TCP over TCP

Test Results

Introduction

In order to provide a secure communication mechanism over an insecure channel, we decided to tunnel all IP traffic via a *SSH* connection. The *SSH* connection is used as a pipe or tunnel and is realized with *PPP* point-to-point interfaces on each side of the tunnel. Routes are defined so that all IP traffic destined for the other side of the insecure channel is sent through the *PPP* interfaces. Thus, no changes are needed for the applications needing a secure transport mechanism.

Using this type of implementation, TCP packets are tunneled over a TCP connection.

In the document http://sites.inka.de/sites/bigred/devel/tcp-tcp.html , this approach is considered bad and unreliable, because of the two levels of TCP stacks working with the same algorithm in case of packet loss.

A private eMail communication with the author is added in the appendix and helped to build an environment in which we tried to simulate the bad lines with which the author observed the mal-behavior of this approach.

The Test Environment

In Figure 1, the topology of our *Test Environment* is shown.

We used a special instrumented *Linux* kernel in system **Wawa**, which allowed us to specified a “drop rate” for routed IP packets. By writing an integer value in the range from 0 to 1000 to */proc/sys/net/ipv4/ip_drop_rate*, we were able to simulate “bad lines” in our environment. Every time the kernel had to route a packet, a random number was generated (range 0 – 1000). If this random number was equal or below the specified “drop rate”, we dropped the packet.

The figure lists the interfaces on each system, and the routes which were installed.

Our test was made by using *ftp* on system **Juno** to transfer a 10MB file to system **Tecra**. We took the average time needed to accomplish five transfers as the result of a test step. After each test step, we incremented the “drop rate” on **Wawa**, starting with 0 and incremented by 10.

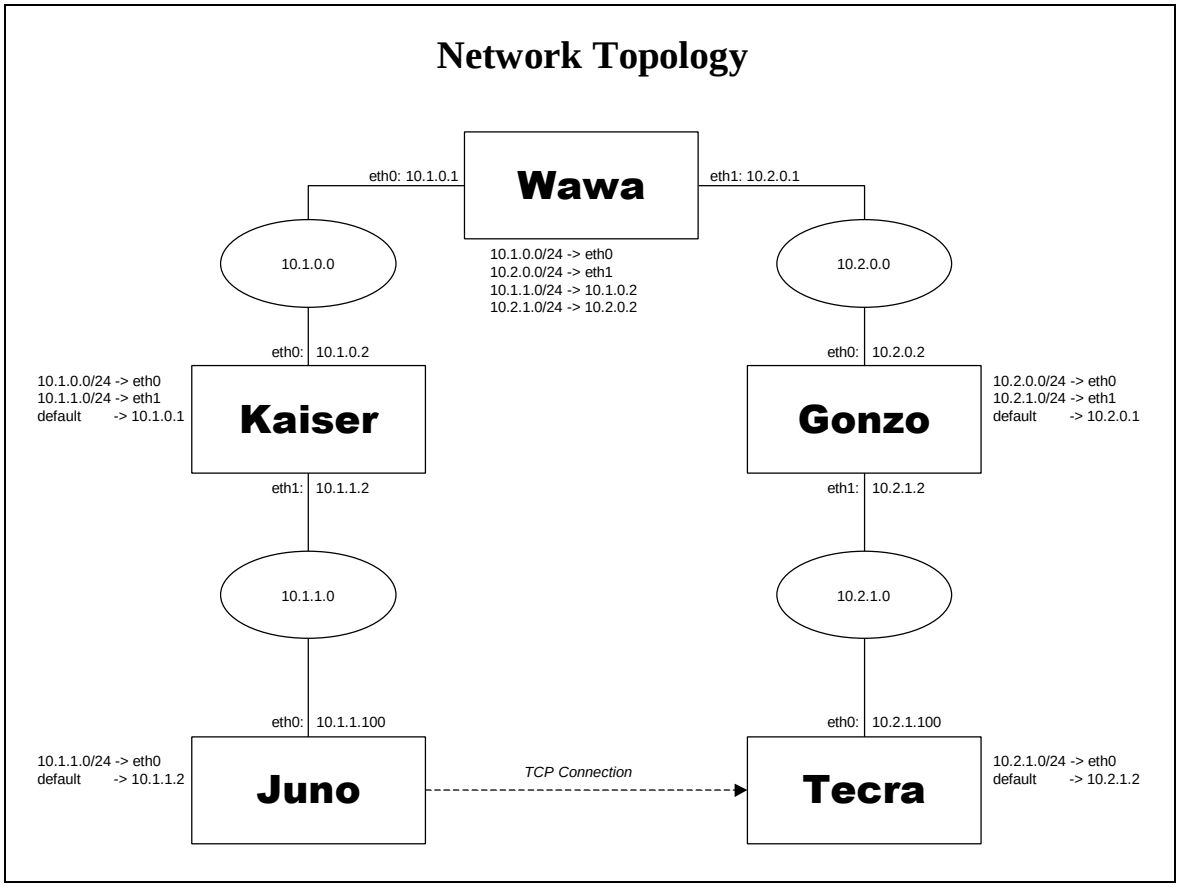

Figure 1: The Test Environment

In Figure 2, the topology of our *Secure Virtual Private Network* is shown.

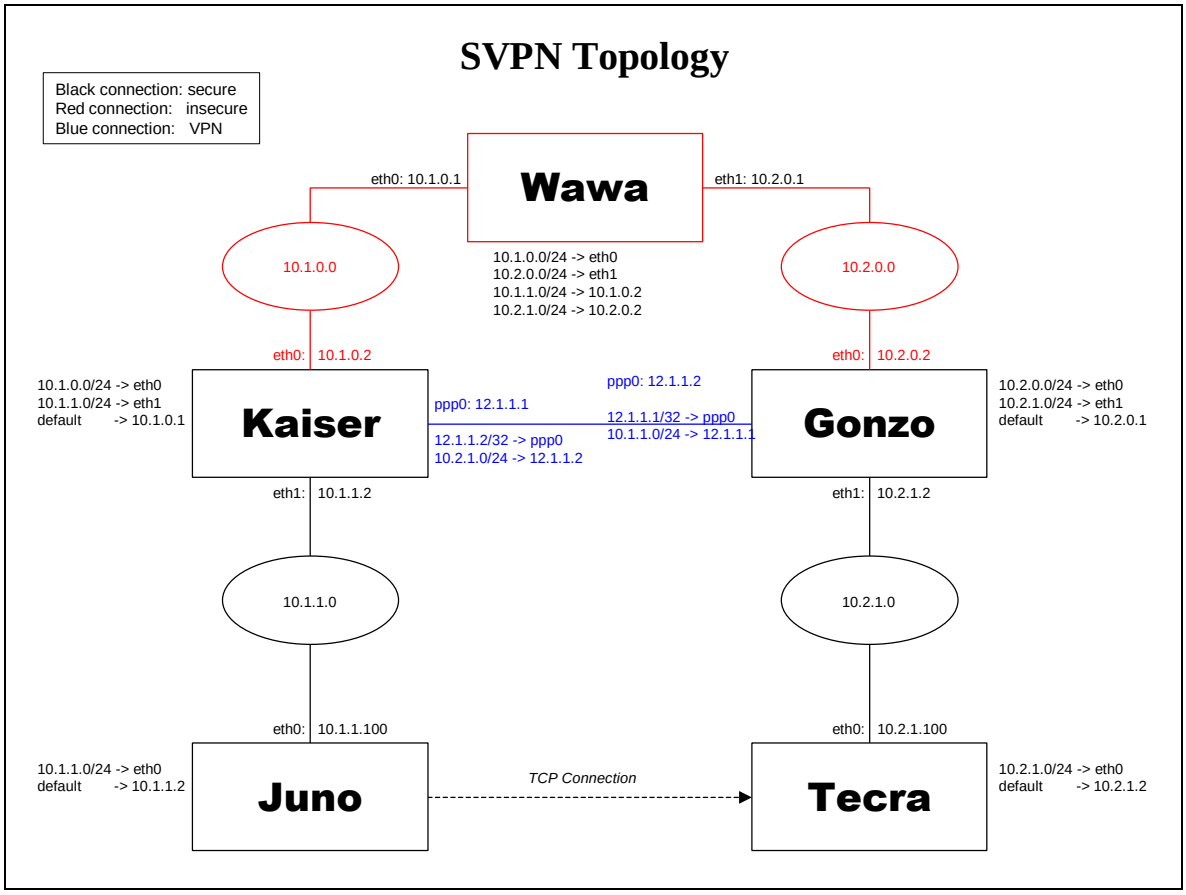

Figure 2: Secure Virtual Private Network

The connection from **Kaiser** to **Gonzo** via **Wawa** is now considered insecure and thus we installed our secure channel via *PPP* point-to-point interfaces connected via *SSH* from **Kaiser** to **Gonzo**. This *virtual* connection is shown in blue in Figure 2.

The IP packets from **Juno** to **Tecra** are now routed via 10.1.1.2, 12.1.1.1, 12.1.1.2, 10.2.1.2. The point-to-point connection from 12.1.1.1 to 12.1.1.2 is realized via *SSH*.

Thus, the “payload” TCP connection is tunneled over an existing TCP connection.

We made the same tests as for the “bare” test environment and recorded the results.

Results

The transfer rates, which were observed in the non SVPN test environment for “drop rates” above 10%, were far away from being continuous. Periods of activity were interrupted by long delays, in which no packets were transmitted. This non SVPN tests were aborted at a “drop rate” of 14% after 1 hour.

The observation in the SVPN environment were different!

Although we had periods of activity and inactivity, the delays between packet transmission phases were much shorter than before. Thus, we had a slow, but

continuos throughput. The tests for the SVPN environment were aborted at a “drop rate” of 25% after 1 hour.

Figure 3 shows the transfer time (in seconds) for both the “bare” test environment (no SVPN) and the TCP-over-TCP environment (SVPN), depending on the “drop rate” (in precent).

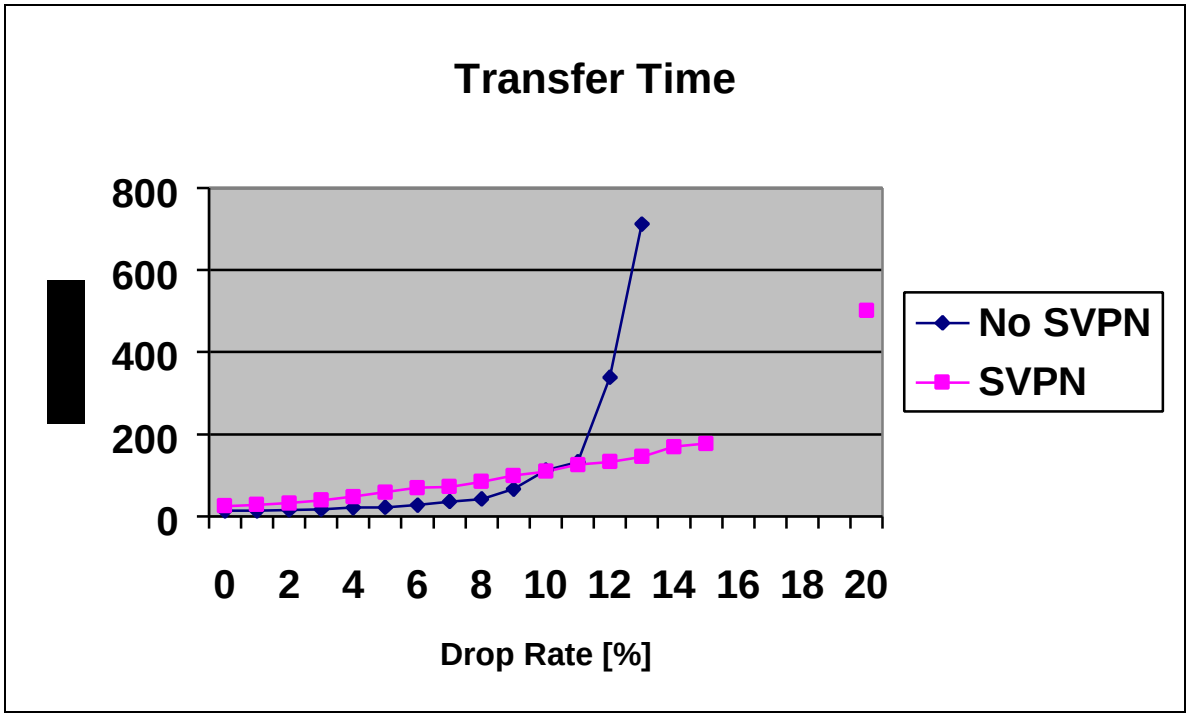

Figure 3: Transfer Time Results

Drop Rate	0	1	2	3	4	5	6	7	8	9	10
No SVPN	13	13	15,8	17,4	20,8	22,2	27	35,8	42,2	66,2	112,2
SVPN	25	28,2	32,2	38,6	47,4	58,8	69,6	72	84,4	98,8	109,4

Drop Rate	11	12	13	14	15	16	17	18	29	20
No SVPN	132	338,4	712	n/a	n/a	n/a	n/a	n/a	n/a	n/a
SVPN	125	132,4	144,8	169	177,2	n/a	n/a	n/a	n/a	501,4

Table 1: Transfer Time Results

Conclusion

The tests justified our assumption that TCP-over-TCP, at least the way we are using it, offers no penalties regarding stability and bandwidth.

More, the stacking of the two TCP layers seems to play a valuable role in that the upper layer “wakes up” the lower layer and thus shortens the periods of inactivity in an environment were packets get lost.

Appendix

We add the response of the author of the document mentioned above to justify our test environment. Our question is indented twice, the authors response is indented once.

```
Olaf Titz wrote:
>
> Hallo,
>
> > Natuerlich wir koennen Ihre Beobachtungen nicht einfach wegdiskutieren.
> > Uns ist klar, dass es in gewissem Umfang zu einem Aufschaukeln der
> > beiden TCP-Ebenen kommen wird. Laut RFC2001/2581 sollte sich allerdings
> > der Timeout der hoeheren Ebene auf ein hoeheres Niveau einstellen als
> > der der unteren Ebene - und so hoffentlich zu einem schwingungsfreien
> > Uebertragungssystem fuehren.
>
> Nach meiner Erfahrung ist das auch so lange unkritisch, wie auf der
> Netzwerkschicht keine wesentlichen Paketverluste auftreten. Dann
> erreicht die untere TCP-Schicht keine hohen Timeouts; die sind ja nur
> dann relevant, wenn mehrere Retries nacheinander verloren gehen (oder
> während der slow-start-Phase bei voller Sende-Queue, was allerdings
> die "Erholungsphase" verlängert).
>
> Ich habe die Beobachtungen, die zu dem Artikel führten, auf einer
> Anbindung mit einem Laserlink gemacht, der von Zeit zu Zeit sehr
> schlechte Verbindung mit hohen Paketverlustraten hatte. Auf einem
> stark belasteten internationalen Link dürfte der Effekt auch
> auftreten, auf einem LAN oder dedizierter Verbindung eher nicht.
>
> > Auch ist die Verwendung gestaffelter Retry-Mechanismen nichts Neues.
> > Einfachstes Beispiel: Modemuebertragung mit V42.bis und
> > zModem-Protokoll. Auch eine Netzwerkkarte macht auf der untersten
> > Hardwareschicht Retries und trotzdem laeuft TCP drueber.
>
> Diese Retry-Timeouts steigen aber nicht mit jedem Retry exponentiell
> an. Das ist die kritische Eigenschaft von TCP in diesem Zusammenhang.
> Die Warteschlange auf der unteren Ebene läuft dann voll, wenn die
> Timeouts auf der unteren Schicht schneller steigen als auf der oberen.
>
> Feste Timeouts oder kurze obere Grenzen in der unteren Schicht führen
> dagegen dazu, dass die obere Schicht genau so schnell wie im anderen
> Fall hohe RTOs bekommt, während die untere nach wie vor versucht, die
> Schlange schnell abzubauen. Soweit ich weiß hat kein anderes gängiges
> Protokoll ein RTO-Maximum von vier Minuten(!).
>
> Unter Umständen wäre es hier sinnvoll, in der _unteren_ TCP-Verbindung
> ein kurzes (im Sekundenbereich) maximales Timeout einzustellen.
> Das widerspricht aber RFC2581 und ist wohl deswegen üblicherweise auch
> nicht konfigurierbar.
>
> Ich hoffe, damit wird die Sache verständlich.
>
> M.f.G.
>
> Olaf Titz
```