

Facilitating Network Auto Configuration In Next Generation Internet Protocols

Matthew Jakeman, Andrew Brampton, Stephen Pink
Computing Department
Lancaster University
Lancaster, UK
Email: {jakeman, brampton, pink}@comp.lancs.ac.uk

Abstract— The infrastructure of the modern Internet has become a complex mesh of varying network types. A single network protocol cannot optimally support every underlying technology and the diverse nature of these networks places increasing strain on the concept of running IP over everything and everything over IP. The introduction of new protocols and services also forces network administrators to employ techniques such as tunneling to ensure end-to-end IP connectivity. Unfortunately these techniques inherently require some form of efficiency trade-off and are not an ideal long term solution.

To address these issues, this paper proposes a new network layer protocol, NP++, which uses a level of indirection between the logical and physical specifications of the protocol. NP++ also enables the protocol to automatically configure which physical mapping is used over a link with no direct input from the user. This allows the protocol to change its transmission characteristics depending on the type of underlying network while presenting a unified view to the upper layers. This ensures a higher level of flexibility along with the potential to increase efficiency. The implementation of the NP++ prototype is also demonstrated with a view to encouraging its use when researching next generation Internet technologies.

I. INTRODUCTION

As the Internet has evolved, many optimisations have been created that attempt to make the best use of the underlying network mediums being used. A large driving force for network optimisations is the increase in the diversity of the network mediums being used. At the core of the Internet are high speed networks that have low packet loss rates and low rates of jitter. However at the edges of the Internet a variety of networking technologies are being used. The most common example of this is wireless networks which have very different characteristics to the core networks such as lower bandwidth, much higher packet loss rates and higher rates of jitter.

There are also many optimisations for the way data is transmitted over networks based on application requirements. Different applications require their networking protocols to have different features to provide the best service to the user.

NP++ is a protocol that allows network protocol characteristics to change dynamically at run time. This allows the diversity of network media in multi-hop network to be exploited.

A. What about IPv6?

It is widely believed that the next generation of Internet protocols will be based around IPv6. This protocol differs only

slightly from IPv4, incorporating a 128 bit address space to replace the 32 bit space of IPv4 that is now straining to cope with the ever increasing number of Internet nodes. IPv6 was conceived in the early 1990's. The process of incorporating this new protocol into the current Internet infrastructure has taken many years and is still not complete. The authors believe this is due to a number of factors.

- 1) Service providers are unwilling to deploy the new protocol until it is absolutely necessary due to their currently being able to charge a monthly fee for providing users with public IP addresses. This is currently justified by the shortage of available addresses. The larger address space available in IPv6 would mean that there would no longer be a shortage of unique IP addresses and so the economic case for charging large fees for extra addresses would no longer obtain.
- 2) IPv6 offers few substantial benefits other than providing a larger address space. Network companies fail to see enough reasons to deploy it and the large cost involved weighs against the few benefits. This is especially true as Network Address Translation (NAT) [1] can be used in a large number of cases to help overcome the address shortage, and Carrier-Grade Network Address Translation [2] is being considered for deployment by some large ISPs.
- 3) End users are not willing to pay more to help cover the costs of a protocol upgrade. Although no new infrastructure is needed, theoretically, to upgrade from IPv4 to IPv6, it is likely that the cost of Internet access would rise to help fund the upgrades. End users are unwilling to pay more than they currently do for an upgrade that would offer very few visible benefits.
- 4) ISP's use a large number of custom scripts and programs to monitor and provide services on their network. These would all have to be re-written to make them compatible with IPv6. This could involve re-training their programming staff to become familiar with new APIs as well as a substantial amount of development time being consumed in the rewriting process. All of this adds to the cost of upgrading the networks to be IPv6 compatible.

The reasons listed above, among others, have led to ISPs being slow to adopt IPv6 on their networks. Customers are

not a driving force for IPv6 as they will see very little benefit and most users won't even notice the change. This obviously means that customers are also unwilling to pay for this service and therefore ISPs are unwilling to provide it. A new protocol needs to offer more flexibility and more advantages than IPv6 to make it more attractive, but it also needs to be backwards compatible with current protocols.

This paper gives an overview the NP++ protocol. NP++ is an alternative protocol that offers significantly more advantages than IPv6 alone. It is able to rewrite packet headers on a hop-by-hop basis depending on a number of factors including the network architecture and application demands. This makes it more flexible across a mesh of varying networks and can provide a more flexible service to Internet users. This paper is not aimed at being an experimental evaluation of the protocols performance. More so it is aimed at giving users of the protocol an overview of its potential and the current implementation which is already in the public domain.

II. NP++ OVERVIEW

NP++ provides a method of flexibly changing the way packets are represented as they are transmitted across a network that is transparent to application developers. This allows the networking protocols to make the best use of the networking resources available to them without any intervention from a user.

To accomplish this NP++ introduces a level of indirection by mapping a single logical specification to the physical specification that best suits the underlying network, allowing the protocol to dynamically adapt to a wide range of network architectures, many of which haven't even been conceived yet. It is in effect, an attempt to future-proof the network protocol. Levels of indirection have been used in computing for many years to achieve new paradigms of computational ability. A prime example of indirection is the introduction of virtual memory, which allows the abstraction of having multiple and independent address spaces whose size can be larger than physical memory.

The *logical specification* of NP++ has a *unified header* description much like that of a traditional network layer protocol. However, it views the *physical specification* of the packet headers that are actually transmitted as a logically separate assembly of fields. Because of this, the physical packet headers are able to change on a hop-by-hop basis, allowing different control information to be transmitted at different temporal frequencies depending on the underlying network technology. This allows hop-by-hop optimisations such as label switching [3], link-specific field arrangement for faster forwarding, header compression [4], security [5], and flexible error detection and correction. Examples of these optimisations and more are discussed in Section III.

NP++ is based on IPv6. IPv6 is the default physical mapping for NP++. If no other physical mapping is defined, the IPv6 header format is used to transmit packets on a link. IPv6 is not only the default physical mapping, it is also the logical specification of NP++. IPv6 is the unifying network protocol

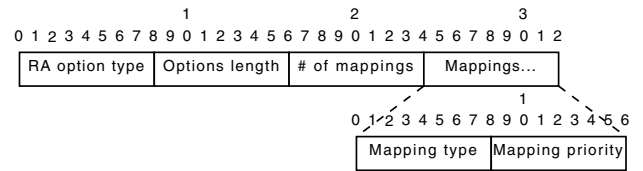


Fig. 1. Mapping Advertisement ICMPv6 Option

specification as seen from the upper layer protocols and applications. We chose IPv6 as the logical format for the packet because IPv6 embodies a general set of functionalities represented in the fields of its base and extension headers, as well as a large set of adjunct functionalities such as autoconfiguration and MTU discovery.

A. Auto Negotiation

NP++ is able to automatically negotiate between hosts and/or routers on a network to decide which physical mapping to use over a network link. To accomplish this the protocol uses new, NP++ specific ICMPv6 options.

1) *Edge Networks*: On networks at the edge of the Internet IPv6 uses *stateless auto-configuration* as a replacement for the traditional DHCP mechanisms widely deployed in IPv4 networks. One of the main components of this auto-configuration is the ICMPv6 Router Advertisement (RA). At a set interval RA's are broadcast across a network containing information such as the networks prefix, the default route etc. When a host computer connects to the network all it has to do is wait until it receives an RA containing the network information. It can then use this information to build its address and configure its network settings.

A basic RA only contains some control information along with a list of router addresses along with their preference. This on its own is not enough for a node connecting to a network to be able to configure its network settings. In order to carry more information, several ICMPv6 options are defined to carry additional information such as the network prefixes. NP++ defines a new *Mapping Advertisement* (MA) ICMPv6 option that allows RA's to carry a list of the physical mappings a router supports (known as the routers specification map). This new ICMPv6 option is depicted in Figure 1.

This option informs a node joining a network which physical mappings the router supports. The newly connected node can then compare this list with its own and choose the mapping that is best suited for both parties. This decision is made by the negotiation daemon based upon the priorities of the mappings advertised and the priorities of its own supported mappings.

2) *Host Negotiation*: Once an RA containing an NP++ *Mapping Advertisement* option has been received, it is passed on to the negotiation application on the receiving host. The application then pulls out the specification map of the advertising node and compares it to its own.

The application on the receiving node uses the consensus algorithm shown in Algorithm 1 to determine the physical

specification to be used between itself and the advertising node.

Algorithm 1 NP++ Consensus Algorithm

- a, b : network nodes
- S_a, S_b : specification maps for a and b
- $max(S)$: spec of S with highest priority number
- $pri(s, S)$: priority that s has in set S
- $avg(x, y)$: average of x and y
- $dom(a, b)$: dominant node between a and b

- 1: $S_{\cap} \leftarrow S_a \cap S_b$
- 2: **for all** $s \in S_{\cap}$ **do**
- 3: $pri(s, S_{\cap}) \leftarrow avg(pri(s, S_a), pri(s, S_b))$
- 4: **end for**
- 5: **return** $s \mid s = max(S_{\cap}) \wedge \forall t [t = max(S_{\cap}) \rightarrow pri(s, S_{dom(a,b)}) \leq pri(t, S_{dom(a,b)})]$

The consensus algorithm works as follows. The intersection of both specification maps is computed, discarding all those specifications supported by only one node. In the *intersection set*, the priority of each specification is equal to its average priority. The consensus selects the specification from the intersection set that has the highest priority number. If there is a tie, then the algorithm selects the specification that has the largest priority number in the specification map of the dominating node. The dominating node is the one that has sent out the RA.

Once this process is complete the host is able to construct a Router Solicitation (RS) ICMPv6 message. It attaches another new NP++ specific Option known as a *Mapping Finalisation* to the RS as shown in Figure 2. This message is then transmitted back to the advertising router containing the mapping to be used over the link and both hosts can set the mapping used to communicate.

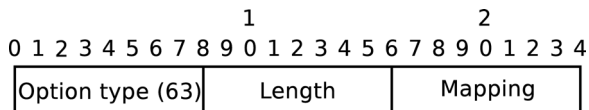


Fig. 2. NP++ Mapping Finalisation ICMPv6 Option

3) *Non Edge Networks*: IPv4 networks use ARP to obtain the Layer 2 address of hosts based on their Layer 3 address. In IPv6 the Neighbour Discovery protocol replaces the ARP mechanism. A host can send out a Neighbour Solicitation (NS) ICMPv6 message to a host to obtain its Layer 2 address. A host that receives an NS replies with a Neighbour Advertisement (NA) message containing the Layer 2 address.

As previously mentioned *stateless auto-configuration* is only widely used in edge networks. NP++ takes advantage of Neighbour Discovery on non-edge networks as a means for disseminating its MA's. Hosts on non edge networks are designated as either a server or client node by the networks administrator. Rather than using RA's to disseminate the MA's a server node encapsulates the MA in an NS and transmits

it to the client node. Upon receipt of this message the client node can perform the same consensus mechanism described in Section II-A.2 but instead of returning an RS containing the MF it encapsulates the MF in an NA and transmits it back to the server. This enables NP++ to perform the mapping negotiation phase over non-edge networks using Neighbour Discovery as well as on edge networks using *stateless auto-configuration*.

The overhead of this process is minimal. The only additional communication required over the standard ICMPv6 systems already used is the transmission of the RS containing the MF on edge networks. Neighbour discovery is already used on most IPv6 networks so the addition of the NP++ messages adds little overhead. The delay between the NS messages on non-edge networks can also be adjusted by a network administrator to tune the frequency at which the negotiation occurs. The delay between RA's in edge networks can also be adjusted for the same purpose. This means the negotiation can occur as often as the network administrators deem necessary.

III. NP++ APPLICATIONS

There are many scenarios in which NP++ can be used to dynamically change the characteristics of a protocol to optimise its transmission for a particular use. The following sections describe some of these in more detail to give an idea of how the flexibility NP++ provides can be utilised to optimise network link performance.

A. Header Compression

Header compression is often a desirable property for networks that have a relatively low level of available bandwidth such as wireless networks. Header compression techniques [4][6][7] are used to reduce the size of headers transmitted over the network. This can be accomplished as many fields in an IPv6 header (such as the source/destination addresses, port numbers etc) stay the same across packets in the same flow. By only periodically sending full packet headers, the values that seldom change can be cached at the receiver. A context identifier is sent in place of the static fields in the header and the receiver can use this to replace the fields and rebuild the full packet header before forwarding the packet.

Using NP++, a wireless link could give a header compression mapping a high priority, and then as long as the wireless router and the host both support the mapping, compressed headers could be sent over the link. Of course if neither node supports a header compression mapping, they can still communicate using the default mapping (i.e., non-compressed headers) which should yield the same performance as standard IPv6.

B. Address Translation

Address translation is another feature that is often desirable. Traditionally address translation is used on Local Area Networks where only 1 host (such as a gateway router) needs to be visible to the outside world. All the other hosts on the network can then use a private, non-routeable address that gets rewritten

as it passes through the gateway. This method has obvious and well-known drawbacks such as the internal hosts not being externally addressable from the Internet, requires knowledge of application addressing and behavior by a network device, etc. The most commonly used form of address translation is Network Address Translation (NAT) [1]. NAT is used extensively at the edges of the Internet. It is especially popular in homes and small businesses to increase the number of hosts that can be used on a single connection such as ADSL.

With IPv6 the need for Address Translation in the traditional sense is removed due to the massively increased address space. However there are specific scenarios for which it is desirable in a different form. An example of this is described in Section III-E.

C. Security

Encryption is often desirable from a user perspective for certain types of traffic flows as it makes it more difficult for someone listening in on the transmission to interpret the data. The traditional method of doing this is using IPSec [5]. One problem with IPSec is that it encrypts the packet headers. This means that systems such as NAT are not able to rewrite the addresses in the headers, so IPSec and NAT cannot both be used on a network together in their standard forms.

NP++ allows an encryption mapping to be more easily used across a link. It also allows combining address translation and encryption into a single mapping if desired. This is a good example of how incompatibilities between technologies can be overcome by using NP++. A single mapping can be created that combines the functionality of a number of technologies allowing them to co-exist on a network. The desired functionality can be built into the mapping so the technologies can co-exist within a single physical specification.

D. Label Switching

High bandwidth networks at the core of the Internet suffer from different bottlenecks than most edge networks. The sheer amount of traffic the routers have to forward mean that route lookups for each packet can prove to be very costly in terms of the amount of processing power they require. Adding a label to the front of each packet allows routers to create a kind of "virtual flow" across a number of hops. By doing this, a router only needs to inspect the label attached to the packet as opposed to doing a full longest-prefix match on the destination IP address to route a packet. This allows the forwarding tables to be substantially smaller more easily searchable than traditional full IP routing tables. These factors combined can substantially reduce the amount of time required to perform a routing lookup. The most commonly used form of label switching is Multi Protocol Label Switching (MPLS) [3].

Providing label switching functionality often requires the use of specialised routers known as Label Switch Routers (LSR's). This means having to deploy networks to take the position and connectivity of the LSR's into account. In NP++, these considerations would not have to be taken into account. When a network is deployed, a normal router could simply

switch to using a MPLS style physical mapping. This allows any network to be easily converted into a label switched network without costly changes to the network infrastructure.

E. Sensor Networks

Wireless sensor networks (WSN's) often use custom network protocols to allow the individual sensors to communicate with each other. These protocols are often specifically designed for use in dynamic, resource constrained environments. This becomes a problem when it is desirable for information to be exchanged between the WSN and users connected to the Internet. The traditional approach to this problem has been the use of specialised gateways such as in the Arch Rock Primer Pack [8]. These allow the WSN to interface with a traditional IP based network.

Despite the use of a gateway to facilitate inter-network communication, developers of end-to-end WSN applications are still faced with a more daunting task when creating their applications. This is because applications running on the sensor networks have to be constructed to communicate using the custom WSN protocol, whereas the application written for a host connected to the Internet has to be written to conform to more conventional Internet protocols such as IPv4 or IPv6.

NP++ can be used in this scenario to enable application developers to write both applications conforming to the same network interface while still having the differing underlying network functionality for the various network technologies used on the links [9]. By creating a special physical mapping for the WSN, the logical specification stays the same end-to-end but the way the data is transmitted across the network can be adjusted to suit the underlying network technology.

By using NP++ within sensor networks an important advantage is gained. As NP++ takes care of rewriting all the packets in and out of the WSN for transmission/receipt across the Internet the nodes on the WSN can be externally addressed. Many sensor networks only allow the nodes to transmit data to a remote monitoring station whereas, by using NP++, the nodes can all be addressed directly via an IPv6 address which gets transparently re-written to the sensor networks own addressing scheme by the physical mapping. This allows Internet users to request data directly from application on sensor network nodes rather than relying on periodic updates from gateways. This is a good example of a modern form of address translation built into a mapping alongside other packet header manipulation.

F. IPv4

NP++ can also be used as a method of IPv4 - IPv6 transitioning. By using an IPv4 physical mapping an NP++ host is quickly transformed into an IPv4 to IPv6 gateway. This allows networks using any physical mapping to communicate via IPv4 networks and is also an aid in the incremental roll out of any new protocol.

G. NP++ Applications Summary

The applications listed in the previous sections are a sample of the many possibilities that could be created to optimise

links. The fact that these can be deployed and selected dynamically at run time makes NP++ extremely flexible and an ideal solution for supporting many network and application demands through the use of physical mappings.

NP++ is backwardly compatible with both IPv4 and IPv6. Whilst providing the possibility to optimise data transmission in whatever way desirable for specific network/application scenarios NP++ is still able to communicate over existing infrastructures such as the Internet.

IV. NP++ IMPLEMENTATION

A full implementation of NP++ is publicly available in the form of a Kernel module for Linux Kernel Version 2.6.

The implementation takes advantage of the NETFILTER [10] system present in the Kernel. This allows IPv6 packets to be intercepted just before they leave and just after they enter a host. This allows the mapping to be applied to the packets and applications can communicate using normal IPv6 sockets unaware that the packets are being optimised in the network.

It is possible to set the mapping being used at any time through the use of the proc file system [11]. Several entries are created by the NP++ code to allow setting the mapping used on a host. This allows the user space applications that have been created to easily interact with the kernel module to set the mappings based on the outcome of the auto negotiation.

A. Physical Mappings API

To enable developers to easily develop for the NP++ protocol a physical mapping API has been created. The API exports a number of functions that developers can use to integrate their own mappings. The API also means that the NP++ code can easily recognise new mappings and pass the appropriate packets on to them for processing. Physical mappings are implemented as Linux Kernel modules and have access to the API functions exported by the main NP++ module as long as it has included the *np-mod.h* header file.

1) *The np_sw Struct:* The *np_sw* structure holds all of the information about a mapping that the NP++ code requires. The definition of the struct along with a short description of each of its fields can be seen in Code Listing 1.

```

1 struct np_sw
2 {
3     struct list_head npswl; // Used Internally
4     u_char phy_id; // ID of the mapping
5     const char *desc; // Textual Description
6     // Input Function
7     int (*phy_input) (struct sk_buff **, const struct net_device *);
8     // Output function
9     struct sk_buff *(*phy_output) (const struct net_device *, struct sk_buff *);
10    void (*phy_slowtimo) (void); // Timeout Function
11    /* Counters */
12    long p_sent; //Packets Sent
13    long p_rcvd; //Packets Received
14    long p_in_dropped; //Incoming Packets Dropped
15    long p_out_dropped; //Outgoing Pacjets Dropped
16 };

```

Listing 1. NP++ Physical Mapping Struct *np_sw*

A physical mapping module must create a *np_sw* struct and to describe itself. It is then able to call the *np_mapping_add()* function. This function takes care of inserting the new mapping into the NP++ architecture when passed a *mp_sw* struct. The functions pointed to by *phy_input* and *phy_output* can be used

by the mapping to transform the packet in whatever way the developer requires. The *phy_slowtimo* function pointer allows a developer to provide a function in their mapping that will be called every 500ms. Periodic functions like this are often useful in protocol development and as such NP++ provides this functionality in mappings.

Because the mappings are implemented as modules they can easily be inserted and removed from the Kernel at run time. Obviously when a mapping module is removed it would leave its state behind in the NP++ architecture. For this purpose the *np_mapping_del()* function can be called, passing the the *np_sw* struct as a function argument. This function removes all the state within the NP++ architecture and safely removes the mapping.

B. Router Advertisement Modifications

As described in Section II-A.1 an extra Router Advertisement option has to be implemented to disseminate the list of a router's available physical mappings. In Linux, RA's are distributed by an application called *radvd*. Modifications have been made to *radvd* to allow it to distribute a list of physical mappings and their priorities. A configuration option has been added to enable the new RA option to be enabled/disabled. When enabled *radvd* will distribute the option containing all supported physical mappings.

The *radvd* configuration file is divided up into interface definitions that describe what RAs should be sent over each interface. The NP++ RA option can be enabled on a per-interface basis. For each interface using the NP++ option, a number of physical mappings can be defined. An example of some of these definitions is depicted in Listing 2.

```

NpMapping DEFAULT/2
{
    Advertise on;
};
NpMapping HEADER.COMPRESSION/8
{
    Advertise on;
};

```

Listing 2. NP++ Radvd Config Physical Mapping Definitions *np_sw*

Every mapping to be advertised has to have its own NpMapping section defined in the *radvd* configuration file. The section consists of the name of the mapping along with the priority following a forward slash. Inside this section is the *Advertise* keyword which can be followed by either *on* or *off* which indicates whether the mapping is to be sent out in the RA or not. In Figure 2 two mappings are being advertised. The DEFAULT mapping is being advertised with a priority of 2 and the HEADER.COMPRESSION mapping is being advertised with a priority of 8. By configuring the mappings in the *radvd* configuration file in this way it is simple to disseminate the mapping information across network links to enable the negotiation process.

C. Mapping Advertisement Application

When the Neighbour Discovery method of advertising specification maps is being used the MAPping ADvertisement (MAPAD) application is employed. This application bundles the MA option into a NS message and transmits it to

neighbours advertising its specification map. MAPAD uses a configuration file such as the one shown in Listing 3 for defining the specification map to advertise along with other general configuration options.

```

1 Mapping DEFAULT Priority 100
2 Mapping HEADER_COMPRESSION Priority 20
3
4 Interface eth1
5 LLAddress fe80::250:56ff:fec0:8/64

```

Listing 3. Example MAPAD Configuration File For Use With NP++

MAPAD also has a listening socket that detects when a MF has been received in an NA. When a MF is detected MAPAD is responsible for setting the mapping on the host based on the finalised mapping received.

D. Host Negotiation Application

The host negotiation application runs on any system that is connecting to a network. This could be an end-host or a router. The application opens a socket that listens specifically for ICMPv6 messages. When one is received, it checks to see if the packet contains an RA or NS and proceeds to process it if it does, or discard it if not.

If an MA is detected the negotiation application applies the consensus algorithm described in Section II-A.2 using its own specification map and the specification map contained in the MA. After it has reached a decision on the mapping to use it constructs an MF and encapsulates it in either a RS or an NA depending on whether the Router Advertisement or Neighbour Discovery mapping advertisement system is being used. This packet is then transmitted back to the advertising host and both hosts can communicate using the newly negotiated mapping.

After the MF has been sent back to the advertising host, the receiving host sets its physical mapping using the proc interface provided by the kernel. The router will also set its physical mapping in the same way once the message has been received and processed.

V. RELATED WORK

Solis and Obraczka [12] designed the *Flexible Interconnecting Protocol*, or FLIP, whose main goal is to accommodate devices with varying power, processing, and communication capabilities. This is the only previously published work that we are aware of which is similar to NP++.

FLIP allows application programmers to define the composition of headers by selecting which fields can be transmitted. It uses a meta-header in each packet to indicate which header fields are present. The design is efficient but only supports field suppression. Unlike NP++, this approach is unable to provide features like label switching, field re-arrangement, compression of complete header chains, and different levels of error detection and correction.

FLIP also allows the user to select the range of protocol functionality to be provided, starting from basic network-level functionality to a full transport protocol like TCP. Compared to NP++, FLIP has a more device-centric approach to adaptability. Functionality can be enabled or disabled to save resources

such as CPU cycles or battery power. NP++ has a network-centric approach with a more flexible network adaptability. We believe that device adaptability can be achieved by providing more variety of protocols over IP or NP++.

VI. CONCLUSION

This paper describes some of the main features of the NP++ protocol, which provides a highly flexible internetworking platform to support the diverse range of networking technologies available both now and in the future. It accomplishes this by using a level of indirection, something new in protocol header design. This design uses a *logical* and *physical* specification of the header. By doing this the underlying characteristics of the protocol can change on a hop-by-hop basis depending on any number of factors. Examples of such factors include the underlying network technology or application demands on the network. The physical specifications are based on proven concepts such as label switching, field re-arrangement and header compression, but the concept of indirection in protocol design is novel. The fact that a new physical mapping can be deployed over each link means that the mapping can be optimised for transmission over future networking technologies with a minimum of new protocol specification as well as requiring no changes to existing applications.

This paper has also demonstrated the functionality of the working NP++ prototype implementation that is publicly available. By deploying NP++, it is much simpler to develop and test protocols without having to modify applications. It also allows development networks to interconnect via the Internet for large scale tests without requiring special gateways.

REFERENCES

- [1] K. Egevang and P. Francis. RFC 1631: The IP network address translator (NAT), May 1994. Status: INFORMATIONAL.
- [2] T. Nishitani and S. Miyakawa. Carrier grade network address translator (nat) behavioral requirements for unicast udp, tcp and icmp, July 2008.
- [3] Eric C. Rosen, Arun Viswanathan, and Ross Callon. Multiprotocol label switching architecture. Internet RFC 3031, January 2001.
- [4] Mikael Degermark, Mathias Engan, Björn Nordgren, and Stephen Pink. Low-loss TCP/IP header compression for wireless networks. 1996.
- [5] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401 (Proposed Standard), November 1998. Obsoleted by RFC 4301, updated by RFC 3168.
- [6] M. Degermark, B. Nordgren, and S. Pink. RFC 2507: IP header compression, February 1999.
- [7] C. Bormann, C. Burmeister, M. Degermark, H. Fukushima, H. Hannu, L-E. Jonsson, R. Hakenberg, T. Koren, K. Le, Z. Liu, A. Martensson, A. Miyazaki, K. Svanbro, T. Wiebke, T. Yoshimura, and H. Zheng. Robust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed. RFC 3095, July 2001.
- [8] Arch Rock. Primer pack. http://www.archrock.com/products/primer_pack_ip.php.
- [9] Matthew Jakeman, Danny Hughes, Geoff Coulson, Gordon S. Blair, Steve Pink, and Kevin Lee. Supporting IPv6 interaction with wireless sensor networks using NP++. In Yingshu Li, Dung T. Huynh, Sajal K. Das, and Ding-Zhu Du, editors, WASA, volume 5258 of *Lecture Notes in Computer Science*, pages 409–419. Springer, 2008.
- [10] The netfilter.org Project. Netfilter, <http://www.netfilter.org>.
- [11] J. Nerin T. Bowden, B. Bauer. The /proc filesystem, November 2000.
- [12] I. Solis and K. Obraczka. Flip: a flexible protocol for efficient communication between heterogeneous devices, 2001.