Um sniffer de rede com interface GTK2

Controle de Tráfego

O construtor de interfaces gráficas Glade traz o poder e a facilidade do arrastar-e-soltar à montagem de GUIs. Scripts em Perl podem interpretar os arquivos XML descritores de interface em tempo de execução. Como exemplo, vamos fabricar um sniffer de redes com uma interface bem bonitinha. POR MICHAEL SCHILLI

Se você precisa saber quem está "logado" em sua rede local e prefere uma interface GTK2 para visualizar a informação, o script *capture.pl* é tudo o que seu médico recomendou. Ele usa um módulo do CPAN chamado *Net::Pcap* (mais informações em [1]) para farejar o tráfego em cabos ou redes sem fio, decodificar os pacotes capturados, determinar se o remetente pertence à rede local e mostra o número IP em um widget de exibição de texto (figura 1).

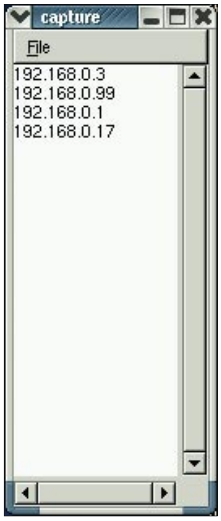

Figura 1: O programa GTK2 *capture.pl* mostra em uma lista todos os computadores ativos na rede local.

O script mostra os resultados mais recentes no topo da lista, atualizada dinamicamente. O menu *File* possui uma opção *Reset* que permite excluir os endereços da lista, além da tradicional opção para sair (*Quit*) do programa.

GUI baseada em diretivas XML

O script não usa instruções diretas no próprio código para definir sua interface gráfica. Em vez disso, interpreta um arquivo descritor em XML no momento em que foi chamado pelo usuário – o termo consagrado é "em tempo de execução". Para criar a interface, os programadores podem usar o Glade 2, disponível em [2]. Depois de interpretar o arquivo descritor, o programa *capture.pl* monta a interface na tela e espera por eventos – como o clique de um mouse, por exemplo.

A maioria dos construtores de interfaces gráficas têm uma abordagem diferente para o problema. Normalmente, o projetista da interface arrasta com o mouse objetos chamados *widgets* (botões, barras de rolagem, áreas de texto) e os posiciona na janela do futuro aplicativo. Depois, define os eventos que são disparados pelos objetos. Depois disso, o construtor de interfaces converte os resultados em código-fonte na linguagem definida pelo programador (C/C++, Java, Python...), deixando o arremate final por conta do desenvolvedor. Infelizmente, depois de "arrematado", os construtores de GUI são incapazes de ler novamente o código.

O Glade pode ser usado das duas formas: tanto gera um código em C como um arquivo descritor em XML, que pode ser interpretado por um programa que utilize a biblioteca *Libglade*. Para programas em Perl, o módulo CPAN *Gtk2::GladeXML* faz a ligação entre a linguagem Perl e a biblioteca.

A figura 2 mostra o Glade em ação. A janela principal é mostrada na parte superior esquerda. Neste screenshot temos a criação de um novo projeto. A barra de ferramentas na parte inferior contém uma coleção de widgets; o

Figura 2: é possível usar o Glade como ferramenta para construir interfaces gráficas sem complicação. Os resultados são armazenados em um arquivo descritor XML.

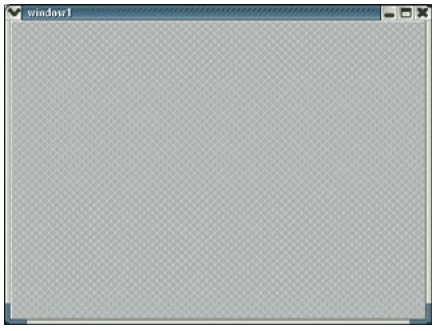

Figura 3: A janela principal, ainda vazia, sendo montada no Glade. Nela, colocaremos os widgets apropriados.

programa que está sendo construído é mostrado no centro. Na parte superior direita podemos alterar os atributos do widget (nome, tamanho, interação com o usuário, sinais). Abaixo da caixa de propriedades há uma visão hierárquica em forma de árvore com os widgets disponíveis para o programa que está sendo montado.

Para criar uma nova GUI, clique no ícone de tela principal na barra de ferramentas (é o botão com as listas azuis). Isso cria uma janela vazia, como vemos na figura 3. Um novo contêiner do tipo Vbox cria uma barra de menus na parte superior da janela e uma área de texto no centro. Para inserir mais widgets, clique no ícone apropriado e depois na janela que está sendo construída.

Ainda falta um menu, uma janela "rolante" e a caixa de texto. A figura 4 mostra a janela do aplicativo, já próxima do polimento final. Há muitos menus inúteis para um programa tão simplório como o *capture.pl*, mas a opção *Edit Menus* na janela de propriedades faz surgir uma caixa de diálogo em que podemos nos livrar de todos eles (figura 5).

Quaisquer outras modificações são muito simples. Por exemplo, o tamanho da janela principal de *capture.pl* é

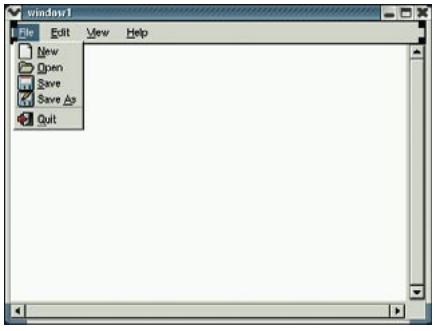

Figura 4: Estamos quase lá. Basta se livrar dos menus supérfluos...

de 300 pixels de comprimento por 120 pixels de largura. Podemos verificar isso (e alterar, se necessário) pela janela de propriedades.

O próximo passo é clicar no botão *Save* na janela principal do Glade e informar o nome do projeto, *capture*, para gravar os arquivos: *capture.glade* e *capture.pglade*. Nós realmente não precisamos do segundo arquivo, mas o primeiro contém o descritor XML para a interface que acabamos de criar.

O script *capture.pl* interpreta o arquivo XML na linha 27 quando o construtor *Gtk2::GladeXML* é chamado. O arquivo XML contém definições e atributos individuais para cada widget, bem como sua posição relativa na janela do programa. Por exemplo, a descrição XML do widget de texto possui as seguintes propriedades:

```
<property name=➥
"editable">False</property>➥
<property name="cursor_visible">➥
</property>
```

Em nosso exemplo, o programador criou, usando apenas o mouse, um widget não editável com cursor invisível. As duas linhas de código a seguir teriam efeito idêntico:

```
$text->set_editable(0);
$text->set_cursor_visible(0);
```

Sinais

O método *signal_autoconnect_all* na linha 56 define a porção dinâmica da GUI. Associando os widgets no descritor XML com sinais predefinidos, como *on_quit1_activate* (o equivalente a selecionar o menu *File* | *Quit* com o mouse) e *on_reset1_activate* (o mesmo com *File* | *Reset*), podemos ligá-los a rotinas em Perl apropriadas.

Os nomes foram automaticamente definidos pelo Glade (figura 5). Se preferir, é possível editá-los no campo de propriedades do Glade. Na linha 67, *capture.pl* inicia a rotina principal *main()*; se tudo estiver certo, a interface desenhada deve surgir na tela, encorajando o usuário a clicar em algo.

Uma viagem sem solavancos

O ato de "escarafunchar" uma rede requer um certo poder de processamento, o que significa que a CPU talvez não consiga atualizar os dados na interface "ao vivo". Congelamentos momentâneos e "soluços" não são, nem de longe, aceitáveis. Para responder prontamente à interação do usuário ao mesmo tempo em que usa e abusa do módulo *Net::Pcap*, o programa *capture.pl* usa a função *fork()* para criar um processo-filho na linha 35. O processo-pai cuida da interface com o usuário enquanto o processo-filho examina diligentemente os pacotes que passam para lá e para cá na rede.

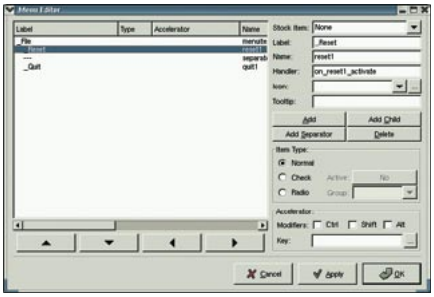

Figura 5: ...usando o editor de menus. Fácil, não?

Antes do *fork()*, um *pipe()* é chamado para criar uma linha de comunicação entre os processos pai e filho. Quando um processo filho descobre um novo IP na rede, usa o *WRITEHANDLE* para enviar a informação ao pai, que usa o *READHANDLE* para pegar a mensagem do outro lado.

Para que a GUI ignore o *pipe* até que alguma coisa aconteça, bem como para obedecer ao controle do usuário sobre a interface, a linha 76 de nosso programa possui um *watch*:

```
Glib::IO->add_watch➥
(fileno(READHANDLE), 'in',➥
\&watch_callback);
```

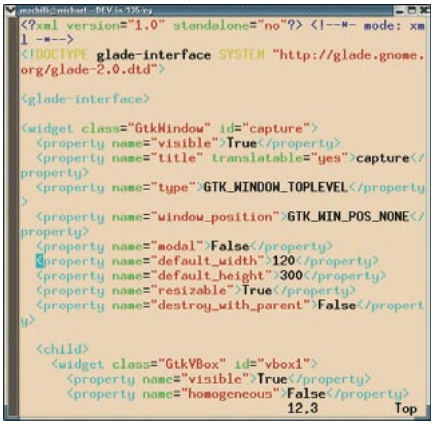

Figura 6: As definições XML no arquivo *capture.glade*, criadas pelo Glade.

Ele chama a função de retorno definida na linha 84 (*watch_callback*) sempre que algum dado chegar a *READHANDLE*. O GTK2 é baseado na biblioteca Glib e é, portanto, capaz de acessar os serviços de baixo nível que a biblioteca oferece. A função *add_watch()* espera um descritor de arquivos em vez de um *handle*, portanto precisaremos uma chamada à função *fileno()* do Perl para converter o *READHANDLE*.

No escritório do sabujo

O módulo *Net::Pcap* é uma interface para a biblioteca libpcap. Ela lê quaisquer pacotes que estejam trafegando na rede para, imediatamente, analisar e filtrar os pacotes em altíssima velocidade de acordo com critérios previamente definidos. Programas como o Ethereal são baseados na libpcap.

A função *snooper()* na linha 107 consulta a primeira interface de rede ativa (tipicamente *eth0*) usando a função *Net::Pcap::lookupdev*. A chamada a *Net::Pcap::lookupnet* identifica, então, o endereço e a máscara de rede correspondente.

Net::Pcap::open_live(), na linha 125, inicia uma captura de tráfego e registra até 1024 bytes por pacote para análise posterior. Como o terceiro parâmetro é *1*, a função coloca a interface de rede em modo promíscuo, ou seja, ordena à placa de rede que registre qualquer pacote que ela veja, não apenas os pacotes destinados a ela. O quarto parâmetro, *-1*, indica que não precisaremos de um tempo de espera (*timeout*). Caso queiramos um *timeout*, o valor indicado será o tempo em milissegundos.

A função *Net::Pcap::loop* na linha 105 pula para um *loop* que executa a função de callback apropriada, *snooper_callback()*, sempre que encontrar um pacote. O segundo parâmetro, *-1*, faz com que o programa "fareje" a rede indefinidamente em vez de parar depois de acumular um certo número de pacotes.

O último parâmetro na chamada a *Net::Pcap::loop* é uma referência a uma matriz de dados úteis, repassados a *snooper_callback()* como primeiro parâmetro a cada execução da função. A matriz contém três valores, *[$fd, $addr, $netmask]*, um descritor de arquivos *$fd* - que será enviado através do *pipe* ao processo-pai - e o endereço e máscara de rede previamente detectados.

Análise dos pacotes

Na linha 128, *Net::Pcap::loop* salta para um loop perpétuo, que chama a função *snooper_callback()* para cada pacote capturado e passa a ela os dados do cabeçalho e do *payload*. Dentro de *snooper_callback()*, a função *NetPacket::Ethernet::strip* extrai a informação Ethernet do pacote, enquanto *NetPacket::IP->decode()* lida com a camada IP e retorna a referência em um *hash*, que armazena o endereço IP de origem em *src_ip*.

inet_aton(), função presente no módulo *Socket*, converte a *string* formatada como "AA.BB.CC.DD" em um formato binário. Os valores do endereço e da máscara de rede (*$addr* e *$netmask*, respectivamente), detectados previamente, são armazenados no formato binário nativo do processador (*Big Endian* ou *Little Endian*). A chamada à função *pack* na linha 154 do programa converte-os ao formato de rede independente do processador.

Depois disso, a função *capture.pl* verifica se o endereço IP *$ip* pertence à rede indicada pela variável *$network_addr*, comparando os valores em *($ip & $mask)* com o endereço de rede em *$network_addr*. Esta condição é satisfeita para pacotes vindos da rede (ou subrede) à qual pertence o computador que está rodando o programa. A função *syswrite()* na linha 156 envia o endereço IP capturado para o processo pai sem usar buffers.

A mensagem atravessa o duto, definido na linha 31, e dispara um evento (graças ao "observador" definido na linha 76) que chama a função *watch_callback()* no processo pai. A matriz global *@IPS*, que contém todos os IPs conhecidos, é atualizada nesse momento. IPs identificados há pouco tempo não são armazenados instantaneamente no hash *%IPS*: a função *unshift()* os envia para o início de uma matriz (chamada *@IPS*), que determina a ordem de exibição. A linha 81 cria uma cadeia de caracteres contendo todos os endereços IP conhecidos até então, um em cada linha. Na linha 99, a função usa essa cadeia de caracteres para atuar o widget de exibição de texto.

Instalação

Como nosso pequeno programa usa uma interface gráfica baseada no GTK2, precisa de um bom punhado de módulos Perl. Alguns dos mais importantes são: *ExtUtils::Depends*, *ExtUtils::PkgConfig*, *Glib*, *Gtk2*, *Gtk2::GladeXML*, *Net::Pcap*, e *NetPacket*. A maneira mais fácil de instalá-los é usar um shell CPAN, mas algumas modificações "à unha" são necessárias. Se a biblioteca *libglade* não estiver instalada em sua máquina, dê uma passada em [3] e baixe-a. O montador de interfaces Glade está disponível em [2].

Durante a instalação de *Net::Pcap* certifique-se de iniciar a fase de testes (*make test*) como usuário root, mesmo que a instalação propriamente dita não precise de privilégios de superusuário. Se mesmo assim uma mensagem de erro insistir em aparecer, tente emitir manualmente o comando *make install* no diretório de compilação do módulo.

Antes de executar o programa, os usuários devem se certificar de que a definição XML da interface gráfica está realmente gravada no arquivo *capture.glade*. Se preferir colocar todos os seus ovos no mesmo cesto, pode modificar a linha 27 e ter tudo dentro de *capture.pl*:

```
my $xml = join "\n", <DATA>;
my $g = Gtk2::GladeXML->➥
new_from_buffer($xml);
```

Depois copie o conteúdo de *capture.glade* para uma seção *DATA* no final do arquivo *capture.pl*:

```
# ... Fim do arquivo capture.pl
__DATA__
<?xml version="1.0" ...
<!DOCTYPE glade-interface ...
```

INFORMAÇÕES

[1] Monitoramento de redes com o módulo Perl Net::Pcap: Robert Casey: "Monitoring Network Traffic with Net::Pcap", The Perl Journal 7/2004, página 6.

[2] Site oficial do Glade: *http://glade.gnome.org*

[3] Código fonte da biblioteca libglade: *http://ftp.gnome.org/pub/GNOME/sources/libglade*

Como a placa de rede será colocada em modo promíscuo, é necessário executar o programa *capture.pl* como root.

O construtor de interfaces gráficas Glade oferece um poder enorme para a criação de GUIs complexas e atraentes. Os recursos de arrastar-e-soltar e WYSIWYG tornam brincadeira de criança uma tarefa que, da maneira tradicional, seria horrivelmente demorada. A representação XML, que independe de plataforma, é muito elegante e enxuta. Sua maior vantagem é retirar do programa principal o código relativo à construção da interface gráfica na tela, normalmente estático, volumoso e desajeitado. Isso permite que os desenvolvedores se concentrem em aspectos mais dinâmicos da lógica do programa. ■

Listagem 1: capture.pl

```
001 #!/usr/bin/perl
002 ##################################
003 # capture -- Gtk2 GUI
004 #       observing the network
005 # Mike Schilli, 2004
006 # (m@perlmeister.com)
007 ##################################
008 use warnings;
009 use strict;
010
011 use Gtk2 -init;
012 use Gtk2::GladeXML;
013 use Glib;
014 use Net::Pcap;
015 use NetPacket::IP;
016 use NetPacket::Ethernet;
017 use Socket;
018
019 our @IPS = ();
020 our %IPS = ();
021
022 die "You need to be root.\n"
023   if $> != 0;
024
025 # Load GUI XML description
026 my $g =
027   Gtk2::GladeXML->new(
028     'capture.glade');
029
030 # Child/Parent comm pipe
031 pipe READHANDLE, WRITEHANDLE
032   or die "Cannot open pipe";
033
034 # Fork off a child
035 our $pid = fork();
036 die "failed to fork"
037   unless defined $pid;
038
039 if ($pid == 0) {
040     # Child, never returns
041     snooper(\*WRITEHANDLE);
042 }
043
044   # Parent, init text window
045 my $buf =
046   Gtk2::TextBuffer->new();
047
048 $buf->set_text(
049     "No activity yet.\n");
050
051 my $text = $g->get_widget(
052     'textview1');
053
054 $text->set_buffer($buf);
055
056 $g->signal_autoconnect_all(
057     on_quit1_activate =>
058       sub {
059
060         # Stop snooper
061         kill('KILL', $pid);
062         wait();
063         Gtk2->main_quit;
064       },
065
066     on_reset1_activate =>
067       sub {
068
069         # Reset display
070         @IPS = ();
071         %IPS = ();
072         $buf->set_text("");
073       },
074 );
075
076 Glib::IO->add_watch(
077     fileno(READHANDLE), 'in',
078     \&watch_callback);
079
080 # Enter main loop
081 Gtk2->main();
082
083 ##################################
084 sub watch_callback {
085 ##################################
086   chomp(my $ip =
087         <READHANDLE>);
088
089   # Register IP if unknown
090   unshift @IPS, $ip unless
091       exists $IPS{$ip};
092
093   $IPS{$ip}++;
094
095   my $text = "";
096
097   $text .= "$_\n" for @IPS;
098
099   $buf->set_text($text);
100
101   # Return true to
102   # keep watch
103   1;
104 }
105
106 ##################################
107 sub snooper {
108 ##################################
109   my($fd) = @_;
110
111   my($err, $addr, $netmask);
112
113   my $dev =
114     Net::Pcap::lookupdev(
115       \$err);
116
117   if(Net::Pcap::lookupnet(
118         $dev, \$addr,
119         \$netmask, \$err))
120   {   die "lookupnet on " .
121         "$dev failed";
122   }
123
124   my $object =
125     Net::Pcap::open_live(
126       $dev, 1024, 1, -1,
127       \$err );
128   Net::Pcap::loop(
129     $object, -1,
130     \&snooper_callback,
131     [$fd, $addr, $netmask]
132   );
133 }
134
135 ##################################
136 sub snooper_callback {
137 ##################################
138   my($user_data, $header,
139       $packet) = @_;
140
141   my($fd, $addr,
142     $netmask) = @$user_data;
143
144   my $edata =
145    NetPacket::Ethernet::strip
146     ($packet);
147
148   my $ip =
149     NetPacket::IP->decode(
150       $edata);
151
152   if((inet_aton(
153         $ip->{src_ip}) &
154       pack('N', $netmask)) eq
155         pack('N', $addr)){
156     syswrite($fd,
157           "$ip->{src_ip}\n");
158   }
159 }
```