Data Networks Project 2: Design Document for Intra-Domain Routing Protocols

Assigned: Wed, 30 May 2007

Due: 11:59pm, Wed, 20 June 2007

1 Introduction

You have just joined Bisco Systems, a networking equipment company, as a Design Engineer. Your first project is to design and implement intradomain routing protocols for Bisco's upcoming product, the GSR9999 router. Your first step is to write a design document.

In particular, you are to design and implement a variant of the distance vector routing protocol (DV) *and* a variant of the link-state routing protocol (LS). The functional specifications of the DV and LS protocols are provided (see Section 6 and 7) and your implementation *must* follow these specifications. You are also given the GSR9999's system interfaces (see Section 5) which you must use to implement the routing protocols.

Note that the routing protocols you are going to implement and the network model in this project are simplified. The DV and LS protocols have a limited set of features. Every node in the network is assumed to be a router and is identified by a unique ID. No hierarchical addressing is used. Each node has a number of ports, which may or may not be connected to a neighbor router. Figure 1 illustrates a simple network example.

In Project 3, you will actually implement these protocols based on your design in a GSR9999 simulator. The simulator essentially provides the GSR9999 system interfaces as described in Section 5. Versions of the simulator are available in C++ and Java, so you may use either of these languages in your protocol design and implementation. You should also assume that your two routing protocols will be encapsulated in a RoutingProtocol object.

Assignment: In this project, we ask you to write a design document of **no more than six pages** (excluding figures and pseudocode listings) to describe the details of your design. The specific requirements are described in Section 2, 3 and 4. The goal of this project is to give you a chance to carefully think through the design of the entire system before you start the implementation, which we will ask you to complete in Project 3.

Grading: Grading of your design document will be based on the correctness and completeness of your design, as well as the quality and clarity of your writing. Your document must be type-set in 11 point New Times Roman font with 1 inch margins and no more then 45 lines per page (like this document). Use of appropriate illustrations (tables, figures) is required. A clear, well-organized writeup with almost no design flaws or missing pieces will earn you full marks. Please pay particular attention to

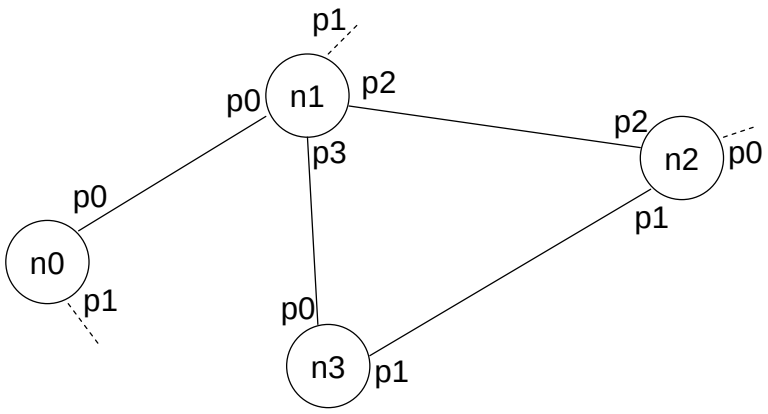

Figure 1: Simple network example.

the organization of your document. A clear, concise, well organized document is far better than a wordy and confusing one. **IMPORTANT**: You are strongly encouraged to schedule a meeting with a TA during the week of June 11, 2007, to discuss a complete draft of your design document. The purpose of this meeting is to provide you with feedback on your design and your document. Watch the newsgroup for meeting scheduling instructions.

Working in groups: You need to form groups of two students for this assignment. We suggest that you use the course mailing list to organize teams.

2 Design Document Guidelines

The purpose of the design document is to provide sufficient design details and implementation instructions such that a programmer (perhaps a summer intern) can independently and successfully implement the routing protocols according to your document. Your design document must contain details such as data structure definitions, parameter definitions, and procedure pseudocode. To complete this document, you will need to first understand the given information in Section 5, 6, and 7. Your document should roughly follow the structure below. Again, use of illustrations like figures and tables when appropriate.

Section A. Design Overview - You should describe the overall organization of your system. What are the major logical components in your design and what is the rationale behind your design? What are the responsibilities of each component? How do the components interact? You should have a block diagram to illustrate the different components in your design.

Section B. Core Data Structures - You should describe the core data structures needed to implement your various components. Give detailed data structure definitions in C/Java and pseudocode for manipulating the data structures if the pseudocode is non-trivial (e.g. forwarding table, distance vector, etc).

Section C. Component Operations - You should describe the implementation of the operations in each component of your design using pseudocode and illustrations etc. See Section 3 for some hints. Here, the focus is on the procedures and algorithms used in your proposed design. How the core data structures are used should be described. You should describe how common system events (alarm, packet arrival) are both generated and handled in your components.

3 Design Hints

Your design document must address the technical issues described below. **Note that this is not an exhaustive list of issues**. These issues are provided to help stimulate and organize your thoughts.

Common System Design Issues - You will need to address design issues that are common to both DV and LS routing protocols. Below are some items that your document must address.

- Router ID - At bootup time, your routing software is initialized with a unique 16-bit router identifier. 0xffff is reserved as an invalid router ID.
- Routing Procotol Selection - At bootup time, your routing software is initialized with the routing protocol that should be used in the system (either DV or LS). Once initialized, this selection does not change.
- Port Status Data Structure - At bootup time, your routing software is initialized with the total number of ports on the router. The total number of ports does not change over time. A port *may* be connected to a neighbor router, or may be unconnected. A neighbor router is uniquely identified by its 16-bit identifier. Connections between neighboring routers have a given non-zero round-trip delay. The round-trip delay to a neighbor router should be used as the link cost in your routing protocols.
- Port Status Monitoring - In the GSR9999 router, port status information (i.e. neighbor router ID and link round-trip delay, if connected) must be detected dynamically by software. Use pseudocode to describe a port status detection algorithm that attempts to test port status once every 10 seconds and updates the port status data structure accordingly. You will need to use the PING and PONG packet types and the time() and set_alarm() system calls specified in Section 5 to learn the neighbor router ID and measure the round-trip delay. You will need to describe (1) how to generate PING packets periodically, (2) how to construct PING packets, (3) how PING packets are processed when received, (4) how PONG packets are processed when received.

  To determine port status using PING/PONG, you must use an embedded timestamp in the PING/PONG messages and use the following procedure. When a router generates a PING packet, it must store the current time in the PING message payload, then send the PING message to a neighbor with the correct source ID. When the neighbor router receives the PING message, it must update the received message's type to PONG, update the source ID to its own, then send the resulting PONG message (with the original timestamp still in the payload) immediately back to the neighbor. When the PONG message is received, the timestamp in the message is compared to the current time to compute the RTT. This is in fact how the `ping` UNIX tool measures RTT.

  Since PING messages are generated every 10 seconds, a port's status is refreshed by PONG messages approximately once every 10 seconds. A link should be declared dead when the status has not been refreshed for 15 seconds. The port status should be properly changed within 1 second of the expiration. That is, if you implement a 1-second periodic check on all the state's freshness, that's sufficient.

- Forwarding Table Data Structure - Describe a suitable C/Java data structure for maintaining the forwarding table. Note that at bootup time, a router in the network does not know how many routers are in the network. The number of routers in the network can also change dynamically. Your data structure must support insertion and deletion of forwarding table entries. This forwarding table structure should be generic such that it can be used by your packet forwarding function no matter which routing protocols (DV or LS) is being used.

- Packet Forwarding - Use pseudocode to describe how to handle the forwarding of regular DATA packets.

- Note that the C/Java definitions of numerical constants for packet types, protocol types, and infinity cost will eventually be provided to you in Project 3 for uniformity. In this design document, however, you can define your own.

- Keep in mind that your router implementation will have to interoperate with other router implementations. Therefore, you need to follow the specifications precisely to ensure interoperability. Moreover, your implementation should be robust to malformed packets from other routers. Your implementation should ignore such packets.

DV Protocol Design Issues - See Section 6 for the specification of the protocol you need to implement. Below are the set of issues you must address.

- Distance Vector Data Structure - Describe a suitable C/Java data structure for maintaining the distance vector at a router. Note that an entry in the distance vector must be refreshed periodically or else it must be removed after a timeout. Your data structure should support dynamic insertion and deletion.

- Distance Vector Freshness Check - Use pseudocode to describe how you will implement a periodic check of distance vector freshness.

- Direct Neighbors Maintenance - Use pseudocode to describe how to insert, delete, and refresh direct neighbors of a router in the distance vector structure.

- DV Announcement - Use pseudocode to describe how to send DV routing update packets.

- DV Routing Update Packet Construction - Use pseudocode and suitable C/Java data structures to describe how a DV routing update packet can be constructed. Your protocol should implement the poison reverse DV variant.

  Link cost (i.e. round-trip time) should be represented in units of milliseconds in routing update messages as a 16-bit unsigned integer.

  The DV update packet should contain as few entries as possible. That is, if a destination D is not reachable from router N, then router N's DV update packets should not include an entry for D. As a result, the size of the DV updates is minimized, and the withdrawal of a route is implicit. In other words, you should not have (ID, INFINITY_COST) pairs in your DV update packets except when necessary for implementing poison reverse.

- DV Packet processing - Use pseudocode to describe how to process DV routing update packets and update the distance vector data structure.

  Your DV implementation should only record the best route among all neighbors for a destination, and not record second best route, third best route, etc. That is, when the best route fails, the only way to discover an alternative route is from getting a new DV update message containing an alternate route.

- Forwarding Table Maintenance - Use pseudocode to describe how to update the forwarding table based on information in the distance vector data structure.

- DV update packets are generated periodically every 30 seconds. Triggered updates should be considered as additional updates separate from the periodic updates. That is, triggered updates should not affect the schedule of the normal 30 second periodic updates. For example, suppose periodic updates are sent at time 0, 30, 60, 90, 120... etc. Even if a triggered update occurs at time 42, the regular updates should still occur at time 60, 90, 120, etc.

LS Protocol Design Issues - See Section 7 for the specification of the protocol you need to implement. Below are the set of issues you must address.

- Link-state data structure - Describe a suitable C/Java data structure for storing link-state information collected from other routers in the network. Note that link-state information must be periodically refreshed or else it is removed after a timeout. Your data structure should support dynamic insertion and deletion. This link-state information will also be used by Dijkstra's algorithm for computing shortest paths.

- Link-state Freshness Check - Use pseudocode to describe how you will implement a periodic check of link-state freshness.

- LS Announcement - Use pseudocode to describe how to generate LS routing update packets.

- LS routing Update Packet Construction - Use pseudocode to describe how to construct a LS routing update packet.

  Link cost (i.e. round-trip time) should be represented in units of milliseconds in routing update messages as a 16-bit unsigned integer.

  The LS update packet should be as small as possible. You should never have (ID, INFINITY_COST) pairs in your LS update packets.

- LS Packet processing - Use pseudocode to describe how to process LS routing update packets and update the link-state data structure.

- Forwarding Table Maintenance - Use pseudocode to describe how to update the forwarding table based on information in the link-state data structure and Dijkstra's shortest path algorithm.

- LS update packets are generated periodically every 30 seconds. Triggered updates should be considered as additional updates separate from the periodic updates. That is, triggered updates should not affect the schedule of the normal 30 second periodic updates, as with the DV algorithm.

4 Submission Instructions

Please submit your design document in Acrobat .pdf format.

You must email your design document **as an attachment** to datanets-projects@mpi-sws.mpg.de by 11:59pm on the due date. In your email, please specify the names of the students in your group and their immatriculation numbers.

IMPORTANT: In addition, your group is strongly encouraged to see a TA during the week of June 11, 2007, to discuss a complete draft of your design document. It is each team's responsibility to schedule this meeting. Please watch the newsgroup for meeting scheduling instructions.

5 Bisco GSR9999 System Interfaces

You can assume that the GSR9999 operating system has only a single thread of execution for all running software. That is, a function call to your routing protocol software will be executed without interruption by the system. Everything happens sequentially on a router. This assumption will greatly reduce the complexity of your code. An implication is that any pending alarm will be set off only after a function call is completed and control has been returned to the operating system. You can assume your functions will take a negligible amount of time to execute, so the impact on the timing of scheduled alarms is negligible.

5.1 Initialization

On bootup, the system will initialize your routing software by calling your init() function with the following information:

- Number of ports
- Your router ID
- Routing protocol used (DV or LS)

5.2 System Call Interfaces

The following system calls are provided by the GSR9999 system.

- $set\_alarm(duration, *d)$ - Set an alarm. You specify an amount of time that the system should wait before setting off an alarm. When an alarm is set off, your $handle\_alarm(F)$ function will be invoked (see Section 5.3). d is a pointer to an object that is associated with the alarm; it is passed to your $handle\_alarm()$ function so that you can use d to figure out what the alarm is meant for.

  When setting an alarm using $set\_alarm()$, the object refered to by d should not be modified by your code (or freed in C++) until the corresponding alarm has been delivered via $handle\_alarm()$.

- $send(p, *pkt, s)$ - Send a packet pkt of size s bytes on the port number p.

When using C++, the memory storing the packet is "owned" by the underlying system, so you should not free or modify a packet's memory after passing it to send(). This also implies that packet memory must be dynamically allocated. On the other hand, on receiving a packet, once your $recv()$ function is called, the packet memory is owned by your routing software, so it is your routing software's duty to free packet memory after a receive if the packet memory is no longer needed.

- $time()$ - Return the current time of the system in milliseconds since bootup.

5.3 Handler Interfaces You Must Implement

- $handle\_alarm(*d)$ - When an alarm is set off, the system calls your $handle\_alarm()$ function. Your function should inspect the associated object refered to by d to determine the correct course of action.

- $recv(p, *pkt, s)$ - When a packet pkt of size s arrives via port number p, your $recv()$ function is called. Your function should inspect the content of pkt to determine the correct course of action.

  When a DATA packet originates at a router (the DATA packet will be created by the simulator's `xmit` event), it will be received by your RoutingProtocol's $recv()$ function with a special incoming port number of 0xffffffff. This indicates that the DATA packet originates locally, rather than being received from a neighbor.

  When a DATA packet is received by its destination, the packet memory should be freed (C++ only).

5.4 General Packet Format

Figure 2 illustrates the general packet format that you must use. Important note: Network byte order is Big Endian. So you must use byte order conversion functions (e.g. htons() and ntohs() in C++) to transmit packets with the proper byte ordering. Packet type is an 8-bit number corresponding to the 5 packet types defined above. The Reserved section is unused. Size is the size of the entire packet in number of bytes. Source ID is the node that generated the packet. Destination ID is the node that the packet is destined to.

5.5 Packet Type

There are five packet types in the system:

- DATA - Regular data packet that needs to be forwarded by a router.
- PING - A packet sent only to an immediate neighbor.
- PONG - A packet sent immediately in response to a PING packet to a neighbor. This allows you to detect the existence of a neighbor as well as measure the round-trip delay (i.e. link cost).
- DV - A distance vector routing update packet.
- LS - A link-state routing update packet.

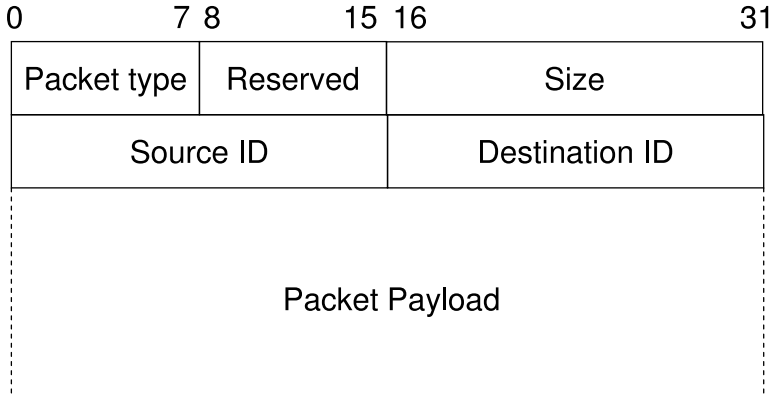

Figure 2: General packet format.

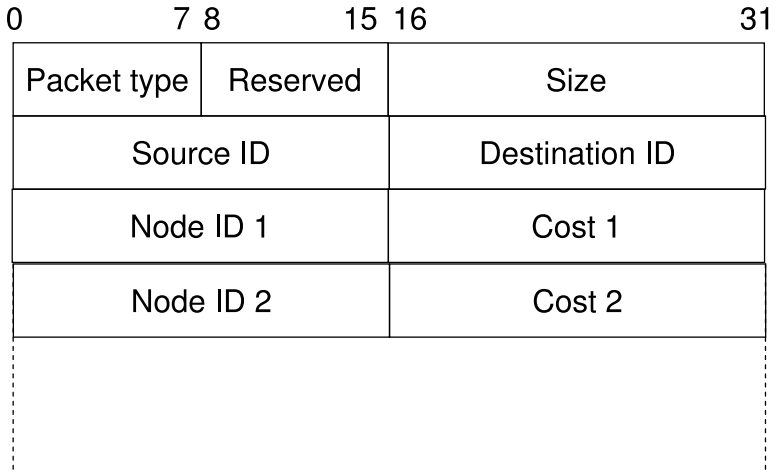

Figure 3: DV update packet format.

5.6 Neighbor Status Detection

You need to use the PING and PONG packet types to periodically check the status of a port and discover whether a neighbor exists, and any neighbor's ID and the round-trip delay to the neighbor. The checks should be performed on every port once every 10 seconds.

6 Specifications of the DV Protocol

6.1 DV Routing Update Packet Format

See Figure 3. You must construct your DV routing update messages according to this format.

6.2 Behavioral Specifications

You must implement the following features:

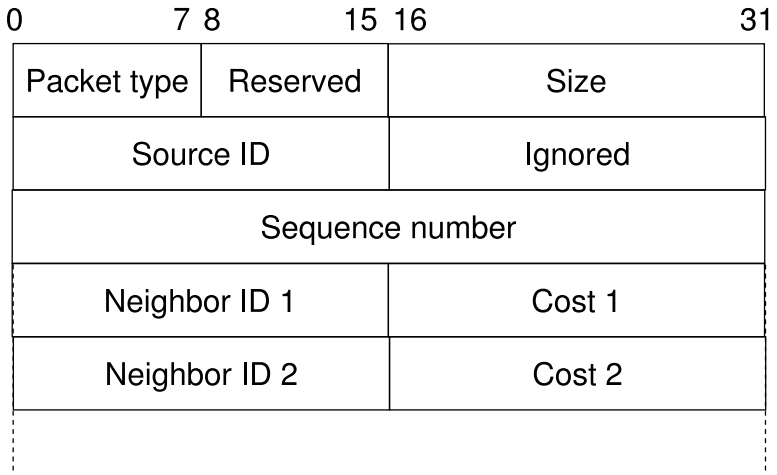

Figure 4: LS update packet format.

- You must implement the poison reverse variant of distance vector protocol.
- The link cost is the round-trip delay (in ms) of a link.
- DV periodic updates are sent every 30 seconds.
- DV entries that are not refreshed within 45 seconds are timed out and removed. You should remove the expired state within 1 second of the expiration time. That is, if you implement a 1-second periodic check of all the state's freshness, that's sufficient.
- DV triggered updates are sent as soon as a local DV change occurs.
- Your DV protocol should update the forwarding table to always reflect the current best known paths.

7 Specifications of the LS Protocol

7.1 LS Routing Update Packet Format

See Figure 4. You must construct your LS routing update messages according to this format.

7.2 Behavioral Specifications

You must implement the following features:

- You must implement flooding of LS update packets.
- You must use sequence numbers to correctly implement LS update.
- You do not need to implement reliable flooding. That is, you do not need to send acknowledgements or retransmit LS update packets.

- The link cost is the round-trip delay (in ms) of a link.
- LS periodic updates are sent every 30 seconds.
- LS entries that are not refreshed within 45 seconds are timed out. You should remove the expired state within 1 second of the expiration time. That is, if you implement a 1-second periodic check of all the state's freshness, that's sufficient.
- LS triggered updates are sent as soon as a neighbor status change is detected.
- Dijkstra's algorithm must be used to compute the correct shortest paths based on the most current link-state information.
- Your LS protocol should update the forwarding table to always reflect the most current computed shortest paths.