SMAPP : Towards Smart Multipath TCP-enabled APP

0. Abstract

- MPTCP는 TCP의 대체물로 설계 및 구현되었다. MPTCP는 다른 경로의 이용을 통제할 수 없는 표준 Socket API를 App에 노출시킨다. 이것은 네트워크의 다중 경로 특성을 모르는 응용 프로그램에서 주요 기능이다. 그러나 이는 특정 분야에서 성능을 높일 수 있는 App에게는 제한 사항이 된다. 이에 우리는 App에게 경로 관리를 위임하는 MPTCP Path Manager(PM)를 제안한다. 이 PM을 통해 App은 MPTCP 경로를 제어할 수 있다. 우리는 Linux MPTCP 커널 위에 이 PM을 구현하였다. 이것은 Kernel에 구성되어 Events와 commands를 User Space App에 드러낸다. 그리고 MPTCP의 Subflow 생성/제거, 재전송 응답과 같은 주요기능을 컨트롤한다. 또한, 우리는 다양한 사용 사례를 통해 이 PM의 이점을 보여준다.

1. Introduction

- TCP는 오늘날 인터넷에서 주요한 프로토콜이다. 이것은 신뢰 있는 ByteStream을 제공하고 이것은 스마트폰에서 데이터센터까지 광범위하게 사용되고 있다. 그러나, TCP가 대중적임에도 불구하고 계속해서 진화한다. MPTCP는 TCP의 가장 최근의 확장판이다. 이것은 TCP의 기본 설계 가정 중 하나를 완전히 변경하였다 => TCP 연결은 항상 4개-tuple(IP-Address and Port of source & destination)로 확인. 모든 패킷들은 항상 4개의 Tuple로 묶여진 Connection과 같은 곳으로 보내어진다. 예를 들어 Cellular, WIFI가 존재하는 스마트폰에서, 이것은 하나의 TCP 연결에 속한 패킷을 다른 인터페이스/주소로 보내는 것은 불가능하다. => (기존TCP의 문제점)

- MPTCP는 하나의 TCP 연결에 속한 패킷을 다른 인터페이스/주소로 보내지 못하는 문제를 해결했다. => (기존 TCP의 문제점을 MPTCP가 해결) MPTCP의 구현은 cellular와 WIFI인터페이스가 장착된 스마트폰과 같은 모바일 기기부터 대규모 데이터 센터까지 다양한 사용 사례를 유발하였다. TCP 확장인 SA, Timestamp, Window Scale 옵션을 오랜 기간동안 구축하는데 몇 년이 걸렸지만, MPTCP는 빠르게 주요 운영 체제의 한부분으로 채택되었다. 2013년 9월 이 후, 모든 Apple의 스마트폰과 태블릿은 Siri Voice Recognition Application에 MPTCP를 사용한다. 2015년 7월에 KT에서는 fast LTE와 fast WIFI를 스마트폰과 결합한 상업적인 MPTCP를 발표하였다. 2015년 9월에는 ~~~ => MPTCP에 대한 여러 사례들

- MPTCP는 3개의 호환성 목표를 메인으로 염두하고 있다. **첫번째1)** 기존 Socket API를 통해 기존 App에서 MPTCP를 사용할 수 있어야 한다는 것이다. 리눅스 커널의 MPTCP 구현은 App이 MPTCP를 사용하도록 허용함으로써 첫번째 목표를 충족시킨다(The Multipath TCP implementation in the Linux kernel meets this objective by allowing any application to use Multipath TCP.). **두번째2)** MPTCP는 반드시 현재 구축된 Network와 호환성이 유지되어야 한다. MPTCP 설계자가 기존의 미들박스를 처리할 수 있도록 다양한 메커니즘 프로토콜을 포함시켜 두번째 목표를 충족시켰다. **세번째3)** 네트워크의 또 다른 유저와 공정해야 한다는 것이다. 이것은 몇 몇의 Congestion Control 스키마의 구현 및 설계를 통해 충족시켰다.

- 이 문서에서 우리는 MPTCP 워킹 그룹의 첫번째 호환성 목표를 재방문/검토(revisit)하였다. 우리는 App이 **일반 바이트 스트림 서비스**만 필요하다고 가정하고, **App이 적응적이며 MPTCP를 통해 최상의 결과를 얻기를 원한다는 가정**으로 시작한다. 우리는 Smart App이 스마트폰과 아마도 데이터 센터에서도 처음 구현될 것이라 기대한다.

- 이 문서는 다음과 같이 구성된다. 처음에는 간단하게 MPTCP와 관련된 것을 Section 2에서 설명한다. 그런 다음 MPTCP Control과 Data Plane을 분리할 것을 제안(Section 3)한다. Data Plane은 예를 들어 데이터 전송과 관련 있는 기능, Control Plane은 예를 들어 User Space에서 서로 다른 Path 관리와 같은 기능이다. 그런 다음 우리는 이러한 접근 방법의 이점을 Section 4에서 4개의 사용 사례로 설명한다.

2. Multipath TCP

- MPTCP의 시그널링 설명
  - MP_CAPABLE, MP_JOIN, etc.

- MPTCP Path-Manager 설명 (Linux 커널에 포함되어 있는 PM 위주 설명)
  - Full-mesh : 여러 인터페이스를 통해 서버를 향한 각 각의 Subflow를 생성한다.
  - ndiffport : 초기 Connection에 사용된 인터페이스와 동일한 인터페이스를 통해 n개의 subflow를 생성한다. 이 PM은 ECMP(Equal Cost Multipath)로 로드 밸런싱된 경로를 활용할 수 있는 데이터 센터를 위해 설계되었다.

➢ 위의 두 PM은 오직 Client 측에서만 사용되는 PM이다. 그 이유는 Client는 연결 시도를 차단하는 NAT와 방화벽 뒤에 있는 경우가 많기 때문.

3. The Subflow Controller

- Linux MPTCP는 모두 Linux 커널에 위치한다. MPTCP의 커널 코드는 대부분 데이터의 전송과 수신에 전념한다. 또한, Subflow의 관리도 full-mesh 와 ndiffports로 불리는 PM에 의해 커널에서 수행된다. 이러한 설계는 성능상의 이유로 진행되었다. **만약, Application에서 Subflow를 제어를 원한다면 이것은 Application에게 불행한 결과이다.** **=> (subflow 제어를 Kernel에서만 수행할 수 있음으로)** 이것은 좋지 않은 솔루션이며, 오직 3개의 PM만이 몇 년 동안 커널에 구현되어 있다는 것도 좋지 않다.

- 우리는 MPTCP의 Data와 Control Plane을 서로 별개로 나누는 방법을 사용해 앞선 설계들을 재검토하였다.
  Data Plane은 데이터의 전송과 같은 모든 기능을 포함한다. 이것은 성능상의 이유로 커널에 남겨두었다.
  => (packet scheduler에 관련한 것은 본 문서에서 제외)
  Control Plane은 Subflow 관리 및 MPTCP Connection 구성과 같은 모든 기능 포함한다. **성능상의 관점에서 볼 때, 이러한 기능을 커널에 배치할 이유가 없다. 더 나아가, 몇 몇의 Application들은 Subflow 관리를 위해 복잡한 정책을 구현하기를 원할 수 있다.**
  => (Application위에 subflow manager를 올려도 문제가 되지 않을 것임을 설명)
  이러한 종류의 코드는 실제로 커널 내부에 위치하기에는 부합하지 않다.
  Application이 MPTCP Kernel code와 상호작용할 수 있도록 우리는 새로운 Netlink family를 정의하였다. Netlink는 IPC(InterProcess Communication) 메커니즘이며, App과 Kernel 사이에 메시지를 주고받도록 하는 것을 지원한다. 이것은 M. Coudron이 제안한 접근법과 유사하다.

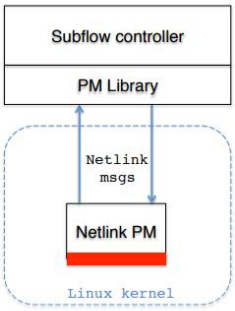


Figure 1: The subflow controller and the `Netlink` path manager

- 그러나, 코드 작성과 Netlink 이벤트를 보내거나 수신하는 것은 App 개발자에겐 복잡할 수 있다. 우리는 라이브러리 내의 모든 Netlink 핸들링의 복잡한 것들을 가상화 시킨다. 이것은 Subflow Controller 실행과 관련된 부분 전부를 Userspace측으로 연결시켜 Subflow Controller 구현을 안심시킨다. 이 라이브러리(Figure 1)는 커널의 일부분인 Netlink PM과 상호작용한다. 이 PM은 Netlink를 통해서 커널 내에 존재하는 PM Interface(Figure 1의 빨간색으로 칠해진 부분)를 사용하고 노출시킨다. PM은 1100줄의 C코드로 구현되었고 라이브러리는 1900줄로 구현되었다. Netlink PM은 커널 내의 이벤트와 상태 정보를 노출시키는 유연한 API를 제공한다. Callback은 Subflow Controller에서 MPTCP 커널에 명시된 Event가 발생하였을 때 트리거되어 제공된다. Subflow Controller는 오직 MPTCP 커널에 등록되어 있는 Event만 수신한다.

- Netlink PM은 Connection, Subflow(s), Type of Event, 등에 대한 정보가 포함된 메시지를 전송하고 수신한다. 이것은 M. Coudron의 초기 프로토타입보다 더 많은 이벤트와 커맨드를 제공한다. ***created*** 이벤트는 MPTCP Connection이 Established 될 때 트리거 된다. 이것은 4개의 Tuple(Subflow의 초기 번호와 그 외 Connection 식별에 필수 되는 정보)을 포함한다. ***estab*** 이벤트는 3way-handshake가 성공했을 때 트리거 되고, ***closed*** 이벤트는 MPTCP Connection의 종료가 되었을 때 트리거 된다. 이 이벤트들을 통해 PM은 App에서 설정한 Connection을 관리할 수 있다.

- ***add_addr*** 과 ***rem_addr*** 이벤트는 MPTCP Option을 가진 원격 호스트에 의해 IP 주소를 알려주고 제거한다. 이러한 이벤트들 덕분에 Subflow Controller는 원격 호스트의 주소를 저장할 수 있고 필요한 경우 새로운 Subflow를 설정(Establish)한다. 이것은 즉시 Subflow를 생성하는 커널 내부의 PM보다 더 유연하다. 또한 이것은 커널 내부의 각 MPTCP Connection마다 정보를 유지하는 것을 감소시킨다.

- ***sub_estab*** 과 ***sub_closed*** 이벤트는 App에서 subflow를 제어하는 것을 가능하게 한다. ***sub_estab*** 이벤트는 새로운 Subflow가 설정(Established)되면 트리거 된다. 서버는 이 이벤트로 Subflow의 개수를 제한할 수 있다. ***sub_closed*** 이벤트는 한 Subflow에서 과도한 재전송으로 인해 종료되었을 때 **RST** 세그먼트가 송신되었으면 트리거 된다. 또한 이 이벤트는 제거(Ex. RTO의 지나친 만기, 목적지에 도달할 수 없는, 등)를 위한 이유를 나타내는 에러코드(standard **errno**)와 관련 있다.

- 마지막은 ***timeout*** 이벤트이다. TCP에서 재전송 타이머 만기는 보통 심각한 loss임을 표시한다. 일반 TCP에서는 재전송 타이머가 너무 자주 만기되

면 App에서 할 수 있는 일이 없다. MPTCP의 경우, 두 번째 경로가 현재 경로와 매우 다른 손실 특성을 가질 수 있기 때문에 앞선 상황과 다르다. 이 이벤트는 재전송 타이머의 값을 보도하고 트리거로 사용하여 추가 Subflow를 생성하는 데에 사용할 수 있다.

- 또한 Netlink PM은 로컬 IP주소가 사용 가능(***new_local_addr***) 또는 사용 불가능(***del_local_addr***)일 때 인터페이스에 의해 트리거되는 이벤트도 수집한다.

- 이 외에도, 라이브러리는 Subflow Controller가 커맨드를 통해서 MPTCP Connection의 상태를 수정하는 것을 허용한다. 우리의 초기 구현은 몇몇의 커맨드 타입을 지원한다. 첫번째, 이것은 Subflow의 생성에 대한 요청을 가능케한다. Controller는 임의의 4-Tuple(IP 주소 와 Port)을 기반으로 한 Subflow 생성을 할 수 있다. 유사한 명령을 사용하면 설정된 하위 흐름(컨트롤러를 통해 생성되거나 피어에 의해 설정됨)을 제거할 수 있습니다. 이를 통해 Subflow Controller가 Subflow를 쉽게 조정할 수 있습니다. Controller는 MPTCP Connection의 Control 블록이나 Subflow 중 하나에서 정보를 검색할 수도 있습니다. 실제로 이는 Linux에서 TCP_INFO 소켓 옵션을 사용하는 것과 같습니다.

4. Sample Use Cases

- 이 섹션 우리는 여러 다른 사례를 통해 어떻게 **Smart Application이 MPTCP와 지능적으로 상호작용하는지에 대한 시범을 보여 User Space의 Subflow Controller에 대한 이점을 설명**한다.

4.1. Smart long-lived connections

- 스마트폰 위 "**ssh**와 같은 몇몇의 App", "다양한 채팅 App", "알림 App"은 오랜 시간 혹은 하루 동안 Connection을 유지하는 Long-Lived Connection을 사용한다. 이러한 Connection은 각 Established Connection의 상태를 유지하는 Firewall과 NAT와 같은 Middleboxes를 포함하는 네트워크에서 운영상의 문제를 가진다. 한 예로 Established 되었지만 최근에 데이터를 전송하지 않은 연결이 있다. 많은 NAT와 Firewall은 몇 시간 뒤 해당 상태의 Connection의 상태를 Drop 시킬 것이다. IETF가 2시간 4분 이상의 timeout을 권장하지만, 구현된 많은 NAT와 Firewall은 수백 초 후에 사용되지 않은 상태를 제거합니다. 더욱이, 많은 네트워크들은 그러한 Middleboxes을 계단식으로 포함한다. 몇 몇의 App은 Established된 각 Connection에 정기적으로 데이터를 전송합니다. 예를 들어 RFC3948에서는 Keepalive Packet을 매 20초마다 전송하는 것을 권장한다. Middleboxes와 App간의 이 싸움의 불행한 결과는 모바일 기기가 단순히 Middleboxes 내의 Established TCP Connection을 위한 상태를 보존하기 위해 많은 에너지를 소비해야 한다는 것이다. 디바이스에 주어진 중요한 에너지를 소비하는 이러한 접근법은 좋지 않다.

- 우리의 첫번째 Subflow Controller는 MPTCP Linux Kernel에 존재하는 fulllmesh PM의 재실행이다. **이 Controller는 C코드 800라인의 User Space에서 실행된다.** 이것은 Section 3에 기술된 모든 Event를 위한 Listener로 실행된다. 이것은 커널 내 PM과 같이 로컬 주소 활성화 및 비활성화 이벤트인 ***new_local_addr***과 ***del_local_addr***에 반응한다. 또한, 이것은 어떠한 Subflow의 실패인 ***sub_closed*** 이벤트에도 반응한다. **이러한 이벤트**(new_local_addr, del_local_addr, sub_closed(한 Subflow에서 과도한 재전송으로 인해 종료되었을 때 RST 세그먼트가 송신되었으면 트리거 된다.))**가 일어날 때, Subflow Contoller는 에러 조건(과도한 Timeout, RST, ICMP 메시지 수신, 등)을 분석한 뒤 반응한다. 에러 조건에 기반하여 이것은 실패한 Subflow를 다시 ReEstablish하고 다른 Timeout 설정을 시도한다.** 이 Controller에 대한 실험은 어려운 네트워크 조건에서도 서로 다른 경로에 걸쳐 Established된 Subflow를 올바르게 유지할 수 있다는 것을 보여준다.

4.2. Smarter backup

4.3. Smarter streaming

4.4. Smarter exploitation of flow-based LB

4.5. User space path manager performances

- 사용자 공간 PM의 CPU 비용에 대한 첫 번째 평가로서, 우리는 직접 1Gbps 이더넷 링크로 연결된 두 호스트 사이에서 간단한 실험을 수행한다. 서버는 2.53GH 및 8GB의 Intel(R) Xeon(R) CPU X3440이고, 클라이언트는 2.33GHz 및 4GB 메모리에 Intel(R) Core(TM)2 Duo CPU E6550을 사용합니다. 서버는 ***lighttpd HTTP*** 웹 서버와 기본 커널 내 PM을 실행합니다. 클라이언트는 1000개의 512KB 파일로 HTTP/1.0 GET 쿼리를 연속적으로 수행합니다. 이 실험은 ***ndiffports*** 경로 관리자의 두 가지 변형인 사용자 공간과 커널에서 수행된다. 이 두 경로 관리자는 초기 하위 흐름이 설정되는 즉시 두 번째 하위 흐름을 만듭니다. 우리는 초기 Subflow의 SYN(즉, **MP_CAPABLE** 옵션 포함)과 두 번째 Subflow의 SYN(즉, **MP_JOIN** 옵션 포함) 사이의 Delay를 측정한다. Figure 3은 두 개의 서로 다른 경로 관리자를 사용하여 측정된 Delay에 대한 CDF를 제공합니다. 커널 내 PM이 사용자 공간 PM보다 약간 더 빠르다는 것을 보여준다. **평균적으로 사용자 공간 PM은 Delay Time을 23마이크로초까지 늘어납니다(커널 PM에 비해서 23Ms 높음). 이 추가 Delay는 작으며 클라이언트에서 허용될 수 있습니다** => Figure 3. 우리는 또한 추가 프로세스를 실행하여 클라이언트의 CPU를 강조하는 실험을 수행했다. 이 경우 커널 내 및 사용자 공간 PM 모두 영향을 받습니다. 사용자 공간 PM로 인한 Delay 증가는 37마이크로초 미만으로 유지됩니다.

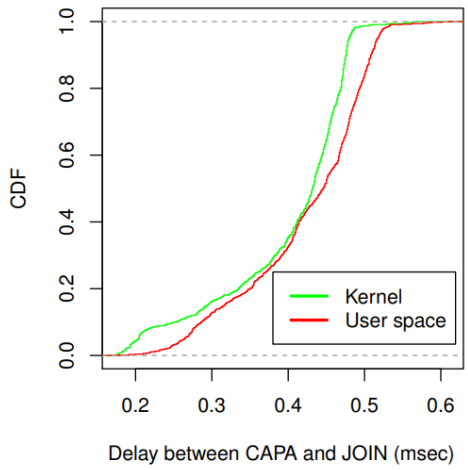


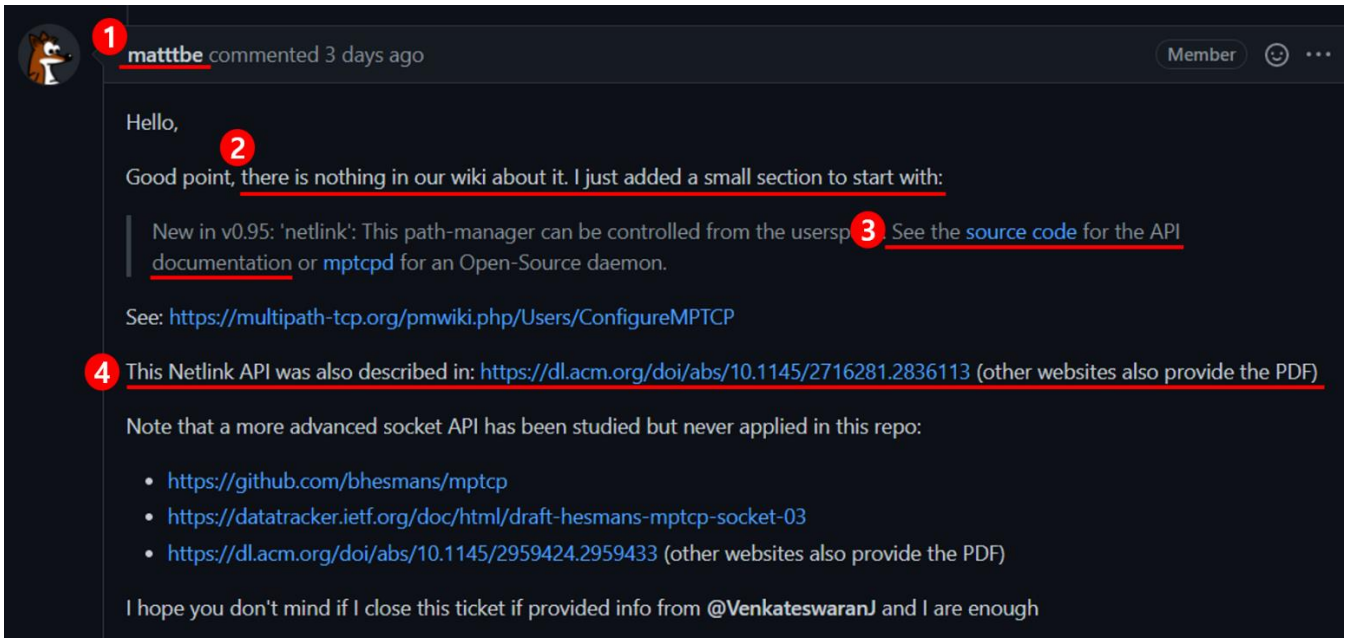
Figure 3: Kernel path manager is slightly faster than user space path manger to open a second subflow

5. Discussion

- MPTCP는 App에서 수정할 수 없다는 가정하에 설계되었다. 이러한 이유로 수정할 수 없는 Socket 인터페이스를 App에 노출시켰다. 이러한 설계 가정은 MPTCP의 배치를 용이하게 하는 핵심이었다. 이제 MPTCP 사용자와 개발자는 Multipath 기능을 더 잘 활용할 수 있는 방법을 모색한다. 우리가 제안한 Subflow Controller는 이러한 문제를 커널이 많은 상태를 유지하고 복잡한 정책을 처리하지 않아도 될 수 있게끔 해결하였다. 우리의 설계는 MPTCP의 Control과 Data Plane의 분리의 첫번째 단계이다. Control Plane은 subflow의 관리에 관한 모든 기능을 포함한다. 반면에 Data Plane은 데이터 송수신 관련된 모든 기능을 포함한다. 이 설계에서는 다음 가정을 유지한다. 네트워킹(Netlink PM) 스택의 메인 파트는 커널에서 실행된다. 실행 관점에서, 이것은 우리가 리눅스 커널에 최적화된 TCP와 MPTCP를 활용할 수 있게 한다. 그러나, 이것은 App 개발자가 MPTCP의 진화에 의존한다는 것 또한 의미한다. 이 대안으로 다양한 연구자들에 의해 제안된 대로 User Space로 네트워킹 스택을 이동하는 것이다. 이것(SMAPP)으로 새로운 기능의 배치와 프로토콜 확장을 쉽게 할 수 있을 것이다. 그러나 이것은 각 App 개발자가 그 만의 스택을 개발할 경우 불필요한 확장을 야기할 수도 있다. 장기적으로 봤을 때, 우리는 우리의 접근 방식과 User Space 네트워킹 스택의 성공이 App 개발자들이 쉽게 사용할 수 있는 고성능 라이브러리의 가용성에 달려 있을 것이라 기대한다.

6. 현재 Netlink PM 구현에 관한 설명

6.1. 해당 내용은 "Netlink PM에 관한 API 문서가 존재하나요?" 라는 질문에 대한 답변에 관한 것이다.



=> https://github.com/multipath-tcp/mptcp/issues/430 발췌

① MPTCP Linux 개발자

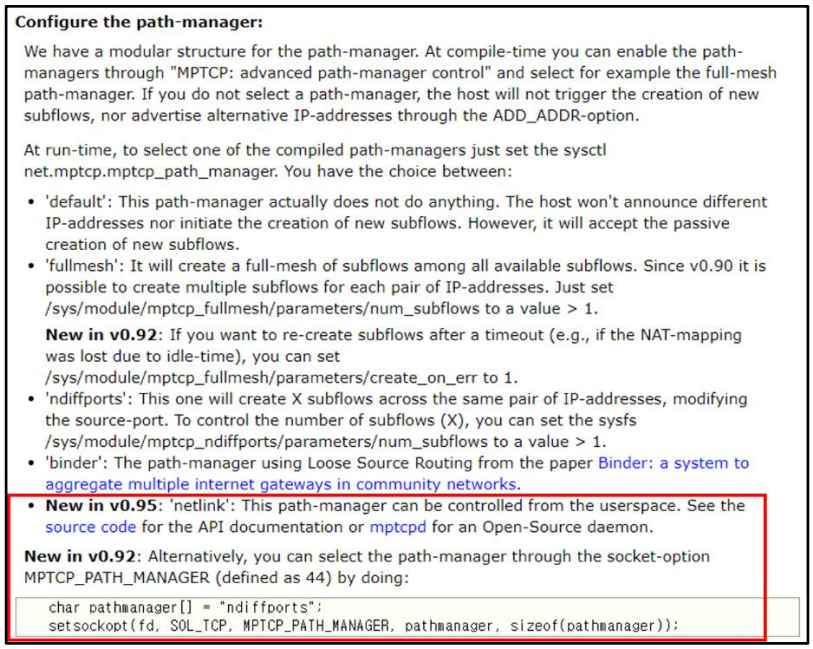
② wiki(불특정 다수가 협업을 통해 직접 내용과 구조를 수정할 수 있는 웹사이트를 일컫는다.)에는 아직 그것(Netlink PM)에 대해 아무것도 없으며, 나는 단지 작은 섹션만을 추가하였다. *=> Netlink PM이 구현은 되어있지만 API를 설명하는 문서가 존재하지 않음.*

③ API 문서화를 위해 source code를 보아라.

=> source code : https://github.com/multipath-tcp/mptcp/blob/mptcp_trunk/include/uapi/linux/mptcp.h

④ 이 Netlink API는 https://dl.acm.org/doi/abs/10.1145/2716281.2836113 (= SMAPP)에서도 설명되어 있습니다.

6.2. Netlink PM이 MPTCP Linux에 구현되어 있음을 나타내는 설명



Configure the path-manager:

We have a modular structure for the path-manager. At compile-time you can enable the path-managers through "MPTCP: advanced path-manager control" and select for example the full-mesh path-manager. If you do not select a path-manager, the host will not trigger the creation of new subflows, nor advertise alternative IP-addresses through the ADD_ADDR-option.

At run-time, to select one of the compiled path-managers just set the sysctl net.mptcp.mptcp_path_manager. You have the choice between:

- 'default': This path-manager actually does not do anything. The host won't announce different IP-addresses nor initiate the creation of new subflows. However, it will accept the passive creation of new subflows.
- 'fullmesh': It will create a full-mesh of subflows among all available subflows. Since v0.90 it is possible to create multiple subflows for each pair of IP-addresses. Just set /sys/module/mptcp_fullmesh/parameters/num_subflows to a value > 1.

  New in v0.92: If you want to re-create subflows after a timeout (e.g., if the NAT-mapping was lost due to idle-time), you can set /sys/module/mptcp_fullmesh/parameters/create_on_err to 1.
- 'ndiffports': This one will create X subflows across the same pair of IP-addresses, modifying the source-port. To control the number of subflows (X), you can set the sysfs /sys/module/mptcp_ndiffports/parameters/num_subflows to a value > 1.
- 'binder': The path-manager using Loose Source Routing from the paper Binder: a system to aggregate multiple internet gateways in community networks.
- **New in v0.95**: 'netlink': This path-manager can be controlled from the userspace. See the source code for the API documentation or mptcpd for an Open-Source daemon.

New in v0.92: Alternatively, you can select the path-manager through the socket-option MPTCP_PATH_MANAGER (defined as 44) by doing:

```
char pathmanager[] = "ndiffports";
setsockopt(fd, SOL_TCP, MPTCP_PATH_MANAGER, pathmanager, sizeof(pathmanager));
```

=> http://multipath-tcp.org/pmwiki.php/Users/ConfigureMPTCP 발췌

=> C언어 Application에서 다음과 같은 방법으로 Netlink PM을 사용할 수 있음을 설명.

6.3. Netlink PM은 generic_netlink를 기반으로 개발되었음

mptcp: new generic Netlink-based PM ①

From: Matthieu Baerts <matthieu.baerts-g9H5/arvmg7k1uMJSBkQmQ-AT-public.gmane.org>
To: mptcp-dev-1cNGNKGn6cRWdXg3Zgxhqoble9XqW/aP-AT-public.gmane.org
Subject: [PATCH 0/4] mptcp: new generic Netlink-based PM
Date: Fri, 23 Mar 2018 17:02:24 +0100
Message-ID: <20180323160228.27848-1-matthieu.baerts@tessares.net>
Cc: Christoph Paasch <cpaasch-2kanFRK1NckAvxtiuMwx3w-AT-public.gmane.org>, mathew.j.martineau-VuQAYsv1563Yd54FQh9/CA-AT-public.gmane.org, ossama.othman-ral2JQCrhuEAvxtiuMwx3w-AT-public.gmane.org, stephen-YuLIe7w5MY0WenYVfaLwtA-AT-public.gmane.org, alexander-VwyScMIDjddn68oJJuIU0Q-AT-public.gmane.org, Gregory Detal <gregory.detal-g9H5/arvmg7k1uMJSBkQmQ-AT-public.gmane.org>, Matthieu Baerts <matthieu.baerts-g9H5/arvmg7k1uMJSBkQmQ-AT-public.gmane.org>

② This path-management module is controlled over a Netlink interface. A userspace module can therefore control the establishment of new subflows and the policy to apply over those new subflows for every connection.

=> https://lwn.net/Articles/750105 발췌

① 제목 : 새로운 generic Netlink 기반 Path Manager

② 내용 중 : 이 PM 모듈은 Netlink 인터페이스를 통해 제어됩니다. 따라서, Userspace 모듈은 새 Subflow의 Establishment와 모든 Connection에 대해 새 Subflow에 적용되는 정책을 제어할 수 있습니다.

6.3.1 What is generic netlink?

What is generic netlink?

Generic netlink was designed to allow kernel modules to easily communicate with userspace applications using netlink. Instead of creating a new top level netlink family for each module, generic netlink provides a simplified interface to enable modules to plug into the generic netlink bus.

=> https://medium.com/@mdlayher/linux-netlink-and-go-part-2-generic-netlink-833bce516400 발췌

- 해석
  => generic netlink는 netlink를 사용하여 userspace의 app과 kernel module이 쉽게 통신할 수 있도록 설계되었다. 각 kernel module에 대해 새로운 최상위 netlink family를 생성하는 대신, generic netlink는 kernel module이 generic netlink bus에 연결할 수 있도록 단순화된 인터페이스를 제공합니다.

- 나의 애매한 해석...
  => 기존의 netlink는 각 kernel module에 각 netlink를 배치하는 방식**(1:1)**
  => 그러나 generic netlink는 하나의 generic netlink에 여러 kernel module을 부착**(1:N)**할 수 있는 것 같다.

❖ Netlink PM 진행을 위해서는 먼저 generic netlink 사용방법을 알아야 다음 사항을 진행할 수 있을 것 같다.
  => generic netlink example 1 : https://wiki.linuxfoundation.org/networking/generic_netlink_howto
  => generic netlink example 2 : https://github.com/gilson27/generic_netlink
  => generic netlink example 3 : https://github.com/a-zaki/genl_ex

참고 문헌

- SMAPP 논문 : https://dl.acm.org/doi/abs/10.1145/2716281.2836113
- MPTCP Linux 공식홈페이지 : www.multipath-tcp.org
- MPTCP Linux 소스코드 저장소 : https://github.com/multipath-tcp/mptcp
- Netlink PM Commit 메시지 : https://lwn.net/Articles/750105
- Generic netlink에 관해 : https://medium.com/@mdlayher/linux-netlink-and-go-part-2-generic-netlink-833bce516400