Implementação de um Protocolo Multicast no Kernel GNU/Linux

Alan B. Menegotto, Marinho P. Barcellos

Universidade do Vale do Rio dos Sinos (UNISINOS)
São Leopoldo – RS – Brasil

menegotto@terra.com.br, marinho@acm.org

Abstract. *Multicast allows the efficient transmission of data from one machine to many. Although IP multicast exists since early 90s, few reliable multicast protocols have been actually implemented. Up to now, no reliable multicast protocol has been implemented in the kernel of a GNU/Linux system. This paper describes the early stages towards the kernel-level implementation of TCP-XM for GNU/Linux. Reliable multicast protocols are inherently complex; protocol development on a network subsystem requires previous knowledge about the protocol itself, the development tools, the kernel network subsystem and the protocol internal data structures.*

Resumo. *Multicast permite a transmissão eficiente de dados de uma máquina para várias. Apesar de multicast no IP existir desde o início da década passada, poucos protocolos para transmissão confiável de dados foram realmente implementados. Não existe, até hoje, nenhum protocolo multicast confiável nativamente no kernel GNU/Linux. Este artigo descreve a implementação, em estágio inicial, do protocolo de transmissão multicast confiável TCP-XM no kernel GNU/Linux. Protocolos multicast confiável são inerentemente complexos; a implementação requer um estudo prévio sobre o protocolo, as ferramentas que serão utilizadas, o funcionamento do subsistema de rede e as estruturas de dados do kernel.*

1. Introdução

O **TCP** (Transmission Control Protocol) [7] é o principal protocolo de transporte da Internet. O TCP oferece uma forma de comunicação confiável, orientada a conexões e com entrega seqüencial de bytes entre um transmissor e um único receptor. Para transmitir os mesmos dados simultaneamente para várias máquinas utilizando o protocolo TCP, é necessário transmitir múltiplas cópias destes dados, uma para cada máquina receptora. Este método não é eficiente quando temos um grande volume de dados para transmitir. **IP Multicast** é o método de transmissão eficiente para transferências 1→N que não necessitam de garantia na entrega dos pacotes. Apesar disso, não é raro encontrar hoje em dia redes sem suporte a IP multicast.

Multicast confiável [3] é a forma de transmissão utilizada quando é necessário garantir a entrega dos pacotes enviados de um transmissor para vários receptores de maneira eficiente. Existem diversos protocolos que implementam multicast confiável, mas não é possível utilizar um protocolo de multicast confiável quando não há conectividade IP multicast entre transmissor e receptores pois os pacotes não chegariam até os mesmos.

O **TCP-XM** [4] é um protocolo de multicast confiável que deriva do TCP e funciona utilizando um método híbrido de transmissão multicast e unicast. A transmissão multicast é utilizada para garantir o desempenho necessário, mas caso IP multicast não esteja disponível, é possível enviar dados para múltiplos receptores através de múltiplas conexões unicast.

Apesar de multicast no IP existir desde o início da década passada, poucos protocolos para transmissão confiável de dados foram realmente implementados. Neste aspecto, TCP-XM consiste uma das notórias (e recentes) exceções. No entanto, assim como o TCP-XM, os protocolos multicast confiável tem sido implementados em nível de usuário. Por não estarem no kernel do sistema, tais implementações tem desempenho limitado e problemas de compatibilidade com os sistemas onde são instalados. Não existe, ainda hoje, nenhum protocolo multicast confiável nativamente implementado no kernel GNU/Linux. Este trabalho objetiva proporcionar a primeira implementação nativa de multicast confiável no kernel do Linux. Os benefícios de multicast confiável foram extensivamente descritos na literatura, e portanto se ter multicast confiável como parte integrada do sistema operacional "out-of-the-box", são evidentes.

O trabalho está organizado da seguinte forma. As Seções 2 e 3, respectivamente, apresentam um breve resumo sobre o protocolo TCP-XM e sobre as ferramentas que serão utilizadas no desenvolvimento. As Seções 4 e 5 descrevem a análise do código-fonte do subsistema de rede do kernel GNU/Linux, com enfoque principal na área de protocolos de transporte (DCCP [8], SCTP [9] e TCP) e nas estruturas básicas de um protocolo. A partir desta análise foi possível determinar quais alterações no kernel GNU/Linux devem ser realizadas para a implementação do protocolo TCP-XM. A estratégia de implementação que está sendo utilizada e o estágio atual do desenvolvimento podem ser encontrados na Seção 6.

2. O Protocolo TCP-XM

O TCP-XM [4] é um protocolo de multicast confiável "sender-initiated" projetado para transmissão de grande quantidade de dados a um pequeno número de receptores. [1] A grande vantagem do protocolo TCP-XM sobre os demais protocolos que emulam multicast sobre unicast (M/TCP, TCP/SMO e TCP-M por exemplo) é a possibilidade de utilizar IP multicast nativo caso exista suporte na rede. [3]

A transmissão de dados utilizando o protocolo TCP-XM é muito similar a uma sessão TCP unicast comum. [2] As únicas diferenças são os mecanismos adicionados para detectar qual a melhor tecnologia de transmissão e para realizar a retransmissão de pacotes (caso ocorram perdas). [4]

Uma sessão TCP-XM pode ser brevemente resumida nos seguintes passos: [4]

1. O transmissor conecta-se aos endereços unicast dos receptores;
2. Um grupo multicast é escolhido randomicamente pelo transmissor ou por sugestão dos receptores;
3. É criada uma TCP PCB para cada receptor. Ocorre então um 3-way-handshake para cada receptor, como no início de uma conexão TCP comum.
4. Os dados que serão transmitidos são replicados em todas as filas de transmissão, localizadas dentro das TCP PCBs.
5. Durante a primeira transmissão os pacotes são enviados utilizando unicast e

multicast simultaneamente. Os pacotes multicast possuem tipo de protocolo TCP mas seu destino é o endereço multicast escolhido. A transmissão unicast será cancelada assim que a transmissão multicast ocorrer com sucesso.

6. Nas transmissões subseqüentes será utilizado o melhor método de transmissão disponível em cada receptor, de forma a aumentar a eficiência da transmissão. Caso necessário, a retransmissão de pacotes é realizada via unicast.
7. A sessão é encerrada com o envio de um pacote FIN a todos os receptores.

A detecção do suporte a IP multicast é realizada através da coleta de informações pelo receptor sobre os últimos `n` (128 por padrão) pacotes recebidos via multicast. Esta informação é contabilizada e enviada ao transmissor dentro do campo de opções dos pacotes de confirmação de recebimento (ACKs). [4]

3. Ferramentas Utilizadas

O desenvolvimento de novas facilidades no kernel de um sistema operacional é um processo que requer um ambiente isolado e protegido para testes. Este isolamento pode ser obtido com o uso de máquinas virtuais.

Existe atualmente uma grande quantidade de soluções e categorias de máquinas virtuais. A máquina virtual que está sendo utilizada nesta implementação é a QEMU (http://www.qemu.org). Esta máquina virtual possui facilidades para a depuração e para a realização de testes com o kernel GNU/Linux.

Além do ambiente para execução de testes é necessário um software para gerenciamento da árvore de desenvolvimento do kernel GNU/Linux empregada. A ferramenta utilizada por grande parte dos desenvolvedores atuais do kernel GNU/Linux chama-se Git (http://git.or.cz/).

O Git possui uma série de facilidades para o desenvolvimento distribuído de software. Existem funções para controle de versão, manipulação de alterações, atualização automática da árvore de desenvolvimento, formatação automática para submissão de patches, dentre outras.

4. Subsistema de Rede do Kernel GNU/Linux

O subsistema de rede do kernel GNU/Linux é o componente responsável pela implementação dos diversos protocolos e pelas demais facilidades fornecidas pelo kernel para comunicação.

O kernel GNU/Linux utiliza a abstração de Sockets BSD [10] para realizar a comunicação entre o contexto do usuário e o subsistema de rede, facilitando o desenvolvimento de novas aplicações.

A arquitetura interna do subsistema de rede do kernel GNU/Linux é estruturada em camadas, lembrando o modelo OSI.

A estrutura básica para comunicação entre as diversas camadas do subsistema de rede chama-se `sk_buff` [5]. O `sk_buff` foi criado para facilitar, padronizar e otimizar esta comunicação.

Existe também a estrutura `socket`, que forma a base para implementação da abstração de sockets BSD no kernel GNU/Linux [10]. A estrutura `socket` possui ponteiros para diversas estruturas, dentre elas a estrutura `sock`, responsável pela

abstração das operações realizadas pelos protocolos da familia `AF_INET`.

As estruturas descritas acima servem basicamente como unidade para transmissão e recepção de pacotes pelos protocolos da familia `AF_INET` no subsistema de rede do kernel GNU/Linux.

5. Arquitetura de um Protocolo de Rede no Kernel GNU/Linux

A implementação de um protocolo no subsistema de rede do kernel GNU/Linux exige a declaração de estruturas básicas, que definirão dentre outras coisas qual o tipo do protocolo, qual a sua família e quais funções responderão por eventos associados ao protocolo. São elas:

- `struct proto_ops`: estrutura de ponteiros para funções que responderão pelos eventos do protocolo. Estas funções fazem a ligação entre o socket e a camada AF_INET.
- `struct proto`: estrutura de ponteiros para as funções do protocolo que fazem a ligação entre a camada AF_INET e o protocolo de transporte real.
- `struct net_protocol`: estrutura de ponteiros para as funções de recepção do protocolo de transporte que é utilizada na comunicação com a camada IP. Esta estrutura é utilizada somente para inserção de protocolos na pilha TCP/IP.
- `struct inet_protosw`: estrutura que define um protocolo de transporte. Esta estrutura contém dentre outras informações ponteiros para as estruturas `proto_ops` e `proto`.
- `struct net_proto_family`: estrutura que determina qual a família do protocolo e qual a função de criação de sockets que o protocolo utiliza.

A correta declaração e inicialização destas estruturas, juntamente com a alocação de memória para as estruturas básicas do protocolo garantem a inicialização do protocolo na pilha TCP/IP do subsistema de rede.

As estruturas básicas para o funcionamento de um protocolo que não pertença a família AF_INET são a `struct proto_ops`, a `struct net_proto_familily` e a `struct proto`.

6. Implementação Inicial do Protocolo TCP-XM

Na implementação de um protocolo de rede existem tarefas que obrigatoriamente deverão ser realizadas. Em resumo podemos citar:

- Criação da infra-estrutura para funcionamento do protocolo no subsistema de rede (estruturas de dados e inicialização do protocolo);
- Criação de mecanismos para comunicação entre o programador e o protocolo;
- Implementação das funções que serão utilizadas pelo protocolo;

Durante a realização das tarefas acima devem ser desenvolvidos também programas no contexto de usuário que serão utilizados em testes durante e após a implementação do protocolo.

O processo de desenvolvimento do protocolo TCP-XM iniciou com uma duplicação das funções e estruturas do protocolo TCP que não podem ser

compartilhadas. Esta foi a estratégia de desenvolvimento utilizada na implementação dos protocolos SCTP [9] e DCCP [8], que também são protocolos de transporte do subsistema de rede do kernel GNU/Linux.

Esta duplicação inicial seguida das devidas alterações sobre o código duplicado facilitou o início da implementação do protocolo TCP-XM. Isto deve-se ao fato do protocolo TCP já possuir uma infra-estrutura correta e funcional que foi automaticamente transferida ao protocolo TCP-XM:

- Estruturas para inicialização e inserção do protocolo na pilha TCP/IP;
- Infra-estrutura para funcionamento do protocolo (estruturas de dados e funções);
- Mecanismos para controle de congestionamento;
- Mecanismos para parametrização on-line do funcionamento do protocolo via sysctl [6];
- Mecanismos para estatística sobre o protocolo via SNMP e procfs [11];

Uma clara desvantagem desta abordagem é a duplicação excessiva de código entre protocolos de rede. A unificação de código comum entre os protocolos da pilha TCP/IP é um dos trabalhos realizados atualmente por Arnaldo Carvalho de Mello (acme) no subsistema de rede.

Após a duplicação inicial do código foi necessário alterar a estrutura `tcpxm_sock`, estrutura aonde são armazenadas as informações relativas a uma conexão TCP-XM. Foram acrescentadas variáveis para armazenar o modo de transmissão, contadores para detecção do modo de transmissão e uma lista encadeada (`klist`) que armazena os endereços unicast dos receptores de determinada sessão.

Dentro da interface sysctl foram adicionadas variáveis para parametrização do funcionamento do protocolo. Estas variáveis controlam a coleta de informações pelos receptores, a inicialização do protocolo e o modo de transmissão default.

Na chamada `setsockopt` do protocolo foi criado também um mecanismo para inclusão (`TCPXM_RCVADD`) e remoção (`TCPXM_RCVDEL`) dos endereços unicast dos receptores na lista encadeada `tcpxm_uc_group`, armazenada dentro da estrutura `tcpxm_sock`. É possível ainda via `setsockopt` configurar o grupo multicast que será utilizado durante a transmissão (`TCPXM_MCGROUP`).

Na chamada `getsockopt` foi acrescentada somente uma opção que retorna qual o grupo multicast escolhido (`TCPXM_MCGROUP`) caso algum tenha sido previamente definido pela chamada `setsockopt`.

Para realização de testes da duplicação inicial e das estruturas adicionadas, foi necessário criar programas no contexto de usuário que utilizam o protocolo TCP-XM e suas opções. Estes programas contém somente as operações básicas para manipulação, parametrização, transmissão e recepção de dados.

Resumindo, já foram realizadas a criação da infra-estrutura para funcionamento do protocolo no subsistema de rede (estruturas de dados e inicialização do protocolo) e a criação de mecanismos para comunicação entre o programador e o protocolo. Resta agora concluir a implementação das funções que responderão aos eventos do protocolo de conexão, transmissão e recepção.

7. Comentários Finais

Até o presente momento toda a infra-estrutura para o funcionamento do protocolo foi implementada. Essa implementação envolveu estudos, experimentos com ferramentas e a duplicação do código do protocolo TCP que não pode ser compartilhado com o protocolo TCP-XM. Também já foi realizada a modificação das rotinas de 3-way-handshake para realização da negociação inicial entre transmissor e receptores.

Para finalizar a implementação restam ainda algumas tarefas:

- Implementar o estabelecimento e encerramento de conexões $1 \rightarrow N$;
- Implementar as rotinas de transmissão e recepção do protocolo TCP-XM;
- Implementar os mecanismos de retransmissão do protocolo TCP-XM;
- Implementar as rotinas de controle de congestionamento do protocolo TCP-XM;

Após a finalização da implementação serão realizados experimentos para avaliar qual o ganho obtido com a implementação do protocolo TCP-XM no kernel GNU/Linux.

8. Bibliografia

[1] Jeacle, K. et al. (2005). "Hybrid Reliable Multicast with TCP-XM", In: 1st International Conference on Emerging Networking Experiments and Technology, França.

[2] Jeacle, K., Crowcroft, J. (2005). "TCP-XM: Unicast-enabled Reliable Multicast". In: 14th International Conference on Computer Communications and Networks, EUA.

[3] PAUL, S. (1998). "Multicasting on the Internet and Its Applications". Kluwer Academic Publishers, 1st edition.

[4] Jeacle, K. (2005). "TCP-XM". 131 p. Dissertação (Doutorado) - Wolfson College, University of Cambridge.

[5] Cox, A. (1996). "Network Buffers and Memory Management", Linux Journal, http://www.linuxjournal.com/article/1312.

[6] Rubbini, A. (1997). "The sysctl Interface", http://www.linuxjournal.com/article/2365.

[7] Defense Advanced Research Projects Agency (1981). "Transmission Control Protocol (RFC 793)", IETF.

[8] Kohler, E.; Handley, M.; Floyd, S. (2005). "Datagram Congestion Control Protocol (Internet Draft draft-ietf-dccp-problem-03)", IETF.

[9] Stewart, R. et al. (2000). "Stream Control Transmission Protocol (RFC 2960)", IETF.

[10] Beck, M. et al. (2002). Linux Kernel Programming, Addison Wesley, 3rd Edition.

[11] Mouw, E. (2001). "Linux Kernel Procfs Guide", http://www.kernelnewbies.org/documents/kdoc/procfs-guide/lkprocfsguide.html.