



# Intel<sup>®</sup> Technology Journal

Communications Processing

## Distributed Control Plane Architecture for Network Elements

# Distributed Control Plane Architecture for Network Elements

Manasi Deval, Intel Labs, Intel Corporation  
Hormuzd Khosravi, Intel Labs, Intel Corporation  
Rajeev Muralidhar, Intel Labs, Intel Corporation  
Suhail Ahmed, Intel Labs, Intel Corporation  
Sanjay Bakshi, Intel Labs, Intel Corporation  
Raj Yavatkar, Intel Communications Group, Intel Corporation

Index words: control plane protocols, network element, scalability, high availability, network processor

## ABSTRACT

With the explosion in the use of the Internet for e-business, entertainment, and wireless communication, Internet control and routing infrastructure are facing limitations in terms of the increasing scale. Because the Internet is now mission-critical to many industries, providing high availability (HA) is a major concern.

In this paper, we discuss the scaling limitations of the routing and signaling protocols used in the Internet and propose a novel method to distribute the functionality of control plane protocols implemented in network elements such as switches and IP routers. The proposed method relieves the scalability bottlenecks in various signaling and routing protocols by selectively off-loading certain periodic operations of the protocols to the data plane processing components within a network element. This leads to scalable, highly available, and robust signaling and routing protocol implementations. We use the popular routing protocol, Open Shortest Path First (OSPF) as an example to discuss the appropriate metrics for evaluating the performance of an OSPF implementation, and then we identify the bottlenecks in scaling-up an OSPF implementation. We also describe a generic method for distributing the functionality of routing and signaling protocols, such as OSPF, across both control and data planes that makes them (a) more scalable, (b) more resilient in the presence of faults, and (c) amenable to faster convergence in the face of link or node failures.

## INTRODUCTION

Imagine waiting at a 4-way stop sign at a crowded intersection and wondering if a traffic signal would be more efficient. The data and communication traffic on the Internet is directed and controlled by network

elements such as IP routers or switches that operate like stop signs and traffic signals. The engineering and design of these network elements is akin to managing traffic at a busy intersection.

As vehicular traffic increases, the roads have to be widened, additional traffic signals have to be added, and the efficiency of signals has to be improved to provide better traffic management. Network traffic, too, is constantly growing because newer services are constantly being deployed to attract more consumers. Newer services lead to an increase in both control and data traffic. Similarly, in order to support the increasing amount of network traffic, the network elements have to support a growing number of physical interfaces leading to a higher density of network elements in any given area (i.e., ISP networks).

Current network elements, such as an IP router, a Layer 3 switch, or a 3G Radio Network Controller, consist of three operational planes: control, forwarding, and management. The control plane executes various signaling, routing, and other control protocols in addition to configuring the forwarding plane. The forwarding plane performs the packet processing operations such as IP forwarding and classification, at or close to the line-rate. Typically, the forwarding plane consists of specialized ASICs or programmable network processors that implement per-packet operations. The control plane, in contrast, typically executes on a centralized, general-purpose processor. The management plane, which provides an administrative interface into the overall system, consists of both software executing on a general-purpose processor as well as probes and counters in the hardware. For example, in an IP router, the control plane executes control protocols such as OSPF [6], RIP [7], RSVP [8], RSVP-TE [17], etc. These control plane protocols exchange updates with their peers across the

network and generate control information such as routing tables or label tables, needed by the forwarding plane to actually forward data packets. There may be multiple forwarding planes to handle a large number of interfaces.

A typical network element in the current ISP network has hundreds of physical or virtual interfaces. There may be 300-500 network elements in an ISP network. Such large numbers of interfaces per network element, when coupled with an increase in the volume and variety of control traffic, limit the scalability of the control plane in terms of its ability to quickly handle control plane messages or to react in a timely manner to events such as link failures. These problems are further compounded when a network element occupies a critical position in the network topology where the reliability of such a node is critical to the operation of the rest of the network.

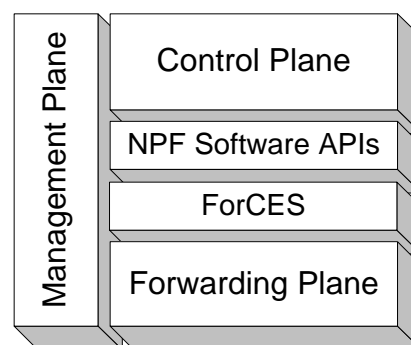
In this paper, we focus on improving the scalability of the routing and signaling protocols in terms of the ability to handle hundreds of interfaces and the associated large volume of control protocol messages and events. For the purposes of this paper, the term *protocol* means a routing or signaling control protocol unless specified otherwise.

We first describe the evolution of network elements and the challenges in designing a control plane. We then discuss some concepts in protocol distribution that are further developed into a Distributed Control Plane (DCP) architecture. The DCP architecture is a distributed software framework that can be used to distribute some of the control plane protocol functionality onto the forwarding planes, thus providing a scalable solution for protocols on the network element. As we will see, this distribution is easily possible and adaptable for forwarding planes that are built using programmable network processors. And finally, we describe a case study of distributing the functionality of the OSPF protocol in the context of the DCP architecture.

## EVOLUTION OF NETWORK ELEMENTS

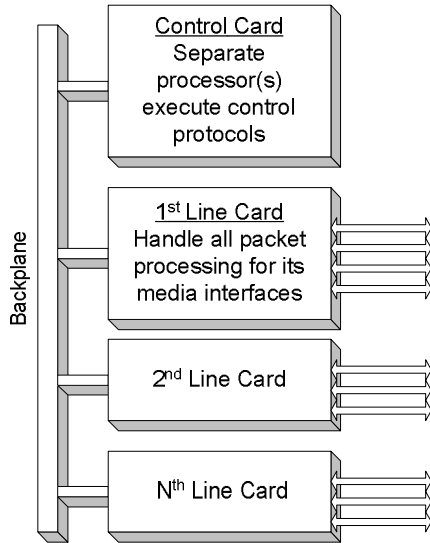
Network element architecture, such as the design of an IP router, has evolved more or less in three generations. In the first generation, control and forwarding planes were executed on a common processing resource. For example, several of Cisco's earlier routers used a common Motorola 68000 for both planes. Until very recently, the network element architecture was in the second generation wherein the control and forwarding planes were explicitly separated so that they could be executed on separate processing resources. But these network elements used proprietary interfaces for separation between control and forwarding planes, and they used ASICs for processing in the forwarding plane. However, with the advent of silicon such as network

processors [21] and other specialty hardware, came modular component architecture. In such an architecture, network elements could be built using off-the-shelf components from different vendors, and, therefore, a more formal and standardized definition of the interface between control and forwarding planes was required. The network elements that are adopting this standardized and open interface between control and forwarding planes constitute the third or the current generation of the network element architecture. As shown in [Figure 1](#), this separation is done by using open APIs, e.g., those standardized by the NPF [15] and the IETF's ForCES [16], between the control and forwarding planes. The NPF APIs provide a standardized programming interface between control and forwarding planes. The control protocols in the control plane interact with corresponding components in the forwarding plane using these APIs. For example, IPv4 routing protocols in the control plane manage the entries in the IPv4 forwarding table in the forwarding plane via the NPF IPv4 APIs. The IETF is in the process of defining the ForCES protocol that uses a logical model for representing the forwarding plane entities and will provide a protocol to communicate the control data associated with these entities from control to forwarding plane. The protocol will be independent of the forwarding plane data model.



**Figure 1: Networking architectural model with separated control and forwarding planes**

[Figure 2](#) shows the architecture of a typical second- or third-generation network element. It consists of a control plane (or control card) and multiple forwarding planes (or line cards). All the control plane processing is done in a *centralized* manner by the control card while the entire media-specific processing and line-rate packet processing is done by the line cards.



**Figure 2: Current architecture of network elements**

Typically a backplane or an interconnect based on ATM, Ethernet, CSIX, or some proprietary switching fabric, connects all the blades and is used to transfer both control and data traffic between the blades. Such an architecture has two key advantages, namely, the processing resources of control and forwarding planes are separated, allowing each to scale independently, and components from different vendors can be used to build each plane because a standardized inter-plane interface is used.

## CONTROL PLANE DESIGN CHALLENGES

As the popularity of the Internet has grown, the link bandwidths have grown from megabit rates to gigabit rates, and new protocols have been introduced to support new services being deployed on the Internet.

In response to the growing end-to-end traffic, a typical ISP network now has as many as 300 to 500 network elements (IP routers). Depending upon its location in the network and the services it needs to support, each network element typically runs a wide variety of protocols such as Border Gateway Protocol Version 4 [9], Intermediate System-to-Intermediate System [10], Open Shortest Path First [6], Routing Information Protocol [7], Internet Group Management Protocol [11], Distance Vector Multicast Routing Protocol [12], and RSVP [8]. For example, a typical Digital Subscriber Line Access Multiplexer (DSLAM) runs a complete IP stack and an ATM stack, where each stack is made up of a number of protocols. Moreover, the network elements such as IP routers and Layer 3 switches may contain hundreds of physical or virtual interfaces. As a result, an IP router

may have to exchange control information with hundreds of peers. Such a growth in bandwidth, network traffic, and network element density impacts various aspects of the operation of a protocol.

## Scalability

As the number of interfaces in a network element increases, the protocol generates more control traffic because it has to communicate with an increasing number of peers via many different interfaces. As newer services are introduced in the network, they either add new types of control traffic to the existing control protocols or introduce new control protocols. Either way it leads to increased complexity and traffic volume in the control plane. In the past, control traffic only accounted for about 2% of traffic in the packet networks, and about 5-15% in the telecom networks. However, with the increase in the number of interfaces and user services supported, this control traffic in packet networks is growing to 5-10% of the total traffic and is expected to be even larger in the telecom networks. For example, consider OSPF [6], a key routing protocol used in most of the ISP networks today. The OSPF protocol periodically exchanges HELLO messages with its directly attached peers. As the number of interfaces or peers increases, the protocol must send, receive, and process a much larger number of such messages. In addition, the OSPF protocol has recently been extended to *OSPF-TE* [19] that supports traffic engineering of the entire network. Such extensions add to the type and number of control messages handled by an implementation of the OSPF protocol.

Contributing to this growth in volume is the per-user state maintained at some network elements. Some protocols, like RNC in the 3G cellular network, establish and manage connections for individual mobile users. Typically, an RNC handles 300,000 to 400,000 users with a projected growth to 500,000 users. Also, with the growing use of VPNs and MPLS, the level of granularity at which forwarding is done in IP routers is moving closer to per-client network level.

All the factors described above contribute to a substantial increase in demand for control plane processing cycles or processors.

## Highly Available Protocols

With the growing use of real-time services such as Internet Telephony and video distribution, loss of connectivity for even a second may result in a considerable loss of information and affect the user experience. So network faults need to be isolated and repaired rapidly.

As there are numerous protocols such as OSPF and BGP executing on the control plane, a link event such as link going up or down, typically triggers processing on all or most of them, causing a huge increase in processing requirements, which may saturate the control plane processor and render a network element non-operational. For example, a link going down in an IP router running BGP [9] would mean that BGP would need to update its internal protocol state machine and send appropriate messages to its peers. In addition, it would need to update thousands of routes on its forwarding plane. For a network element to be highly available, not only does it have to react to faults quickly, but it also has to scale up its processing capacity to handle the extra processing needed in the case of faults.

#### **Robustness to Control Plane Denial of Service Attacks**

Malicious applications or viruses may launch a Denial of Service (DoS) attack on network elements, so the protocols running on them must be robust enough to operate in spite of such attacks. Many protocols use an authentication mechanism to ensure that the packets received are from valid nodes. An example of a typical DoS attack on a control plane is flooding the control plane with spoofed BGP [9] control packets. BGP packets must typically be authenticated before they are processed. Traditionally, the forwarding plane sends all control packets to the control plane for authentication and processing. If there is a significant increase in the spoofed traffic, the control plane would be flooded with authentication requests causing interference with other control plane processing.

### **PROTOCOL DISTRIBUTION: CONCEPTS**

All of the problems described in the previous section can be alleviated if the control plane processing could be performed in a distributed fashion, such as taking some of the functions and simply distributing them to individual line cards. Such a distribution can deal better with Denial of Service attacks, because processing is now closer to the point of attack and can therefore react quicker and sooner.

We need to bring together the following three elements to achieve distribution of the functionality of a control protocol:

- First, evaluate the internals of a control protocol to identify frequently performed functions that will benefit from offloading to separate processing resources.
- Second, utilize a multi-processor network element hardware platform that provides several options in terms of where a protocol function can be executed.

- Third, define a software architecture and associated infrastructure that enables the creation of a distributed implementation of any protocol.

### **Evaluating a Control Protocol for Distribution**

There are two main issues to be considered while intelligently distributing control processing functions: determining the function(s) of the protocol that can actually be distributed without affecting the correctness of a protocol, and deciding which parts of a network element can host such distributed functions.

#### **What to Distribute? Anatomy of a Protocol**

Operations of any control protocol like BGP, OSPF, or RSVP, etc. can be split into three main categories: (1) link-specific functions involving packet forwarding and maintenance of connectivity with protocol peers, (2) protocol-specific processing, such as maintaining the protocol state machine, and (3) functions that update the forwarding plane with the control information.

1. Functions that fall into the category of link-specific functions or independent functions involve parts of a protocol responsible for communication with its protocol peer(s) to maintain the adjacency and status of the connectivity, and parts of a protocol that involve processing of packets on a per-interface basis (for instance, filtering incoming protocol updates from specific protocol peers). These functions are ideal candidates for distribution to processors on line cards. Such distribution helps reduce the computation load on the control processor(s) and bandwidth load on the interconnect fabric, and it also accelerates peer discovery and failure detection.
2. Functions that fall into the category of protocol processing functions are mainly related to computation, based on updates received from the protocol peers to generate new peer updates and new control information, such as new IP routes for installation in the forwarding plane. The protocol state machine maintenance functions belong to this category. These functions need to be considered on a case-by-case basis for distribution, with the focus mainly on reducing the computation load on the control processor(s) when network failures cause additional message traffic.
3. The forwarding plane update or centralized functions consist of updates to refresh the control information in the forwarding plane—for example, route table updates. These functions can rarely be distributed.

### Where to Distribute? Types of Protocol Distribution

There are mainly two ways of distributing control plane functionality:

1. *Functional Distribution.* A control plane protocol consists of many distinct components that implement different protocol functions. In the case where a distinct component is independent of other components, it can be cleanly separated from the rest of the protocol functions and be implemented elsewhere. For example, the HELLO messages in the OSPF protocol perform the task of keeping track of the availability of peer nodes or the liveness of adjacent links. Such a function is an attractive target for distribution as it can be cleanly separated from the rest of the OSPF protocol functionality and tends to require frequent communication and processing.

There are two drivers for functional distribution, (1) the need to separate functions of a protocol that compete for a significant portion of the control CPU capacity, and (2) the desire to exploit additional benefits expected in executing a particular function at a certain location within a system. For example, functions involving physical interface state should be executed on a line card so that the particular function can scale well as the number of physical interfaces increases. The primary goals of functional distribution are to enhance the scalability of a control protocol and make it more responsive to external events by executing certain functions closer to their domain of activity or influence. This type of distribution requires multiple processors in a system. In this paper, we focus on using this distribution method for implementing scalable and robust control planes.

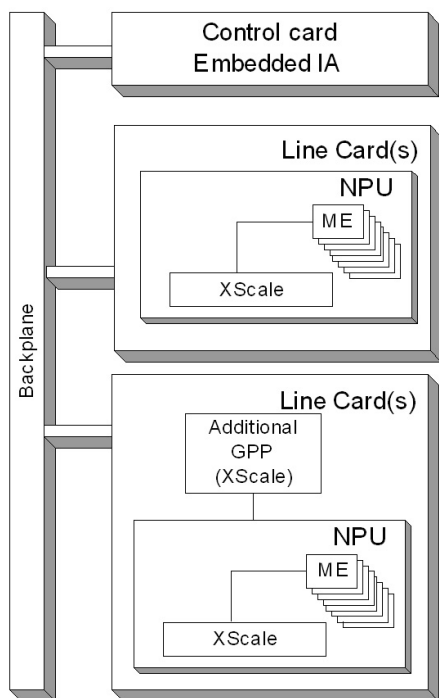
2. *Layered Distribution.* Every network protocol stack consists of multiple layers wherein each layer is responsible for a distinct function and processing of both incoming and outgoing packets. For example, a Node B Application Part (NBAP) layer [20] in a Radio Network Controller (RNC) stack relies on lower protocol layers such as *Iub Frame Protocol*, followed by the UDP/IP layers, and finally, the PPP layer. Because these layers are distinct, it is sometimes useful to perform functions related to one or more of these layers in a separate location to take advantage of the available processing resources. (Even though this is an alternate but valuable way of distributing the control plane functionality, we have chosen to focus on the functional distribution in this paper; we hope, however, to address this method in our future work.)

## HARDWARE PLATFORM ARCHITECTURE FOR DISTRIBUTED NETWORK ELEMENTS

Based on the discussion in the previous section, following are the high-level requirements for the hardware platform needed to implement functional distribution within control protocols:

1. It should support multiple processors where control functions can be executed. Typically these processors are either present on line cards and optionally on cards hosting management or other applications.
2. It should be possible to seamlessly scale the compute cycles available for control processing. Typically this is done by adding new line cards or application cards to the network element whenever needed.
3. It should be possible to distribute control protocol functions to an appropriate processor within the system.

As explained in the previous section, there are three classes of protocol functions: Class 1 is link specific (possibly communication or IO intensive), Class 2 is protocol specific, processing or compute intensive, and Class 3 is centralized. Thus, in order to implement functional distribution for such control functions, a hardware platform with three processing levels or tiers would be needed. The lowest or first processing level (for example, processors on line cards) could execute Class 1 and optionally Class 2 control functions. The second processing level (for example, processors on application cards) could execute Class 2 and optionally Class 1 control functions. The third tier (for example, the traditional control card) could execute Class 3 and optionally Class 1/Class 2 control functions. The middle or second tier is the optional processing level and may not exist in all systems. It may also not be required if other processing systems are deemed sufficient. This implies that the hardware platform must at least consist of the first and the third processing tiers. Correspondingly, Intel has introduced a multi-tiered hardware platform of network elements as shown in [Figure 3](#) and [Figure 4](#).



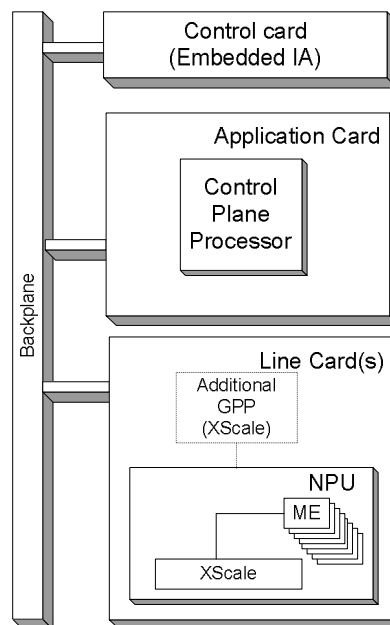
**Figure 3: Two-tier network element hardware platform**

Figure 3 shows a two-tier hardware platform consisting of a control card built around an embedded IA processor, targeted for compute-intensive tasks (such as execution of control stacks, database lookups, etc.) and multiple line cards built around IXP network processor(s) targeted for communication-intensive tasks (such as wire-speed packet routing, forwarding, etc.). The IXP network processor constitutes multiple microengines and one core processor based on Intel XScale® technology. For example, the IXP2400 has eight microengines [13]. There are two variants of the line cards existing today, namely, line cards with one or more IXPs, and line cards with an additional standalone general-purpose processor (GPP) (e.g., the Intel XScale). Besides the processor on the control card in such a hardware platform, control functions can be distributed to and executed on one or more of the IXPs or on a standalone GPP core, as the case may be. Thus, such hardware platforms provide additional compute cycles for control plane processing whenever a new line card is added to the network element, which is a basic premise for constructing scalable network elements. As the control processing functions can be distributed across processors present in line cards, the computation bottlenecks that exist in the centralized control card are either minimized or

® Intel XScale is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

removed. In addition to this, the system’s response time to external events is reduced by distributing functions closer to wire on line cards. For example, line cards built using the IXP2800 [21] have support for accelerating compute-intensive cryptographic processing. This feature could be used to delegate cryptographic functions within a control protocol to the IXP2800. Lastly, it is quite possible that some network elements may have two control cards for high-availability purposes wherein one control card is active and the other acts as a standby. We are not proposing to use the standby card for offloading purposes.

In order to augment the two-tiered hardware platform further, Intel has also introduced a new family of control plane processors [14]. These processors can be used to add another tier to the two-tiered hardware platform as shown in Figure 4.



**Figure 4: Three-tier network element hardware platform**

The control plane processors are targeted for certain compute-intensive communication problems (e.g., deep packet inspection, encryption, etc.) and for compute-only problems (e.g., executing certain control stacks). Control plane processors have features such as low power, low heat dissipation, and smaller die size that make them ideal for the implementation of application cards. Another possible configuration for this hardware platform is to have the control plane processor co-exist with the IXP network processor(s) on the line cards instead of some other GPP.

## DISTRIBUTED CONTROL PLANE (DCP) ARCHITECTURE

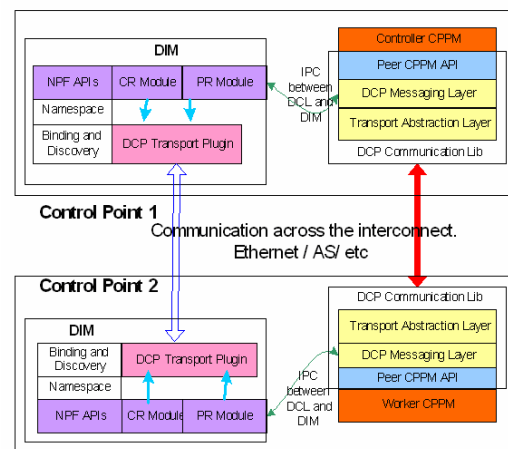
In this section we describe the Distributed Control Plane (Software) Architecture and see how it supports functional distribution of protocol implementations. Some of the key design considerations for this architecture are described below:

1. *Protocol independence.* The architecture should provide the primitives that are common across all kinds of control protocols.
2. *Pluggable architecture.* The architecture should provide well-defined interfaces so that a new protocol implementation can be cleanly plugged in. The protocols use these interfaces to discover processing target capabilities and exchange related protocol-specific information between the components of an implementation.
3. *Multiple processor hierarchies.* The architecture should be able to scale to multiple processor resources either on different processing targets, on the same processing target, or on the same processor core, as in the case of multi-core CPUs.
4. *Interconnect independence.* The architecture should provide an interconnect-agnostic interface to communication protocols. Depending upon the system configuration, two control protocol functions may communicate over different types of interconnects.

A distributed network element has one or more control cards and multiple forwarding plane blades. The DCP architecture consists of two major components: the *DCP Infrastructure Module* and the *DCP Communication Library*. A Control Point (C-Pt) is defined as a control or forwarding plane executing a DCP Infrastructure Module (DIM) along with one or more routing or signaling protocol functions. The DIM maintains the current view of the network element, which entails keeping track of the capabilities and resources of other C-Pts in the network element and the Control Plane Protocol Modules (CPPMs) running on each of them. A protocol implementation that executes a single function or the complete functionality of a protocol as per an RFC or an industry standard is known as a CPPM. Multiple CPPMs work in conjunction to provide a complete implementation of the protocol. The protocol function that has been separated out of a core protocol implementation forms the *Worker CPPM* while the core functionality of the protocol is known as the *Controller CPPM*.

The Distributed Communication Library (DCL) is linked with the CPPM and is responsible for transparently providing communication between the CPPMs. In our implementation, the DIM and CPPM run in different processes. The DCL communicates with the DIM to discover the peer CPPMs, and it provides an abstraction of the peer CPPMs. The common interface between peer CPPMs is known as the CPPM Peer Interface.

[Figure 5](#) shows the operation of a distributed protocol on a DCP-enabled Network Element. Each C-Pt has a DIM and a CPPM running on it. The control points and the CPPMs are connected over the same interconnect.



**Figure 5: Operation of distributed protocol on a DCP-enabled network element**

### DCP Infrastructure Module

The DIM contains the logic required by a C-Pt to discover other C-Pts present in a system, establish connectivity with them, and exchange information about the C-Pt capabilities (such as a list of CPPMs registered and their capabilities). The DIM is configured with a policy to allow a C-Pt to communicate with selected C-Pts.

As seen in Figure 5, the DIM is made up of the namespace, Binding and Discovery module, CPPM Registration (CR) module, and the Packet Redirection (PR) module. It also provides a standardized interface to protocol stacks via the NPF [15] APIs. The namespace maintains a logical view of the network element and its components. Various modules register with the namespace to receive appropriate events when there are changes to the namespace. The Binding and Discovery module discovers other C-Pts, exchanges configuration information with them, and then updates the namespace with this information. The information exchanged

includes the properties of control points, and the capabilities of the CPPMs running on them.

The CPPM Registration (CR) module allows a CPPM to register its capabilities and get information about its peers. A CPPM uses the Packet Redirection (PR) module to specify the protocol packets of interest to itself. If no such filters are set, the Controller CPPM receives all protocol packets by default.

The DCP Transport Plug-in is responsible for communication between the DIMs. This layer hides the specifics of the physical interconnect and associated transport protocol that are used to exchange information between two DIMs. It provides a connection-oriented interface to set up connections and transfer data between any two DIMs in the system and takes care of managing underlying transport protocols and interconnects. New plug-ins can be easily added to support additional interconnects and associated transport protocols.

### **DCP Communication Library**

The DCP Communication Library (DCL) provides an abstraction of the peer CPPMs in the form of the resources they own or control. It is primarily responsible for transparent communication between the CPPMs and for implementing advanced policies for load-balancing and fast-failover. At initialization, the DCL is configured with CPPM capabilities, such as CPPM, CPPM function, resources, and the interconnect. The implementation of the DCL is independent of the CPPM function or protocol that it is linked to. The operation of this library is akin to an intelligent RPC mechanism.

The DCP messaging layer controls the actual communication between the CPPMs across the Peer CPPM API. It provides functions to serialize and de-serialize data, and transparently exchange it between the associated CPPMs over the network. This layer uses information from the CRM to set up and manage the communication channels among the associated CPPMs. This layer can detect CPPM and connection failures and is informed about the changes in the configuration of peer CPPMs via events from the DIM. Depending on the policy configured at initialization, DCL implements load balancing, redundancy, and dynamic failover. The DCP messaging layer uses the Transport Abstraction layer to set up, manage connections, and provide interconnect/transport independent communication.

The DCL thus hides the details of the interconnect, the transport layer, and the configuration of various CPPMs to allow for easy integration of existing protocol stacks. It also provides support to a CPPM for load-balancing or

fast-failover, which is fundamental to a highly available CPPM implementation.

The Distributed Control Plane Architecture is, thus, a generic architecture that enables transparent and seamless distribution of control protocol functions across multiple processor levels in the system. The architecture enables distribution of functionality away from a hotspot closer to places in the system where it can function best, making the system highly scalable. The architecture is independent of the interconnect technology or the processor hierarchy in the system and provides well-defined interfaces so that different control protocols can be plugged in cleanly and easily.

### **CASE STUDY: THE OSPF ROUTING PROTOCOL**

In order to validate our distributed architecture, we chose the Open Shortest Path First (OSPF) protocol, one of the most widely deployed intra-domain routing protocols in the Internet. OSPF is a link state routing protocol, which means that it maintains a database describing the entire network topology of the domain, and it uses this database to calculate the shortest paths to all the nodes in that network. Functionally, the OSPF protocol consists of two major parts: (1) creating adjacencies between neighboring routers in order to exchange routing information and, (2) exchanging this information and synchronizing the databases between the routers. The Hello protocol in OSPF is responsible for establishing and maintaining neighbor relationships. OSPF uses Link State Advertisements (LSAs) to exchange the database information between routers. After synchronization of the database, OSPF uses Dijkstra's Shortest Path First (SPF) algorithm to find the shortest path to all routers/nodes in the network.

In order to monitor liveness of peer routers, each OSPF router exchanges Hello packets once every 10 seconds and each OSPF router considers a neighbor dead after it fails to respond to three consecutive Hello packets. Thus the adjacency failure detection time for OSPF-based routers currently deployed in IP networks is about 40 seconds. This means that any faults in the network or changes in the topology will be detected by the OSPF routers only after 40 seconds. In today's gigabit networks, this would mean the loss of a large amount of data traffic until another path is restored. This slow convergence has motivated discussion around millisecond convergence times and SONET-like restoration schemes for intra-domain routing protocols like the OSPF protocol in standardization forums such as the IETF. Faster failure detection, and consequently,

smaller Hello intervals are key to achieving millisecond convergence [1].

We conducted some experiments to find the computation requirements of OSPF running on the Control Plane CPU. We observed the behavior of the protocol under different network sizes and with a reduced Hello interval time. Figure 6 shows our experimental setup. We used an Intel® Pentium® III 1 GHz processor running Linux and the Moy\* OSPF protocol code [18] as the device under test (DUT). We connected this Linux router to a system running our simulation network. Our simulator simulated a network of OSPF routers connected in a grid topology. For the purposes of our experiments, we had all the routers including the DUT belong to a single area.

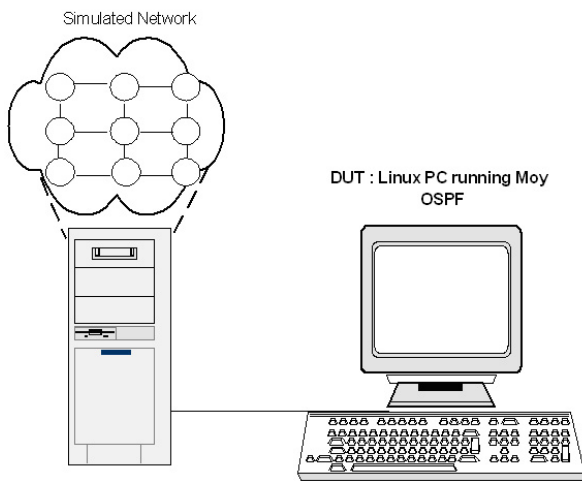


Figure 6: Experimental setup

Our first experiment was aimed at finding how the SPF calculation time varies as the size of the network of OSPF routers increases. Today’s operational ISP networks have 300-500 nodes in a single area. For example, as reported in [1], in the year 2001, one of the key AT&T networks had 292 IP routers and 765 physical links connecting the IP routers in a single OSPF area. In our experiment, we varied the number of nodes in the simulation network from between 0-900. A plot of the SPF calculation time on the DUT versus the number of routers in the network for both grid and a mesh network is shown in Figure 7. As expected, the SPF calculation time increases with the network size, since the complexity of Dijkstra’s

algorithm is  $O(eln)$ , where  $e$  is the number of edges in the network and  $n$  is the number of nodes. For a grid network topology,  $e \sim 2n$  or  $O(n)$ ; whereas for a full mesh network topology,  $e \sim n^2/2$  or  $O(n^2)$ . Thus for a full mesh network, the SPF behavior would be more quadratic in nature than for a grid network. Similar results for OSPF’s SPF complexity in today’s operational networks have been reported by others as well [1], [5].

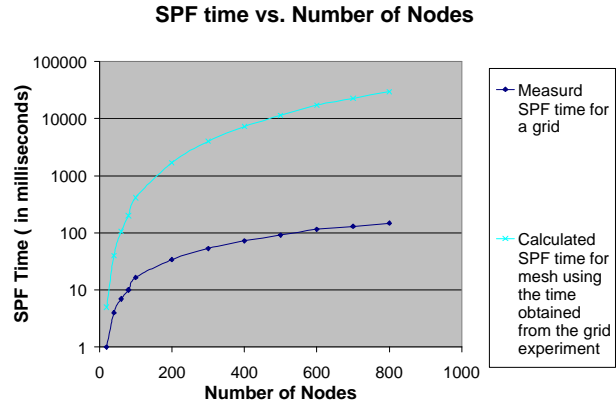


Figure 7: SPF calculation time

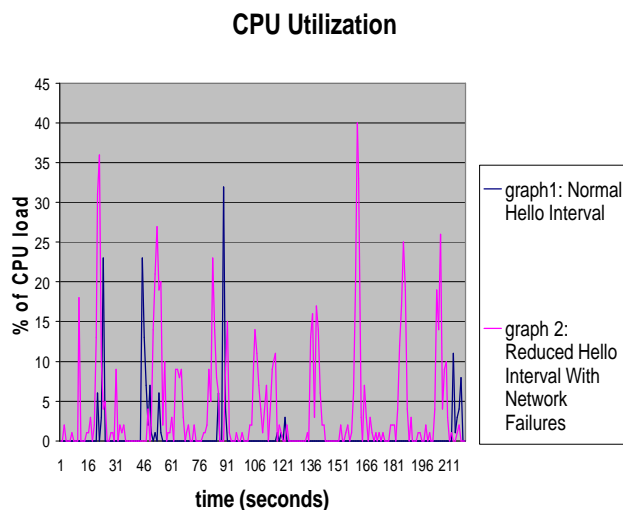
Our second experiment measured the CPU utilization of the control plane processor as the Hello interval time is reduced. We used 400 nodes in our simulation network to simulate a typical OSPF area size in an ISP. The results are shown in Figure 8. We observed the CPU behavior with only the OSPF protocol, first with the normal Hello interval of 10 seconds and corresponding Router Dead Interval of 40 seconds (graph 1 in Figure 8), and then with the Hello interval reduced to one second and the dead interval reduced to four seconds (graph 2 in Figure 8). For the latter case, we also triggered faults in the simulation network every 1-4 seconds over a period of three minutes to simulate dynamic updates happening in a real network over a short period of time. Potentially, different kinds of faults include events such as a router going down and a link going up or down. By manually injecting faults, we are just generalizing the effect of any such failure in the network and investigating the behavior of OSPF. Additionally, we wanted to see the impact of back-to-back faults on OSPF. These are rare but when they do occur, they have a significant impact as they either render the router non-operational or require various timers to be introduced in OSPF implementation, making it more compute-intensive.

As can be seen from graph 2 in Figure 8, the CPU load has numerous peaks over a short period of time with the maximum peak at 40%. Note that this is the load on the

®Intel Pentium III is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

\*Other brands and names are the property of their respective owners.

CPU with only the OSPF protocol code running, in a simple (grid topology) 400-node network. In a real router, there would be other applications running for monitoring, configuration, and management of the router. For Area Border Routers in the Internet, there would be multiple instances of OSPF, one for each, in addition to other protocols running on the Control Plane CPU.



**Figure 8: CPU utilization of control plane processor with (a) a normal Hello interval and (b) a reduced Hello interval and network failures**

It is clear from these experiments that the demand on the control plane processor by OSPF increases quadratically with network size and a reduction in the *Hello interval* time. Thus, moving to millisecond convergence will further increase the demand on the control plane processor. In order to support millisecond convergence for OSPF routers running in large networks, the OSPF implementation needs to be able to scale with the network size. This is very important because the increase in the CPU load can lead to Hello misses in the OSPF implementation, which could in turn lead to what are called *false adjacency breakdowns* in the network. Such conditions can easily multiply in a congested network and have been known to render an entire routing domain in-operational.

### Distributed OSPF

As mentioned before, OSPF functionality can be divided into the Hello protocol and the database synchronization mechanism. For a typical network element consisting of 10 line cards or forwarding elements (FEs) with 10 interfaces each, the CP would need to process  $2*3*100 = 600$  Hello packets every second (assuming a 333 millisecond *Hello interval* and Hello packets being sent

and received at this interval on 100 interfaces). Assuming each Hello packet requires 1 ms time to process, receive, and send a reply, this would mean 600 ms are spent just processing Hello packets on the CP. If we offload this functionality onto each of the FEs, then each FE would need to process only  $2*3*10 = 60$  Hello packets every second. Thus, in addition to reducing the processing and IO load on the Control Plane CPU, this functional distribution also scales with the system resources (FEs) and does not put all the load on a single FE.

We used the Distributed Control Plane framework to distribute OSPF such that the Hello protocol processing is offloaded on to the forwarding elements or line cards in the network element.

In our implementation, the line cards were composed of two IXP 2400 network processors, and the Hello protocol processing for each line card was offloaded onto the XScale processor. We used an Intel Pentium III 1 GHz processor running Linux as the control plane. The test-bed consisted of a distributed OSPF implementation in a network of non-distributed OSPF routers. The distributed OSPF consisted of a worker CPPM and a controller CPPM. The worker CPPM implemented the Hello protocol, the Designated (and Backup) Router election, and the Interface Finite State Machine (IFSM). All the other parts of the OSPF protocol including the Neighbor Finite State machine (NFSM), maintenance of the Link State Database (LSDB), routing algorithms, etc. are implemented by the controller CPPM. Each CPPM defined a *Peer Interface* via which they exchange configuration, state synchronization, and other messages.

The following messages were sent by the worker CPPM to the controller CPPM using the CPPM Peer Interface:

1. Neighbor parameters of a new neighbor or the changed parameters of an existing neighbor (such as neighbor ID, its idea of Designated/Backup Designated router, etc.).
2. Interface states as a result of the IFSM being executed.
3. Events derived from the received Hello packets such as *HelloReceived*, *AdjOK*, *2-WayReceived* [6] etc. that trigger the NFSM.

Similarly the following messages were sent by the controller CPPM to the worker CPPM:

1. Configuration information such as router ID, interface identifiers on which the Hellos are to be sent and received, type of interface, Hello interval, Hello dead interval, area associated with the interface, etc.

2. Start/Stop Hello on an interface
3. Reset router
4. Neighbor states as a result of the NFSM being executed.
5. *NeighborChange* [6] event that triggers the IFSM.

The above messages are exchanged using DCL primarily in the beginning till the Neighbor is considered *fully operational* [6], after which there are no message exchanges. The worker handles the adjacency maintenance and informs the controller only when critical changes occur in the neighbor parameters. These changes are detected by examining the received Hello packets.

## FUTURE WORK

We believe that our prototype version of a distributed OSPF router is a key step towards achieving millisecond convergence in OSPF networks. In order to achieve this, we are working towards offloading the Hello processing onto the IXP's microengines, thereby exploiting the power and programmability of the IXP network processor. By performing the Hello processing in the microengines, we can detect failures faster and run SPF calculations as frequently as required without affecting the load on the control plane processor. We will perform more experiments related to the processor utilization and other performance criteria to further validate our conclusions. In addition, we are also working towards functional distribution of other key protocols such as RSVP-TE and BGP.

## CONCLUSION

Increase in network traffic, deployment of new services, and an increase in the number of network interfaces per router are combining to create huge processing and bandwidth demands for control plane processing. In addition, high-availability requirements translate into the need for millisecond convergence when links or nodes fail. Such a convergence requires much more frequent communication among control plane entities. We need novel approaches to scale up the implementation of control protocols to accommodate these needs. We propose a distributed control plane architecture that allows a control plane protocol to seamlessly and transparently move its CPU- or IO-intensive functions away from the control card to other available processing resources within the network element. Our approach offers the following advantages:

1. Faster response to network events such as failure of a link or peer to achieve quicker network convergence.

This enables high availability and enhances the robustness of the networks, enabling support for applications that expect deterministic network behavior.

2. Scalability and improved control plane processing speed because CPU-intensive functions, such as encryption and deep-packet inspection, can be offloaded and scaled across auxiliary processing elements in the system. This also results in faster response from the protocol.
3. Resilience towards Denial of Service attacks targeted at the control plane by filtering malicious protocol packets causing these attacks on the line cards or the forwarding plane. This improves the overall robustness of the network element.

Our proposal is based on an experimental analysis based on a simulation study that examined the amount of CPU capacity needed for OSPF computations and handling OSPF messages and events.

Our analysis of the OSPF protocol can also be extended to other routing and signaling protocols. For example, BGP and LMP analysis will likely show authentication bottlenecks whereas an analysis of RSVP-TE would show the need for distributing the state related to each channel once it is established. Similarly, the DCP architecture could be used to offload the encryption/decryption functions of control protocols. We intend to study these aspects further.

## ACKNOWLEDGMENTS

We thank Gary Hemminger and Haseeb Budhani from IP Infusion for helping us with data about typical ISPs. We also thank Wayne Allen of Intel Labs for his invaluable help with our experiments.

## REFERENCES

- [1]. Basu, A and Riecke, J., "Stability Issues in OSPF Routing," *ACM SIGCOMM 2001*, UC San Diego, CA.
- [2]. Alaettinoglu, C., Jacobson, V., Yu, H., "[Towards Millisecond IGP Convergence](#)," *NANOG 20*, October 2000.
- [3]. McPherson, D., Brendel, E. et. al., "[Panel: Core Network Design and Vendor Prophecies](#)," *NANOG 25*, June 2002.
- [4]. Alaettinoglu, C. and Casner, S., "[ISIS Routing on the Qwest Backbone: a Recipe for Subsecond ISIS Convergence](#)," *NANOG 24*, February 2002.

- [5]. Sheikh, A. and Greenberg, A., "Experience in Black-box OSPF Measurement," in *Proceedings. ACM SIGCOMM Internet Measurement Workshop (IMW)*, 2001, pp. 113-125, Nov. 2001.
- [6]. Moy, J., "OSPF Version 2," *IETF RFC 2328*, April 1998.
- [7]. Malkin, G., "RIP Version 2 Carrying Additional Information," *IETF RFC 1388*, January 1993.
- [8]. Braden, R. et. al., "Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification," *IETF RFC 2205*, September 1997.
- [9]. Rekhter, Y., "A Border Gateway Protocol 4 (BGP-4)," *IETF RFC 1771*, March 1995.
- [10]. Callon, R., "Use of OSI IS-IS for Routing in TCP/IP and Dual Environments," *IETF RFC 1195*, December 1990.
- [11]. Fenner, W., "Internet Group Management Protocol, Version 2," *IETF RFC 2236*, November 1997.
- [12]. Waitzman, D., Partridge, C., and Deering, S., "Distance vector multicast routing protocol," *IETF RFC 1075*, November 1988.
- [13]. <http://developer.intel.com/design/network/products/npfamily/ixp2400.htm>
- [14]. <http://developer.intel.com/design/network/products/cpp/ixc1100.htm>
- [15]. <http://www.npforum.org>
- [16]. <http://www.ietf.org/html.charters/forces-charter.html>
- [17]. "RSVP-TE: Extensions to RSVP for LSP Tunnels," *IETF RFC 3209*, December 2001.
- [18]. Moy, J.T., "OSPF Complete Implementation," September 2000.
- [19]. "OSPF TE: Traffic Engineering Extensions to OSPF Version 2," *IETF RFC 3630*, September 2003.
- [20]. <http://www.3gpp.org>
- [21]. Matthew Adiletta et. al., "The Next Generation of Intel IXP Network Processors," *Intel Technology Journal, Volume 6, Issue 03*, August 15, 2002, pp. 6-18.

## AUTHORS' BIOGRAPHIES

**Manasi Deval** works on defining and developing technology for modular networking and communication platforms at Intel Labs. She received her M.S. degree in

Electrical Engineering from the University Of Southern California and a B.E. degree in Electronics and Telecommunication from Maharashtra Institute of Technology, India. Her e-mail is manasi.deval at intel.com.

**Hormuzd Khosravi** is a senior network software engineer in Intel Labs. He has been involved in defining and developing standards and technology for modular networking and communications platforms at Intel. Previously, Hormuzd held the position of research assistant at C&C Research Labs, NEC USA. Hormuzd received his M.S. degree in Computer Engineering from Rutgers University and his B.E. degree from Bombay University, India. His e-mail is hormuzd.m.khosravi at intel.com.

**Rajeev Muralidhar** is a senior network software engineer at Intel Labs where he is involved in defining and developing standards and technology for modular networking and communications platforms. Rajeev is also the principal editor of the Packet Handler Task Group at the Network Processing Forum. He received his M.S. degree from Rutgers University and his B.E. degree from the National Institute of Technology, India, both in Computer Engineering. His e-mail is rajeev.d.muralidhar at intel.com.

**Suhail Ahmed** works as a senior network software engineer in Intel Labs where he is involved in defining and developing technology for modular networking and communications platforms. He received his M.S. degree in Electrical Engineering from SIU, Carbondale and his B.E. degree in Electronics from Bangalore University, India. His e-mail is suhail.ahmed at intel.com.

**Sanjay Bakshi** is currently an engineering manager and architect in the Intel Corporate Technology Group. He has lead a number of projects related to the usage of the Intel Internet Exchange Processor in various fields such as IP routing, MPLS, 3G wireless, and next-generation control plane architecture. Sanjay received his B.E. degree in Computer Science from the Regional Engineering College, Tiruchirapalli, India. His e-mail is sanjay.bakshi at intel.com.

**Raj Yavatkar** is currently the chief software architect for the Internet Exchange Architecture at the Intel Communications Group. Raj received his Ph.D. degree in Computer Science from Purdue University in 1989. He is a co-author of the book *Inside the Internet's Resource reSerVation Protocol (RSVP)* published by John Wiley in 1998. He currently serves on the editorial board of the IEEE Network magazine. His e-mail is raj.yavatkar at intel.com.

Copyright © Intel Corporation 2003. This publication was downloaded from <http://developer.intel.com/>.

Legal notices at <http://developer.intel.com/sites/developer/tradmarx.htm>.

For further information visit:

[developer.intel.com/technology/itj/index.htm](http://developer.intel.com/technology/itj/index.htm)