determines which fields are to be included in the FLIP packet header. Therefore, applications send just what they need, avoiding the cost of transmitting and processing unnecessary information.

Take for example an application that sends out data in 1000-byte chunks. Using IPv4, the overhead would be 20 bytes (corresponding to the IPv4 header), which is not a lot relative to the payload. However, if hosts are just sending 1-byte heartbeat messages (e.g., either their address or some form of identification), then 20 bytes of header would

Table 1. Packet size comparison

Data	none			1 byte			1000 bytes		
	IPv4	IPv6	FLIP	IPv4	IPv6	FLIP	IPv4	IPv6	FLIP
IPv4 functions	20	N/A	21	21 (2000%)	N/A	22 (2100%)	1020 (2%)	N/A	1021 (2.1%)
IPv6 functions	N/A	40	43	N/A	41 (4000%)	44 (4300%)	N/A	1040 (4%)	1043 (4.3%)
Dest. & Source	20	40	10	21 (2000%)	41 (4000%)	11 (1000%)	1020 (2%)	1040 (4%)	1010 (1%)
Dest. only	20	40	3	21 (2000%)	41 (4000%)	4 (300%)	1020 (2%)	1040 (4%)	1003 (0.3%)

seem unacceptable. Fields such as fragmentation information, ToS, or even packet length (in the case of fixed-size packets) would be adding unnecessary overhead and wasting network, and even more importantly, device resources (such as power). If IPv4 is used, the message would be 21 bytes long, where only 1 byte is payload. The corresponding FLIP packet would come down to 7 bytes: 2 meta-header bytes, 4-byte address, and 1-byte heartbeat, which results in a 200% increase in efficiency. If MAC addresses can be used for host identification and the data can fit in 6 bits (instead of 1 byte), the resulting FLIP packet will be only 1 byte long.

When comparing the functionality of the two protocols, we need to examine the issue of header compatibility. IP header fields are easily mapped into FLIP fields. Indeed, FLIP was designed with IP-compatibility in mind. Clear examples of FLIP-IP compatibility are the 2-, 4-, and 16-byte address fields that accommodate IPv4 and IPv6 addresses.

FLIP is fully compatible with IPv6. To emulate it's functionality the higher layers would select version, flow, length, protocol (for next header), TTL (for hop limit) and 128 bit addresses for source and destination. The overhead of using FLIP instead of IPv6 is three bytes: two bytes from the meta-header and one extra flow id byte (FLIP's flow id is 4 bytes long while IPv6's is only 3). This additional overhead is relatively low: it results in only 7.5% header size increase.

If we want to achieve full IPv4 compatibility however, we need to assign a slightly different meaning to the flow field. FLIP allows this since it can be tightly integrated with the higher layers. For IPv4 emulation we would choose the same fields as IPv6 plus Checksum. The flow field would be considered as IPv4's id + fragment information, which accommodates into 4 bytes. Even the IPv4 options can be included in the user defined fields. Without these fields, and taking a look at only the normal 20 byte IPv4 header, our overhead is 1 byte. That would be because we have 2 bytes of meta headers, don't have a header length field (4 bits) and our priority field is only 4 bits compared to the IPv4's 1 byte ToS field.

Of course if FLIP's flexibility is not needed, then this small increase might be unwarranted. In most situations, devices that speak IP do not need FLIP. Gateways interfacing FLIP clouds and IP networks will need to speak both. However the main point in comparing FLIP's and IP's functionality is to show that FLIP can be used by very simple devices with minimum overhead, and, at the same time, provide IP-style functionality when needed with minimum cost.

FLIP's main drawback, when compared to IP (or any "fixed-header" protocol), is associated with the fact that header parsing becomes a more involved task. Clearly, higher header processing overhead implies that it takes longer to forward packets. Regarding protocol implementation, communication between layers is more complicated since now varying-size data has to be passed between layers. Furthermore, allowing users to modify protocol header fields raises implementation correctness issues.

Table 1 shows a comparison between the use of IP and FLIP. The columns define the payload size and the protocol used. The rows define the functionality expected of the protocol. In the case of Destination and Source we mean 4 byte addresses, and in the Destination only we mean 2 byte addresses. It might seem like an unfair comparison but we are trying to illustrate the flexibility of FLIP. The cells show the packet size. The number in parenthesis is the size of the header compared to the payload.

5.2. FLIP-IP Integration

FLIP's goal is **not** to replace but rather **extend** the scope of IP to interconnect clouds of varying capability devices to the existing IP infrastructure. Below, we examine different FLIP-IP integration strategies.

One way of integrating the two protocols is through simple encapsulation. For example, in order to interconnect FLIP-capable islands across an IP infrastructure, FLIP tunnels can be used. Upon leaving a FLIP cloud, FLIP packets are encapsulated into IP datagrams by a FLIP-IP gateway. When reaching the FLIP-capable network destination, IP-FLIP gateways restore the original FLIP packets, stripping off the IP envelope.

A less common scenario is to tunnel IP traffic through FLIP networks. IP datagrams could be encapsulated in a header indicating a "IP-in-FLIP" type and an address. FLIP routers, upon viewing the "IP-in-FLIP" type would do some fast path forwarding scheme, bypassing FLIP's regular mechanisms.

5.3. Sensor Networks

Sensor networks and their applications are one of FLIP's key target application domains. In most sensor network scenarios, the goal is energy conservation as sensing devices rely on relatively short lifetime batteries. Typically, sensor network applications imply that sensors will be left on the field unattended for extended periods of time and must conserve energy in order to maximize the whole network's operational time.

Sensor devices, and implicitly sensor networks, are data driven in the sense that the whole network cooperates on the task of communicating data from sensors to end users. In these kinds of scenarios, FLIP optimizes communication among nodes by only transmitting the required information with minimum protocol overhead. For instance, FLIP's ESP provides application programmers with a very lightweight packet that can be used to coordinate between peers in radio range or send small data chunks. The inclusion or exclusion of destination and source fields could determine the scope of the data as routable or one-hop (such as a "running out of battery" or "hello" messages). FLIP's different address types allow proposals such as the address-free architecture [8] to coexist with more traditional addressing schemes.

In order to evaluate how FLIP addresses the needs of sensor network applications, we selected as a case study the directed diffusion architecture [1]. Directed diffusion is a communication paradigm for sensor networks which establishes *interests* for specific data (e.g., number of cars that flow through busy intersections during rush hour). Relevant data flows towards nodes that expressed interest in named information. Routing is done by the application, which aggregates data when possible.

We examined a sample implementation of directed diffusion that uses commercially available sensing nodes [14]. Sensors exchange data in the form of attribute-value pairs. This data is transmitted using the fixed-header packet format shown in Figure 7. Notice that this protocol is completely tailored, and thus optimal, to the requirements of directed diffusion.

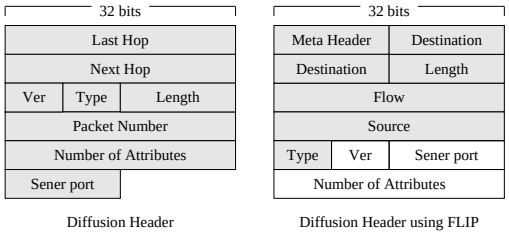

Figure 7. Diffusion header

Now consider using FLIP as the underlying network protocol for this implementation of directed diffusion. Version information is application related, thus is carried in the payload. This is also the case of *number of attributes* and *source port*. *Last* and *next hop* information is carried in the source and destination fields, respectively. *Message type* and *length* correspond to FLIP's protocol and length fields. *Packet number* can be mapped into *flow*. Consequently, the total overhead incurred by FLIP is the 2-byte meta-header. Figure 7 shows the resulting packet, the header part being the shaded one.

Since FLIP was not available to directed diffusion implementors, we can only speculate how FLIP could help this diffusion implementation. FLIP definitely offers a lot more functionality than diffusion's current static headers. FLIP could be used as the underlying protocol interconnecting sensors among themselves and to other devices such as data gatherers.

Table 2. Special situations packet size

	IPv4	IPv6	FLIP
6bit data	21 (2666%)	41 (5333%)	1 (33%)
14bit data	22 (1142%)	42 (2286%)	2 (14%)

Table 2 shows a comparison between FLIP and IP for some sample scenarios. The first column describes what is being transmitted. This is obviously modeled after the FLIP ESP packets. It is aimed at showing the inefficiencies of IP in these situations. The number in parentheses is the header overhead (compared to the size of the data).

Consider directed diffusion, whose original header is 22 bytes long. If IPv4 was used to implement directed diffusion, it would incur an overhead of 9 bytes, and would have to carry *packet number* information in the payload. In the case of IPv6, the overhead would increase to 29 bytes. If FLIP was to be used, the overhead would be only 2 bytes.

We anticipate that the benefits of using FLIP will be evident in scenarios where varying-capability devices need to be interconnected. For instance, in a sensor network using FLIP as the interconnection protocol, sensors can save considerable resources (especially power) by using FLIP's ESP, yet they can still communicate with more sophisticated devices, such as data gatherers.

6. Related Work

Communication protocols for wireless networks have been an active research area with efforts like Packet Radio [18], GloMo [2] and the IETF's Mobile Ad-hoc Networks (manet) working group [6]. Recently, some research has turned to embedded systems and sensor networks. To our knowledge, FLIP is the only initiative to develop a protocol to interconnect heterogeneous devices. Almeroth et

al. [10] introduced the main concepts behind FLIP.

The SCADDS [15] project has done research in sensor networks, developing the Directed Diffusion [1] architecture. The Dynamic Sensor Networks [16] is also in the area trying to take advantage of GPS. The WINS [5] project describes a basic sensor network environment. Piconet [3] has deals with a low-range radio network.

Also related to this research is network header compression. Work has been done before [9, 12, 11, 4] and normally focuses on TCP/IP and improving performance.

7. Conclusion and Future Work

This paper described the design and implementation of FLIP, a network protocol whose goal is to accommodate varying capability devices. FLIP uses customizable headers to satisfy, with minimal overhead, the requirements of a wide-range of applications and devices. We implemented FLIP under Linux and used the BSD socket abstraction to make FLIP available to application programmers.

We conducted a preliminary evaluation of FLIP comparing its overhead and functionality with IP (IPv4 and IPv6) having an small overhead (1 and 3 bytes respectively) for normal IP situations. We showed that the overhead of the header could be greatly reduced in situations that demand less functions than those offered by conventional IP, specially with small payloads. We also evaluated FLIP in the context of sensor network environments, where a small increment in the overhead can provide a protocol with complete flexibility.

This work on FLIP helped us identify several directions we plan to explore. First, we plan to complete the design of FLIP, including (1) FLIP's complete header specification and (2) design and specification of other components of the FLIP layer, such as routing, and transport layer functionality (e.g., reliable/ordered delivery).

In completing FLIP's design, we plan to address the needs of current and future applications. The current meta-header bitmap has been defined to provide IP-like functionality in an efficient way, minimizing unneeded fields. The current design also tries to minimize protocol overhead in the case of very simple devices such as sensors. The question is then what will be the requirements of applications to come? FLIP's inherent flexibility will allow it to evolve and adjust to applications' needs. Issues that we will need to investigate include: what is the adequate bitmap size and what is the role of header field ordering on FLIP's efficiency.

Embedded device technology evolution will also influence FLIP. If future devices are reasonably powerful in terms of processing and storage capabilities, the FLIP stack itself will not need to be heavily optimized in terms of processing overhead. Optimizing transmission overhead (in order to conserve energy) will likely be the main goal.

Another direction we plan to explore is how FLIP can be integrated with different MAC protocols. We believe that exposing the MAC and physical layers will greatly increase FLIP's efficiency. For instance, if the underlying MAC protocol provides integrity check, we may not need to verify packet integrity at the FLIP layer.

References

[1] C. Intanagonwiwat, R. Govindan and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *6th International Conference on Mobile Computing and Networking (MobiCom)*. ACM, August 2000.

[2] DARPA. http://www.darpa.mil/ito/solicitations/glomo/ glomobrief.html, February 1995.

[3] F. Benner, D. Clarke, J. Evans, A. Hopper, A. Jones and D. Leask. Piconet: Embedded mobile networking. *IEEE Personal Communications*, 4(5):8–15, October 1997.

[4] G. Mamais, M. Markaki, M. H. Sherif and G. Stassinopoulos. Evaluation of the casner-jacobson algorithm for compressing the RTP/UDP/IP headers. In *ICSS98*, June 1998.

[5] G. Pottie and W. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43, Issue 5, May 2000.

[6] IETF. http://www.ietf.org/html.charters/manet-charter.html, July 2000.

[7] J. B. Postel. Internet Protocol, RFC-791, September 1981.

[8] J. Elson and D. Estrin. An address-free architecture for dynamic sensor networks. Technical Report 00-724, Computer Science Department USC, January 2000.

[9] J. Lilley, J. Yang, H. Balakrishnan and S. Seshan. A unifi ed header compression framework for low-bandwidth links. In *6th International Conference on Mobile Computing and Networking (MobiCom)*. ACM, August 2000.

[10] K. Almeroth, K. Obraczka and D. De Lucia. A lightweigth protocol for interconnecting heterogeneous devices in dynamic environments. In *International Conference on Multimedia Computing and Systems (ICMCS)*. IEEE, June 1999.

[11] M. Degermark, B. Nordgren, S. Pink. IP header compression, RFC-2507, February 1999.

[12] M. Degermark, M. Engan, B. Norgreen and S. Pink. Low-loss TCP/IP header compression for wireless networks. In *International Conference on Mobile Computing and Networking (MobiCom)*. ACM, November 1996.

[13] S. E. Deering and R. Hinden. Internet Protocol, version 6 (IPv6) specifi cation, RFC-1883, December 1995.

[14] Sensoria Corporation. http://www.sensoria.com/, August 2000.

[15] USC/ISI. http://www.isi.edu/scadds/, May 2000.

[16] USC/ISI, UCLA, Virginia Tech. Dynamic sensor networks (dsn), http://www.east.isi.edu/div10/dsn/, August 2000.

[17] V. Hayes. IEEE standard for wireless LAN medium access control and physical layer specifi cations. Technical Report 802.11-1997, IEEE, June 1997.

[18] Packet radio topics in professional journals, http://www.tapr.org/tapr/html/biblio.html, August 2000.